Computing and Software for Big Science (2024) 8:8
https://doi.org/10.1007/s41781-024-00117-0

RESEARCH

Real-Time Graph Building on FPGAs for Machine Learning Trigger Applications in Particle Physics

Marc Neu[1] · Jürgen Becker[1] · Philipp Dorwarth[2] · Torben Ferber[2] · Lea Reuter[2] · Slavomira Stefkova[2] · Kai Unger[1]

Received: 14 July 2023 / Accepted: 11 March 2024

Abstract
We present a design methodology that enables the semi-automatic generation of a hardware-accelerated graph building architectures for locally constrained graphs based on formally described detector definitions. In addition, we define a similarity measure in order to compare our locally constrained graph building approaches with commonly used k-nearest neighbour building approaches. To demonstrate the feasibility of our solution for particle physics applications, we implemented a real-time graph building approach in a case study for the Belle II central drift chamber using Field-Programmable Gate Arrays (FPGAs). Our presented solution adheres to all throughput and latency constraints currently present in the hardware-based trigger of the Belle II experiment. We achieve constant time complexity at the expense of linear space complexity and thus prove that our automated methodology generates online graph building designs suitable for a wide range of particle physics applications. By enabling an hardware-accelerated preprocessing of graphs, we enable the deployment of novel Graph Neural Networks (GNNs) in first-level triggers of particle physics experiments.

Keywords Graph building · Graph neural networks · Field programmable gate arrays · Particle physics · Machine learning · Nearest neighbour · Belle II

Introduction

Machine Learning is widely used in particle physics for various reconstruction tasks and Graph Neural Networks (GNNs) are recognised as one possible solution for irregular geometries in high energy physics. GNNs have proven suitable for jet clustering [1], calorimeter clustering [2], particle track reconstruction [3–5], particle tagging [6, 7] and particle flow reconstruction [8]. However, all applications described above are implemented in an offline environment, relying on high performance computing clusters utilising Central Processing Units (CPUs) and Graphics Processing Units (GPUs) to achieve the required throughput for the analysis of collision events. Therefore, existing implementations are not suitable for real-time particle tracking and reconstruction in trigger systems of particle detectors.

The realisation of GNNs on FPGAs for particle tracking is an active area of research [4, 9–11]. Due to latency and throughput constraints, a suitable implementation meeting all requirements imposed by particle physics experiments is yet to be developed. Especially the generation of input graphs under latency constraints is a challenge that has not received full attention so far in the evaluation of existing

✉ Marc Neu
marc.neu@kit.edu

Jürgen Becker
juergen.becker@kit.edu

Philipp Dorwarth
philipp.dorwarth@student.kit.edu

Torben Ferber
torben.ferber@kit.edu

Lea Reuter
lea.reuter@kit.edu

Slavomira Stefkova
slavomira.stefkova@kit.edu

Kai Unger
kai.unger@kit.edu

[1] Institut fuer Technik der Informationsverarbeitung (ITIV), Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

[2] Institute of Experimental Particle Physics (ETP), Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

prototypes. Current prototypes as described in [4, 9] are trained on preprocessed graph datasets, taking into account geometric properties of detectors. However, a holistic implementation of GNNs for triggers requires the consideration of the entire data flow chain. This raises the question on how to build graphs under latency constraints in high-throughput particle physics applications.

In our work, we consider constraints from currently operating first level trigger systems [12–14]: event processing rates in the order of 10 to 100 MHz and latencies in the order of 1 to 10 μ s render the utilisation of compound platforms based on CPUs and Field Programmable Gate Arrays (FPGAs) used in other research areas infeasible [15, 16]. For example, the typical transmission latency between CPU and FPGA on the same chip is already larger than 100ns, making up a considerable processing time [17].

