arXiv:2012.01563v1 [physics.ins-det] 30 Nov 2020

FERMILAB-CONF-20-622-CMS-SCD

Accelerated Charged Particle Tracking with Graph Neural Networks on FPGAs

Aneesh Heintz[*]
Cornell Unviersity
Ithaca, NY 14850, USA

Vesal Razavimaleki*, Javier Duarte
University of California San Diego
La Jolla, CA 92093, USA

Gage DeZoort, Isobel Ojalvo, Savannah Thais
Princeton University
Princeton, NJ 08544, USA

Markus Atkinson, Mark Neubauer
University of Illinois at Urbana-Champaign
Champaign, IL 61820, USA

Lindsey Gray, Sergo Jindariani, Nhan Tran
Fermi National Accelerator Laboratory
Batavia, IL 60310, USA

Philip Harris, Dylan Rankin
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA

Thea Aarrestad, Vladimir Loncar[†], **Maurizio Pierini, Sioni Summers**
European Organization for Nuclear Research (CERN)
CH-1211 Geneva 23, Switzerland

Jennifer Ngadiuba
California Institute of Technology
Pasadena, CA 92115, USA

Mia Liu
Purdue University
West Lafayette, IN 47907, USA

Edward Kreinar
HawkEye360
Herndon, VA 20170, USA

Zhenbin Wu
University of Illinois at Chicago
Chicago, IL 60607, USA

Abstract

We develop and study FPGA implementations of algorithms for charged particle tracking based on graph neural networks. The two complementary FPGA designs are based on OpenCL, a framework for writing programs that execute across heterogeneous platforms, and `hls4ml`, a high-level-synthesis-based compiler for neural network to firmware conversion. We evaluate and compare the resource usage, latency, and tracking performance of our implementations based on a benchmark dataset. We find a considerable speedup over CPU-based execution is possible, potentially enabling such algorithms to be used effectively in future computing workflows and the FPGA-based Level-1 trigger at the CERN Large Hadron Collider.

[*]These two authors contributed equally.
[†]Also at Institute of Physics Belgrade, Belgrad, Serbia.

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

1 Introduction

In high energy physics (HEP), charged particle tracking [1, 2] is a crucial task necessary for the accurate determination of the kinematics of the particles produced in a collision event. The objective of tracking algorithms is to identify the trajectories of charged particles created in the collisions that bend in a magnetic field and ionize the material of detectors, providing position measurements along the trajectory of each particle. Current tracking algorithms [3–8] scale worse than quadratically in the number of hits, which are expected to increase dramatically at higher beam intensities. This motivates the study of alternative algorithms with different computational scaling. Another important consideration is the ability to accelerate these algorithms using highly-parallel heterogeneous computing resources like graphics processing units (GPUs) and field-programmable gate arrays (FPGAs) as further improvements in single-core CPU performance may be limited [9, 10]. Recent efforts [11, 12] have demonstrated the effectiveness of graph neural networks (GNNs) to correctly classify "segments" belonging to tracks. Graph-based approaches are well suited to this task because tracking data can be naturally encoded as a graph structure [13] and GNNs consider local information between pairs of hits to learn relationships between them in order to "connect the dots" to infer tracks.

In this paper, we accelerate a GNN [12] for segment classification, based on the interaction network (IN) architecture [14, 15], with FPGA implementations. Such implementations enable efficient processing, in both speed and energy consumption for large HEP datasets. They may also enable the use of GNNs in the high-throughput, FPGA-based data filter system, known as the Level-1 trigger [16–19], which has strict sub-microsecond latency requirements that only FPGAs or application-specific integrated circuits (ASICs) can meet. We design two complementary implementations using OpenCL [20] and `hls4ml` [21, 22], a specialized compiler for converting machine learning (ML) algorithms into FPGA firmware. We evaluate the resource usage, latency, and tracking performance of our implementations based on the benchmark TrackML dataset [1].

2 TrackML Data and Interaction Network Models

To benchmark the algorithms, we use the TrackML dataset [1], which consists of simulated 3D measurements of particles coming from independent collision events at the LHC. For our application, the data is embedded as a graph by considering the hits as nodes and pairs of hits on adjacent layers as edges. The edges corresponding to true track segments are labeled as one and all others are labeled as zero. The goal of the segment classifier algorithm is to correctly classify the edges as true or spurious.

