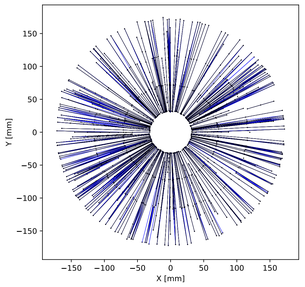
Accelerated Charged Particle Tracking with Graph Neural Networks on FPGAs

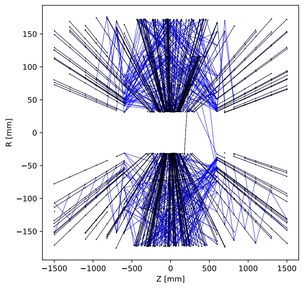
Aneesh Heintz[1], **Vesal Razavimaleki**[2], Javier Duarte[2], Gage DeZoort[3], Isobel Ojalvo[3], Savannah Thais[3], Markus Atkinson[4], Mark Neubauer[4], Lindsey Gray[5], Sergo Jindariani[5], Nhan Tran[5], Phil Harris[6], Dylan Rankin[6], Thea Aarrestad[7], Vladimir Loncar[7], Maurizio Pierini[7], Sioni Summers[7], Jennifer Ngadiuba[8], Mia Liu[9], Edward Kreinar[10], Zhenbin Wu[11]

[1]Cornell [2]UC San Diego [3]Princeton [4]UIUC [5]Fermilab [6]MIT [7]CERN [8]Caltech [9]Purdue [10]Hawkeye360 [11]UIC

Introduction

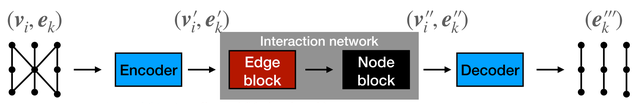
- Tracking approached as a graph "edge classification" problem
- Data represented as graphs
  - Nodes → Hits
  - Edges → Doublet connections
- Interest in GNN inference in FPGA-based trigger and co-processors to improve offline computational performance
- FPGA implementations of GNN segment classifiers explored using `hls4ml` and OpenCL
- `hls4ml`: compiler for physicists and ML experts to convert ML algorithms into FPGA firmware
- OpenCL: framework for writing programs that execute across heterogenous platforms (CPUs, GPUs, FPGAs, etc.)





Model Architectures

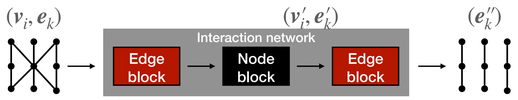
- `hls4ml` Implementation
  - Architecture: Exa.TrkX NeurIPS 2019 Segment Classifier
  - Encoder (edges/nodes): 4/3 → (8, 8)
  - Interaction Network (edge and node blocks): 8 → (8, 8)
  - Decoder (edges): 8 → (8, 8, 8, 1)



$e_k' = \phi_1^e(e_k)$ $e_k'' = \phi_2^e(e_k', v_{r_k}', v_{s_k}')$ $v_i'' = \phi_2^v(\bar{e}_i'', v_i')$ $e_k''' = \phi_3^e(e_k'')$

$v_i' = \phi_1^v(v_i)$ $\bar{e}_i'' = \rho^{e\to v}(E_i')$

- OpenCL Implementation
  - Architecture: Interaction Network
  - Edge block: 7 → (250, 250, 250, 1), node block: 4 → (200, 200, 3)

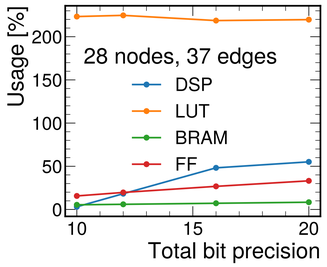


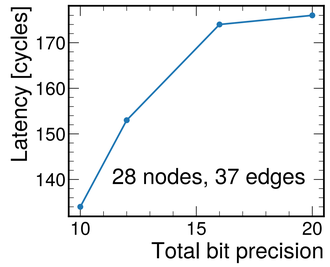
$e_k' = \phi_2^e(e_k, v_{r_k}, v_{s_k})$ $v_i' = \phi_2^v(\bar{e}_i', v_i)$ $e_k'' = \phi_2^e(e_k', v_{r_k}', v_{s_k}')$