To overcome the research gap, our work comprises the following contributions: first, we outline existing nearest neighbour graph-building methods and evaluate their feasibility for trigger applications. Second, we develop a methodology to transform formal graph-building approaches to hardware-accelerated processing elements in an automated way. Third, we evaluate our proposed toolchain on the Belle II central drift chamber (CDC), demonstrating the feasibility of our solution to build graphs under the constraints imposed by current trigger systems.

The paper is organised as follows: in sect "Related Work", we give an overview of related work on FPGA-accelerated graph building. The CDC, the event simulation and details of the beam background simulation are described in sect "Simulation and Dataset". The methodology for transforming discrete sensor signals into a graphical representation is discussed in sect. "Graph Building". The procedure for implementing real-time graph building in hardware is described in sect. "Toolchain". A concrete example of real-time graph building for the Belle II CDC is provided in section . We summarise our results in sect. "Conclusion".

Related Work

Previous work on FPGA-accelerated GNNs for particle physics utilise input graphs based on synchronous sampled collision events as input for training and inference of the respective networks [4, 18]. Early studies made use of fully connected graphs which lead to scalability challenges for detectors with more than 10 individual sensors [19]. Typical particle physics trigger systems have much higher number of sensors though (see Table 1).

Aiming to significantly reduce the maximum size of input graphs, the geometric arrangement of sensors in the detector has been considered recently [3, 5]. Nevertheless, input graphs are currently generated offline, stored in the FPGA memory and are accessed over AXI[1]-Mapped Memory interfaces in prototype implementations [9]. However, as sensors in detectors are read out as individual channels without providing relational information, the processing of input graphs must be considered as part of the critical path in online track reconstruction and trigger algorithms.

Table 1 Input parameters for the first-level trigger systems in three current particle physics detectors

	CMS [9, 22]	Belle II [20]	DUNE [21]
Subsystem	Muon	CDC	ProtoDune SP
Number of sensors	6500	14,336	15,360
Trigger data input rate	40 MHz	32 MHz	2 MHz

For CMS, 95 % quantiles for the number of sensor hits per event is reported in [9], while for the Belle II CDC [20] and DUNE [21] the number of sensors inputs is given

While building suitable input graphs for neural networks is a rather recent application, general nearest neighbour (NN) graph building has been studied extensively in literature [23–25]. In order to reduce the computational demand of NN graph-building algorithms, continuous efforts have been made towards building approximate graphs making use of local sensitive hashing [26, 27], backtracking [28], or small world graphs [29]. Performance improvements from these algorithms have been demonstrated for applications targeting high-dimensional graphs containing more than 10^6 vertices such as database queries [30]. There are two key challenges that limit the generalisation of these techniques in the particle physics trigger context. First, k-nearest neighbour (k-NN) algorithms inherently rely on sequential processing and present challenges in efficient parallelisation. Second, while there is a wide range of graph-processing frameworks available (see Ref. [31] for a survey on graph processing accelerators), none of them meet the stringent latency and throughput requirements of current particle physics trigger systems: FFNG [32] focuses on the domain of high-performance computing and therefore does not impose hard real-time constraints. GraphGen [33] relies on external memory controllers which introduce additional latency into the system. GraphACT [16, 34] utilises preprocessing techniques on CPU-FPGA compound structures in order to optimise throughput and energy efficiency which again introduces non-determinism and additional latency. And lastly, current GNN accelerators like HyGCN [35] or AWB-GCN [36] use the previously described techniques to reduce the required system bandwidth and improve the

[1] AXI: Advanced eXtensible Interface, is an on-chip communication bus protocol.

energy efficiency of the inference. They are therefore not suitable for particle physics applications.

Simulation and Dataset

In this work, we use simulated Belle II events to benchmark the graph-building algorithms. The detector geometry and interactions of final state particles with the material are simulated using GEANT4 [37], which is combined with the simulation of a detector response in the Belle II Analysis Software Framework [38]. The Belle II detector consists of several subdetectors arranged around the beam pipe in a cylindrical structure that is described in detail in Refs. [39, 40]. The solenoid's central axis is the z-axis of the laboratory frame. The longitudinal direction, the transverse xy plane with azimuthal angle ϕ, and the polar angle θ are defined with respect to the detector's solenoidal axis in the direction of the electron beam. The CDC consists of 14336 sense wires surrounded by field wires which are arranged in nine so-called superlayers of two types: axial and stereo superlayers. The stereo superlayers are slightly angled, allowing for 3D reconstruction of the track. In the simulated events, we only keep the detector response of the CDC.