We focus the scope of the task by considering the innermost layers of the detector, which usually correspond to the "pixel" detector of a modern HEP detector. We consider two subdetector regions in the endcaps, consisting of 7 layers each, and one in the barrel of the TrackML detector, consisting of 4 layers. Edges are constructed from hit pairs on adjacent layers satisfying $\Delta z < 15$ cm and $\Delta\phi/\Delta r < 2.62 \times 10^{-4}$. Particles with transverse momentum (p_T) greater than a given threshold are used to define the hits (nodes) and true edges. For the `hls4ml` model, we consider graphs corresponding to particle $p_\mathrm{T} > 2$ GeV, while for the OpenCL implementation we study the scaling as a function of p_T from 1 to 5 GeV. A characteristic graph for a single event is shown in Fig. 1 for $p_\mathrm{T} > 2$ GeV. The graph size depends strongly on the minimum p_T. For $p_\mathrm{T} > 1$ GeV (2 GeV), the event graph contains approximately 5,300 (1,100) nodes and 16,600 (1,500) edges on average.

Following the approach of Ref. [11, 12], we define a "segment classifier" using an IN model [14, 15] to learn which edges connect hits belonging to the same track. Relative to Ref. [12], the model architecture is simplified for the more limited task and the FPGA implementations. To implement it with `hls4ml`, the model shown in Figure 1 (lower left) is used. The same type of model is used for the OpenCL implementation, as shown in Figure 1 (lower right), without encoder and decoder networks and a different IN structure. In addition, while both models consider the 3 input node features (r, ϕ, z), the `hls4ml` model considers 4 input edge features $(\Delta r, \Delta\phi, \Delta z, \Delta R)$, while the OpenCL model does not consider any. Instead, in this model, the input edge features are zero vectors $\boldsymbol{e}_k = \mathbf{0}$, which can be thought of as the "initial guess" of the edge weights, i.e. all edges are initially assumed to be fake. Though we show results separately for these specific versions of the models, the `hls4ml` and OpenCL implementations we developed are modular and flexible enough to accommodate both models as well as other permutations. Appendix A demonstrates this by implementing a smaller version of the second model in `hls4ml`. Benchmarking the same model implemented in both frameworks is planned for future work.