$\bar{e}_i' = \rho^{e\to v}(E_i)$

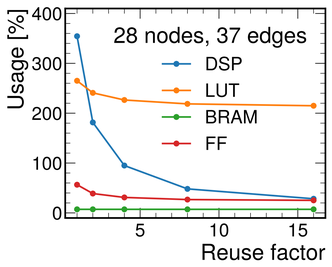
`hls4ml` Latency and Resources

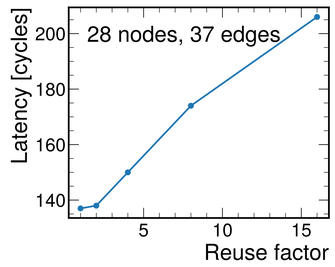
- Efficiency improvements for design targeting Xilinx KU115 FPGA:
  - Pipelining with reuse factor at edge/node block-level
  - Input streaming: implement incoming data as FIFO to recycle resources
  - Loop unrolling, zero-padding up to max. graph size
- For 1/64 of TrackML detector, $p_{\mathrm{T}_{\mathrm{min}}} = 2$ GeV (28 nodes, 37 edges at 95th percentile)
- Achieves latency of 650 ns to 1 μs
- Scan vs. bit precision show lower bit width results in smaller area, faster execution





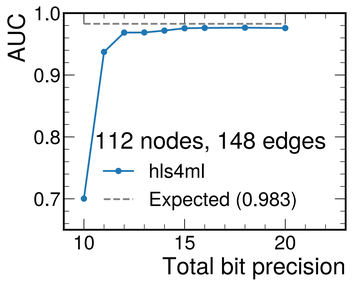
- Scan vs. reuse factor show trade-off between resource usage and latency





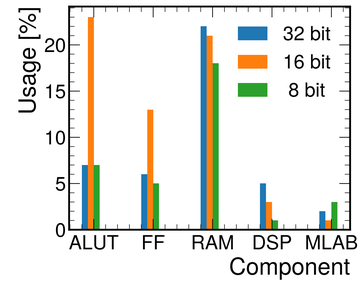
`hls4ml` Performance

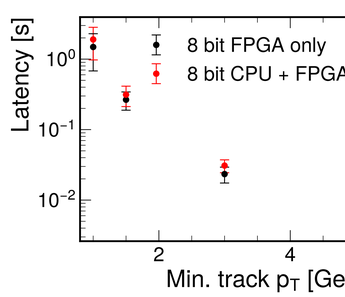
- GNN correctly classifies track segments with AUC ∼ 0.983
- AUC scan vs. fixed-point bit precision `<total,integer>` shows good performance for `<12,6>`
- For 1/16 of a TrackML detector, $p_{\mathrm{T}_{\mathrm{min}}} = 2$ GeV (112 nodes, 148 edges at 95th percentile)

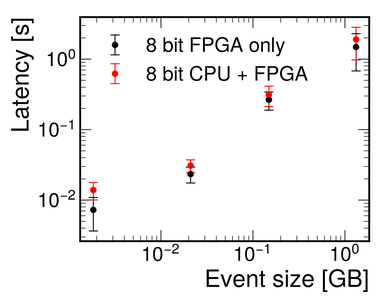


OpenCL Latency and Resources

- Efficiency improvements for design targeting Arria 10 GX 1150 FPGA:
  - 2D local memory tiling/register blocking: reduce redundancy/latency of reading off-chip memory
  - Double buffering: allow host to process/transfer data while kernel executes
  - Loop unrolling
- Scales up more easily to larger graph sizes (smaller $p_{\mathrm{T}_{\mathrm{min}}}$)
- Achieves latency of 10 ms to 1 s including CPU-FPGA I/O







Summary

- Two complementary implementations of GNNs on FPGAs
- Current performance promising for trigger-level applications (`hls4ml`) and co-processing applications (OpenCL and `hls4ml`)
- OpenCL implementation scales more easily to larger graphs while `hls4ml` implementation has latency/throughput advantage
- Future Work
  - Further detailed comparisons between the implementations based on the same model
  - Comparison with GPU co-processors
  - Additional optimizations such as quantization-aware training