We simulated two muons (μ^+,μ^-) per event with momentum $0.5 < p < 5\,\text{GeV/c}$, and direction $17° < \theta < 150°$ and $0° < \phi < 360°$ drawn randomly from independent uniform distributions in p, θ, and ϕ. The generated polar angle range corresponds to the full CDC acceptance. Each of the muons is displaced from the interaction point between 20 cm and 100 cm, where the displacement is drawn randomly from independent uniform distributions.

As part of the simulation, we overlay simulated beam background events corresponding to instantaneous luminosity of $\mathcal{L}_{\text{beam}} = 6.5 \times 10^{35}\ \text{cm}^{-2}\text{s}^{-1}$ [41, 42]. The conditions we simulate are similar to the conditions that we expect to occur when the design of the experiment reaches its ultimate luminosity.

An example of an event display for a physical event $e^+e^- \rightarrow \mu^+\mu^-(\gamma)$ is shown in Fig. 1. It is visible that the overall hit distribution of the exemplary event is dominated by the simulated beam background signal.

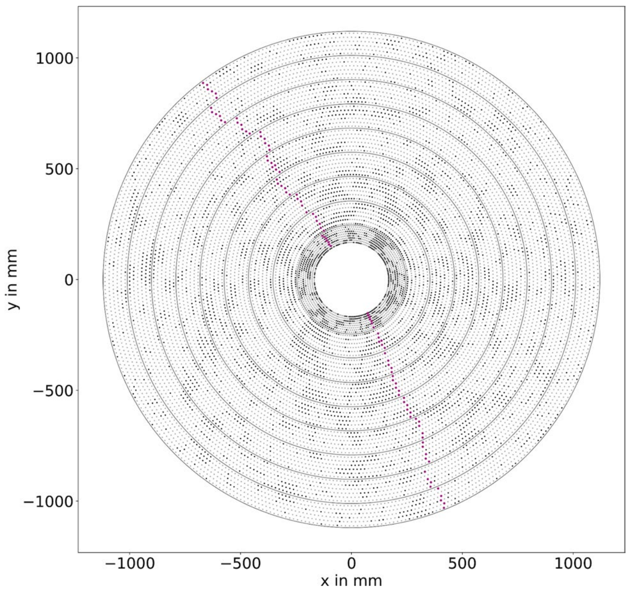


Fig. 1 Typical event display showing the transverse plane of the Belle II CDC. Hits generated by signal muon particles are shown with purple markers and background hits by black markers

Graph Building

This work proposes a methodology for transforming discrete sensor signals captured inside a particle detector into a graphical representation under real-time constraints. Particular importance is given to the use-case of particle physics trigger algorithms, adhering to tight latency constraints in the sub-microsecond timescale.

Current large-scale particle detectors are composed of various discrete sensors and often, due to technical limitations, placed heterogeneously inside the system. For this reason, signals from the sensors cannot be considered regularly distributed, as it is the case with, for example, monolithic image sensors. In the following, a detector D is defined as a set of N discrete sensors $\{\vec{s}_1, ..., \vec{s}_N\}$, where each individual sensor $\vec{s}_i$ is described by a feature vector of length f. Some examples for described features are the euclidean location inside the detector, the timing information of the received signal, or a discrete *hit identifier*. To map relational connections between individual sensors, a graph based on the detector description is generated which contains the respective sensor features.