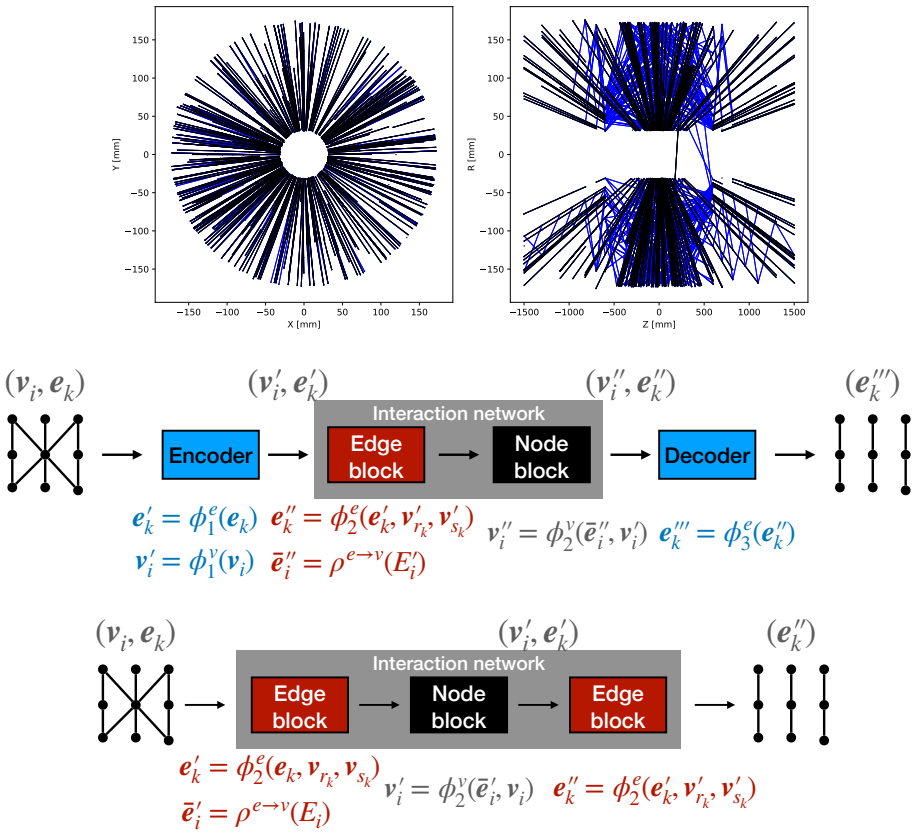


Figure 1: A characteristic graph of particles with $p_{\mathrm{T}} > 2$ GeV for one event in x–y view (upper left) and r–z view (upper right). The black edges correspond to true track segments, while the blue edges are spurious. GNN architectures used for the `hls4ml` (middle) and OpenCL (lower) implementations.

For the `hls4ml` model, an encoder composed of two neural networks, ϕ_1^e and ϕ_1^v, transforms input node and edge features into hidden representations. Both ϕ_1^e and ϕ_1^v have two layers with 8 neurons each and rectified linear unit (ReLU) [23, 24] activation functions. The IN is divided into two parts: an *edge block* (or *relational model*) and a *node block* (or *object model*). The edge block network ϕ_2^e takes as input a pair of node features with the corresponding edge features and has layers of sizes $(8, 8)$ with ReLU activation. The outputs of the ϕ_2^e are updated edge features, which can be considered "messages" sent between the nodes and aggregated at each node. The node block network ϕ_2^v takes as input the aggregated messages and the updated node features and consists of layers of sizes $(8, 8)$ with ReLU activation. Finally, the decoder network ϕ_3^v transforms edge features into an edge weight classifier (the probability for a given edge to be a true track segment), and has layers of sizes $(8, 8, 8, 1)$ with ReLU activation on all but the final layer that has a sigmoid activation.

For the model used in the OpenCL implementation, the IN begins with an edge block where ϕ_2^e has layers of sizes $(250, 250, 250, 1)$ with ReLU activations except for the final layer, which has a sigmoid activation. This is followed by a node block where ϕ_2^v has layers of sizes $(200, 200, 3)$ with ReLU activations except for the final layer. Subsequently, the same edge block is repeated to calculate the edge weights. The number of floating point operations (FLOPs) for both of these models scales with the number of nodes, number of edges, and size of the neural networks.

3 Implementations

OpenCL is an open-source C-based interface for parallel computing on various hardware platforms, including CPUs, GPUs, digital signal processors (DSPs), and FPGAs, using task and data-based parallelism [20]. The OpenCL implementation of the IN adopts a CPU-plus-FPGA coprocessing approach where the host program on the CPU manages the application, and all computational

operations are accelerated using dedicated kernels deployed on the FPGA that take capitalize on the device's hardware architecture to parallelize operations. The matrix multiplication kernel is repeatedly executed during a forward pass of the network and leverages the FPGA architecture for an efficient data-parallel implementation. This kernel uses both 2D local memory tiling and 2D register blocking to reduce the redundancy and latency of reading from globally shared off-chip memory. Because the input graph sizes to the network changes per event, the matrix multiplication kernels pad each matrix before computing the result. Throughout the forward pass, several optimizations are made to speed up computation. One example is the use of double buffering, which allows kernels to transfer data and execute instructions concurrently. All loops iterations executed in the OpenCL kernels are "unrolled" to run in parallel, which decreases the latency at the cost of increased hardware resource consumption. The OpenCL implementation is tested with an Intel Programmable Accelerator Card featuring an Arria 10 GX 1150 FPGA.

`hls4ml` [21] is an open-source converter of ML models into FPGA firmware utilizing Xilinx Vitis [25] high-level synthesis (HLS) [26] for both pure FPGA hardware applications and coprocessing kernels. In addition to implementing a GNN using available `hls4ml` tools, efforts are in progress to extend the compiler to support basic GNN building blocks. Each forward pass of IN is pipelined such that matrix multiplications are performed in parallel. Pipelining is performed at the level of the IN edge and node blocks and the amount of pipelining is tunable through the *reuse factor* parameter which controls the initiation interval (II) of each block. In conjunction with block-level pipelining, all loops are fully unrolled to decrease latency. All GNN model inputs are implemented with a streaming interface, which creates a FIFO that recycles its storage of array elements over each passage of a neural network block. Streaming decreases resource utilization but may increase latency. The input graph size is truncated to 112 nodes and 148 edges, which corresponds to the 95th percentile graph size for the η and ϕ sectors, and uses zero-padding to make the inputs a uniform size. To make scans of resources and timing more tractable, we further subdivide these graphs into quarters of up to 28 nodes and 37 edges just for the purposes of presentation. Tests of the `hls4ml` implementation target a Xilinx Kintex UltraScale (KU) 115 FPGA.

4 Results

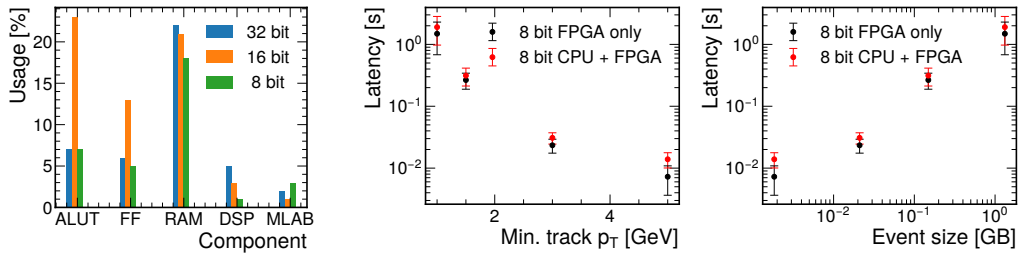


Figure 2: Resource system area analysis for the OpenCL implementation on an Arria 10 GX 1150 FPGA over multiple input data precision sizes (left). Scalability study in terms of latency versus minimum p_{T} (center) and event size (right).

For the OpenCL implementation, Fig. 2 (left) compares the resource usage for input data represented with 8-, 16-, and 32-bit floating point precision. For most components, operating with lower precision typically leads to lower resource usage, though the system area is dependent on other factors as well. Execution times also increase with increasing precision from 8 to 32 bits, but not substantially, especially for lower minimum p_{T}. Additionally, lower precision leads to smaller input data sizes, enabling the implementation to process events at lower p_{T}.

Figure 2 shows how the OpenCL implementation scales with minimum p_{T} (center) and input data size (right). Our implementation is able to process graphs with a minimum p_{T} as low as 1 GeV, which is limited by hardware storage, such as host CPU RAM and on-device local RAM. An increase in input data size increases the execution times. The largest bottlenecks are matrix multiplication operations and the overhead time that accompanies the enqueueing the kernels. Even though this overhead time increases as graph input size increases, enqueueing becomes more efficient. This scalability

study shows that the coprocessing approach in OpenCL is flexible with respect to data size and can handle significant increases in data sizes. However, we note that our CPU-FPGA coprocessing implementation is slower by about a factor of 10 from a pure CPU-based approach. In particular, CPU-based inference of the same model for $p_{\mathrm{T}} > 1$ (5) GeV graphs in PyTorch [27] is about 86 (2) ms.

For the `hls4ml` implementation, we first scan the fixed point precision total bit width to determine its impact on the physics performance of the algorithm as well as the latency and resource usage on the FPGA. We evaluate the receiver operating characteristic (ROC) curve for the segment classifier, and use the area under the curve (AUC) as a performance metric. Figure 3 (far left) shows the AUC as a function of the total bit precision, while the integer part is fixed to 6 bits. We see that with 12 total bits, we effectively reproduce the 32-bit floating point model. Figure 3 (center left) also shows the latency in clock cycles (for a 5 ns clock period) as a function of the total bit precision, which ranges from about 650 ns to 1 μs. For CPU-based inference of the same model, the latency is about 27 ms for the same sectorized $p_{\mathrm{T}} > 2$ GeV graph in the `graph_nets` framework [28] based on TensorFlow [29]. We also scan the reuse factor at constant fixed point precision of $\langle 16, 6 \rangle$, to study the resources and timing as a function of decreasing concurrency. Figure 3 shows the latency (center left) and resource usage estimates (far right) versus reuse factor. By construction, the II for the algorithm is equal to the reuse factor. We note that the HLS lookup table (LUT) usage tends to be overestimated, while the HLS DSP usage tends to be accurate [21, 30, 31]. Nonetheless, to make the algorithm fit on a single FPGA, larger reuse factors or bigger FPGAs may be necessary.