Formally described, a graph building algorithm generates an non-directional graph $G(D, E)$, where D is the set of vertices of the graph, and $E \subseteq D \times D$ is the set of edges. The set of vertices is directly given by the previously described set of sensors in a detector. Each edge $e_{ij} = e(\vec{s}_i, \vec{s}_j) \in E$ with $\vec{s}_i, \vec{s}_j \in D$ in the graph connects two sensors based on a building specification, that depends on sensor features. In the following, we consider the case of building non-directed graphs. We do not introduce any fundamental restrictions that limit the generalisation of our concept to directed graphs.

In general, graph building approaches are tailored to the specific detector and physics case. We consider three approaches that can be classified into two classes of nearest-neighbour graph building: locally constrained graphs, and locally unconstrained graphs.

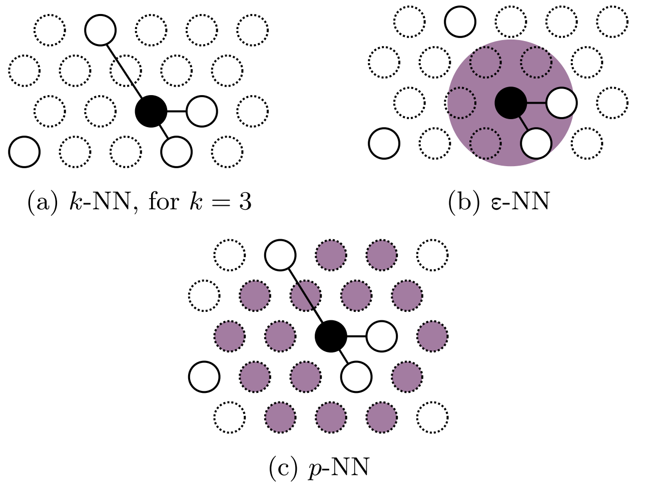


Fig. 2 Example for the three different approaches of building nearest neighbour graphs. Sensors inside a detector are depicted as circles. A sensor which is hit by a particle is identified by a solid outline, those without a hit by a dotted outline. The query vertices are depicted in black. Edges connecting two nearest neighbours are indicated by a solid line. Nodes filled with purple are considered candidate sensors, which are part of the specified search pattern around the query vertex

Figure 2 depicts an exemplary cut-out of a detector, in which sensors are placed heterogeneously in two-dimensional space. For simplicity, sensors are aligned in a grid-like structure without restricting the generality of our graph-building approach. A graph is built for a query vertex which is depicted by a solid black circle. We use the exemplary query vertex to illustrate NN-graph building on a single vertex for simplicity. In the following, we compare the three building approaches and explain their differences.

k-NN

k-NN graph building is illustrated on a single query node in Fig. 2a. Repeating the building algorithm sequentially leads to a worst-case execution time complexity of $\mathcal{O}(k|D|\log(|D|)$ [23]. To reduce the execution time, parallelisation of the algorithm has been studied in Ref. [24], achieving a lower theoretical complexity. Based on the optimisation, a linear $\mathcal{O}(|D|)$ time complexity is achieved in experimental evaluation [25]. Nevertheless, substantial processing overhead and limitations through exclusive-read, exclusive-write memory interfaces limit the usability for trigger applications. To achieve a higher degree of parallelisation, algorithms as described in Refs. [27, 28] make use of locally constrained approximate graphs.

ϵ-NN

ϵ-NN graph building is illustrated on a single query node in Fig. 2b. The parameter ϵ defines an upper bound for the distance of a candidate vertex from the query vertex. All vertices for which Eq. (1) holds true are connected in a graph, yielding a locally constrained graph. Figuratively, a uniform sphere is placed over a query point joining all edges which are inside the sphere into the graph:

$$d(\vec{x}_i, \vec{x}_j) = \|\vec{x}_i - \vec{x}_j\|_2 < \epsilon. \tag{1}$$

Since the ϵ-NN approach is controlled by only one parameter, it is a general approach to building location-constrained graphs. However, variations between adjacent sensors in heterogeneous detectors are not well represented in the ϵ-NN algorithm.