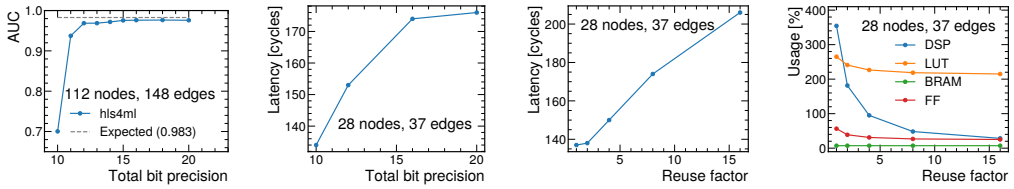


Figure 3: Segment classifier AUC versus fixed point precision total bit width (far left) for the `hls4ml` implementation. Latency in clock cycles for a 5 ns clock period (center left) as a function of the total bit width for the `hls4ml` implementation. The reuse factor (and thus II) is set to be 8. Latency (center right) and resource usage estimates (far right) relative to the available resources on a Xilinx KU 115 FPGA versus reuse factor for fixed total bit precision of $\langle 16, 6 \rangle$.

5 Summary and Outlook

We develop and study two complementary FPGA implementations of algorithms for charged particle tracking based on graph neural networks. The first, using OpenCL, targets CPU-FPGA coprocessing applications and achieves a latency of between 10 ms–1 s depending on the minimum p_{T} for full event graphs, including data transfer and I/O, for the model under consideration. The second, using `hls4ml`, targets both coprocessing and custom trigger (ultra low latency) applications and has an expected latency of 650 ns–1 μs, considering only the execution time on the FPGA, for smaller, sectorized input graphs and a more compact model, although more work is needed to reduce the resource consumption. Compared to CPU-based execution of the same models, the speedup for the `hls4ml` implementation is considerable, but further optimizations are needed to improve on the CPU latency for the OpenCL implementation. Continued development in this direction may allow such algorithms to be used effectively in future computing workflows [32] and the Level-1 trigger at the LHC. In future work, we plan to study detailed comparisons of the two implementations based on the same model, as well as comparing to GPU coprocessors [33]. Other optimizations of the GNN model may also be possible, such as more efficient architectures [30] and use of quantization-aware training [31, 34] to reduce the necessary precision.

Broader Impact

This work may be used to accelerate particle tracking and other reconstruction algorithms in high energy physics experiments. While accelerated machine learning on FPGAs has many potential

benefits to science and society, including high-quality, fast data reconstruction and selection in experiments, automated detector control, or smarter IoT devices, it may also be used for nefarious purposes, such as surveillance or military unmanned aerial vehicles.

Acknowledgments and Disclosure of Funding

We acknowledge the Fast Machine Learning collective as an open community of multi-domain experts and collaborators. This community was important for the development of this project. The simulations presented for the OpenCL implementation in this article were performed on computational resources managed and supported by Princeton Research Computing, a consortium of groups including the Princeton Institute for Computational Science and Engineering (PICSciE) and the Office of Information Technology's High Performance Computing Center and Visualization Laboratory at Princeton University. Work for the `hls4ml` implementation was partially performed on the Pacific Research Platform Nautilus HyperCluster supported by NSF awards CNS-1730158, ACI-1540112, ACI-1541349, OAC-1826967, the University of California Office of the President, and the University of California San Diego's California Institute for Telecommunications and Information Technology/Qualcomm Institute. Thanks to CENIC for the 100 Gpbs networks.

A. H., V. R., and S. T. are supported by IRIS-HEP through the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-1836650. J. D. is supported by the U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187. T. A., V. L., M. P., and S. S. are supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 772369). L. G., S. J., and N. T. are supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics. P. H. is supported by a Massachusetts Institute of Technology University grant. Z. W. is supported by the National Science Foundation under Grants No. 1606321 and 115164.

A Alternative GNN Model in `hls4ml`

In addition to the first model shown in Fig. 1, we also implement the second model in `hls4ml`. In this model, the same architecture is used as the one implemented in OpenCL, except the neural network sizes are reduced: the ϕ_2^e network has layers of sizes $(8, 8, 8, 1)$ and ϕ_2^v has layers of sizes $(8, 8, 3)$. Figure 4 shows the AUC (far left) and the latency in clock cycles for a 5 ns clock period (center left) as a function of the total bit precision, while the integer part is fixed to 6 bits. We see that above 16 total bits, the quantized model effectively reproduces the 32-bit floating point model. Figure 4 also shows the latency (center left) and resource usage estimates (far right) versus reuse factor at a constant fixed point precision of $\langle 16, 6 \rangle$. This small implemented model reasonably fits on a single FPGA with a latency less than 1 μs.

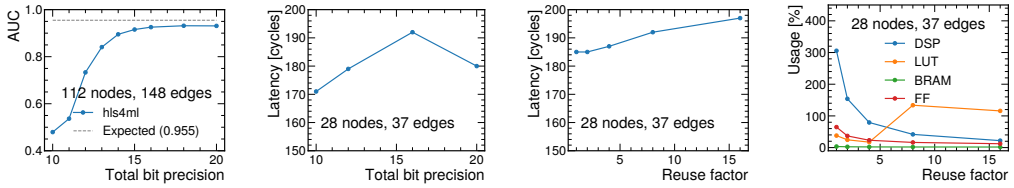


Figure 4: Alternative GNN segment classifier model implemented in `hls4ml`. AUC (far left) and latency (center left) in clock cycles for a 5 ns clock period as a function of the total bit width. The reuse factor (and thus II) is set to be 8. Latency (center right) and resource usage estimates (far right) relative to the available resources on a Xilinx KU 115 FPGA versus reuse factor for fixed total bit precision of $\langle 16, 6 \rangle$.

References

[1] S. Amrouche et al., "The tracking machine learning challenge: Accuracy phase", in *The NeurIPS '18 Competition*, S. Escalera and R. Herbrich, eds., p. 231. Springer, Cham, Switzerland, 2020. arXiv:1904.06778. doi:10.1007/978-3-030-29135-8_9.

[2] A. Strandlie and R. Frühwirth, "Track and vertex reconstruction: From classical to adaptive methods", *Rev. Mod. Phys.* **82** (2010) 1419, doi:10.1103/RevModPhys.82.1419.

[3] CMS Collaboration, "Description and performance of track and primary-vertex reconstruction with the CMS tracker", *J. Instrum.* (2014) P10009, doi:10.1088/1748-0221/9/10/P10009, arXiv:1405.6569.

[4] ATLAS Collaboration, "Performance of the ATLAS track reconstruction algorithms in dense environments in LHC Run 2", *Eur. Phys. J. C* **77** (2017) 673, doi:10.1140/epjc/s10052-017-5225-7, arXiv:1704.07983.

[5] P. Billoir, "Progressive track recognition with a Kalman-like fitting procedure", *Comput. Phys. Comm.* **57** (1989) 390, doi:10.1016/0010-4655(89)90249-X.

[6] P. Billoir and S. Qian, "Simultaneous pattern recognition and track fitting by the Kalman filtering method", *Nucl. Instrum. Methods Phys. Res. A* **294** (1990) 219, doi:10.1016/0168-9002(90)91835-Y.

[7] R. Mankel, "A concurrent track evolution algorithm for pattern recognition in the hera-b main tracking system", *Nucl. Instrum. Methods Phys. Res. A* **395** (1997) 169, doi:10.1016/S0168-9002(97)00705-5.

[8] R. Frühwirth, "Application of Kalman filtering to track and vertex fitting", *Nucl. Instrum. Methods Phys. Res. A* **262** (1987) 444, doi:10.1016/0168-9002(87)90887-4.

[9] H. Esmaeilzadeh et al., "Dark silicon and the end of multicore scaling", in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, p. 365. ACM, New York, NY, USA, 2011. doi:10.1145/2000064.2000108.