p-NN

Pattern nearest-neighbour (p-NN) graph building is illustrated on a single query node in Fig. 2c. For building the graph, every candidate sensor is checked and, if the predefined condition $p(\vec{x}_i, \vec{x}_j)$ in Eq. (2) is fulfilled, the edge between candidate node and query node is included in the graph:

$$p(\vec{x}_i, \vec{x}_j) \Rightarrow \text{True}. \tag{2}$$

Comparison

When comparing the k-NN, the ϵ-NN and the p-NN algorithms, it is obvious that in general all three approaches yield different graphs for the same input set of sensors. The p-NN building and the ϵ-NN building can both be considered locally constrained algorithms with differing degrees of freedom. While ϵ-NN building maps the locality into exactly one parameter, the definition of the p-NN building offers more flexibility. In contrast, the k-NN approach differs as outliers far away from a query point might be included. Nevertheless it is noted in Ref. [43], that on a uniformly distributed dataset a suitable upper bound ϵ^* exists, for which the resulting ϵ-NN graph is a good approximation of corresponding k-NN graph.

Toolchain

In the following, we leverage the described mathematical property to demonstrate the feasibility of building approximate k-NN graphs for trigger applications. First, we provide a methodology to evaluate the approximate equivalence of k-NN, ϵ-NN and p-NN graph building approaches, providing a measure of generality for k-NN parameters chosen in offline track reconstruction algorithms [3, 21]. Second, we semi-automatically generate a generic hardware implementation for the p-NN graph building, thus demonstrating the feasibility of graph-based signal processing in low-level trigger systems.Since ϵ-NN graph building is a special case of *p*-NN graph building, we have also covered this case in our implementation. Third, we perform a case study on the Belle II trigger system demonstrating achievable throughput and latency measures in the environment of trigger applications.

Hardware Generator Methodology

Algorithms that generate graphs by relating multiple signal channels belong to the domain of digital signal processing. As such they share characteristics of typical signal processing applications like digital filters or neural networks. Both applications are data-flow dominated and require a large number of multiply-and-accumulate operators and optimisations for data throughput. Thus, implementing these algorithms on FPGAs improves latency and throughput in comparison to an implementation on general purpose processors [44].

Various high-level synthesis (HLS) frameworks have been developed to reduce the required design effort such as FINN [45, 46] and HLS4ML [47, 48] with which the realisation of the GarNet, a specific GNN architecture, is possible. Although these frameworks offer a low entry barrier for the development of FPGA algorithms, they are unsuitable for the implementation of our graph building concept.

Therefore, we propose a generator-based methodology enabling to transform a graph building algorithm into an actual firmware implementation, that grants us complete design freedom at the register transfer level. Figure 3 illustrates our development flow for both the generation of an intermediate representation of the circuit and an algorithmic evaluation of the building approach. As an input a database containing the formal definition of a detector is expected alongside hyperparameters, e.g. ϵ for the ϵ-NN graph building. Based on the selected approach, an intermediate-graph representation is generated, containing informationon how the building approach is mapped onto the detector. The intermediate-graph representation serves as an input for the hardware generation and the algorithmic evaluation.

On one side, an intermediate-circuit representation is generated by combining the intermediate-graph representation and parameterised hardware modules from our hardware description language (HDL) template library. The template library contains the elementary building blocks required to implement online graph building, in particular the static routing network, the edge processing elements and interface definitions. We use Chisel3 [49] as hardware-design language providing an entry point to register transfer-level circuit designs in Scala.