[10] R. H. Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", *IEEE J. Solid-State Circuits* **9** (1974) 256, doi:10.1109/JSSC.1974.1050511.

[11] S. Farrell et al., "Novel deep learning methods for track reconstruction", in *4th International Workshop Connecting The Dots 2018*. 2018. arXiv:1810.06111.

[12] X. Ju et al., "Graph neural networks for particle reconstruction in high energy physics detectors", in *Machine Learning and the Physical Sciences Workshop at the 33rd Annual Conference on Neural Information Processing Systems*. 2019. arXiv:2003.11603.

[13] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics", doi:10.1088/2632-2153/abbf9a, arXiv:2007.13681. Accepted by *Mach. Learn.: Sci. Tech.*

[14] P. W. Battaglia et al., "Interaction networks for learning about objects, relations and physics", in *Advances in Neural Information Processing Systems*, volume 29, p. 4502. 2016. arXiv:1612.00222.

[15] P. W. Battaglia et al., "Relational inductive biases, deep learning, and graph networks", arXiv:1806.01261.

[16] ATLAS Collaboration, "Operation of the ATLAS trigger system in Run 2", *J. Instrum.* **15** (2020) P10004, doi:10.1088/1748-0221/15/10/P10004, arXiv:2007.12539.

[17] ATLAS Collaboration, "Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System", ATLAS Technical Design Report CERN-LHCC-2017-020. ATLAS-TDR-029, 2017.

[18] CMS Collaboration, "Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV", *J. Instrum.* **15** (2020) P10017, doi:10.1088/1748-0221/15/10/P10017, arXiv:2006.10165.

[19] CMS Collaboration, "The Phase-2 upgrade of the CMS Level-1 trigger", CMS Technical Design Report CERN-LHCC-2020-004. CMS-TDR-021, 2020.

[20] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems", *Comput. Sci. Eng.* **12** (2010) 66, doi:10.1109/MCSE.2010.69.

[21] J. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics", *J. Instrum.* **13** (2018) P07027, doi:10.1088/1748-0221/13/07/P07027, arXiv:1804.06913.

[22] V. Loncar et al., "fastmachinelearning/hls4ml: aster", 10, 2020. doi:10.5281/zenodo.4161550, https://github.com/fastmachinelearning/hls4ml.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines", in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, p. 807. Omnipress, Madison, WI, USA, 2010.

[24] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, eds., volume 15 of *Proceedings of Machine Learning Research*, p. 315. JMLR, Fort Lauderdale, FL, USA, 4, 2011.

[25] Xilinx, Inc., "Vivado design suite user guide: High level synthesis", 2020. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug902-vivado-high-level-synthesis.pdf.

[26] D. O'Loughlin et al., "Xilinx vivado high level synthesis: Case studies", in *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET Year*, p. 352. 2014. doi:10.1049/cp.2014.0713.

[27] A. Paszkeet al., "PyTorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems*, H. Wallach et al., eds., volume 32, p. 8024. Curran Associates, Inc., 2019. arXiv:1912.01703.

[28] DeepMind, "graph_nets", 2019. https://github.com/deepmind/graph_nets.

[29] M. Abadiet al., "TensorFlow: Large-scale machine learning on heterogeneous systems", 2015. https://www.tensorflow.org.

[30] Y. Iiyama et al., "Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics", doi:10.3389/fdata.2020.598927, arXiv:2008.03601. Accepted by *Front. Big Data*.

[31] G. Di Guglielmo et al., "Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml", doi:10.1088/2632-2153/aba042, arXiv:2003.06308. Accepted by *Mach. Learn.: Sci. Technol.*

[32] D. S. Rankin et al., "FPGAs-as-a-service toolkit (FaaST)", in *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. 2020. arXiv:2010.08556.

[33] J. Krupa et al., "GPU coprocessors as a service for deep learning inference in high energy physics", (2020). arXiv:2007.10359. Submitted to *Mach. Learn.: Sci. Technol.*

[34] C. N. Coelho et al., "Automatic deep heterogeneous quantization of deep neural networks for ultra low-area, low-latency inference on the edge at particle colliders", arXiv:2006.10159. Submitted to *Nat. Mach. Intell.*