On the other side, the intermediate-graph representation is evaluated on a user-defined dataset and compared to a

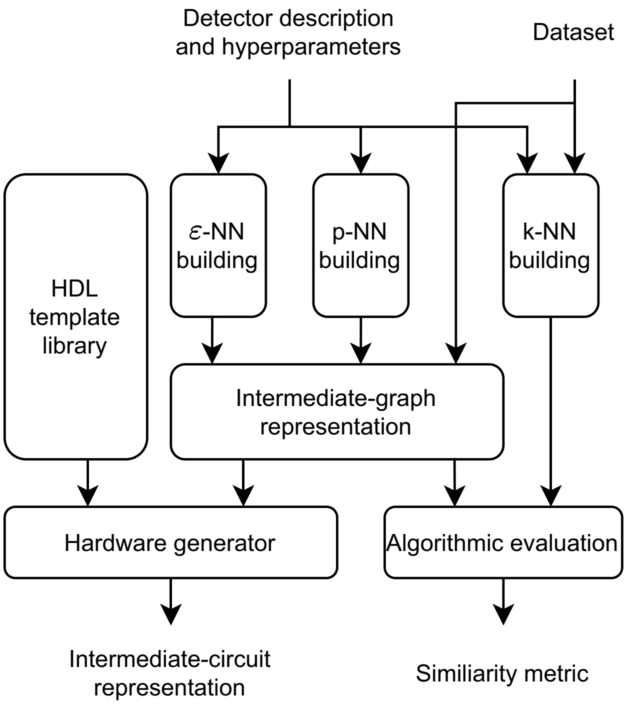


Fig. 3 Proposed generator-based methodology for our graph building approach. On the left side, the development flow for the hardware implementation is depicted, yielding an intermediate hardware representation. On the right side, flow for the algorithmic evaluation of the algorithms is shown

generic *k*-NN graph-building approach. To achieve a quantitative comparison, we introduce similarity metrics for different operating conditions in the detector in section . This result can be used to iteratively adapt hyperparameters in the ϵ-NN or *p*-NN approach, improving the similarity to *k*-NN graphs that are often used in offline track reconstruction.

Intermediate-Graph Representation

The parameter ϵ in the ϵ-NN approach and the pattern function in the *p*-NN approach limit the dimensionality of the graph under construction. In comparison to fully connected graphs, the maximum number of edges is lowered by imposing local constraints on the connectedness of sensors in the detector. Local constraints are implemented by considering the influence of static sensor features, like euclidean distances between sensors, during design time of the FPGA firmware. Leveraging the a priori knowledge of the sensor position, the computational effort required during online inference of the algorithm is lowered.

Algorithm 1 describes the procedure to derive the intermediate-graph representation of an arbitrary graph-building procedure. As an input the formally described set of sensors D is given. Iterating over every sensor in the detector, the locality of not yet visited sensors is checked by a user-defined *metric* describing the graph building approach. If a sensor is considered to be in the neighbourhood of another sensor, the connection is added to the resulting set of edge candidates E. All edges in E must be checked for their validity during the inference of the online graph building.

The combination of the formal detector description and the set of candidate edges is sufficient to describe an arbitrary building approach on non-directed graphs. According to algorithm 1, the worst-case time complexity during design-time amounts to $\mathcal{O}(|D|^2)$, which is higher than the worst-case time-complexity of state-of-the-art k-NN building approaches. However, the worst-case time-complexity during run-time is now only dependent on the number of identified edges during design-time. Therefore, generating a graph of low dimensionality by choosing a suitable *metric* , e.g. a small ϵ in the ϵ-NN approach, considerably lowers the number of required comparisons at run-time. Such an optimisation would not be possible when using a k-NN approach, as even for a low dimensionality all possible edges must be considered.

Algorithm 1 Design-time graph building

```
Input: Set of Sensors $D$
Output: Set of Edges $E$
procedure BUILDGRAPH($D$)
    $E \leftarrow \emptyset$
    while $D \not\subset \emptyset$ do
        $s_i \leftarrow D.pop()$
        for all $s_j \in D$ do
            if $metric(s_i, s_j)$ then
                $E \leftarrow E \cup \{e_{ij}\}$
            end if
        end for
    end while
    return $E$
end procedure
```

Full Toolchain Integration

Our methodology covers the conversion of an arbitrary graph building algorithm into an intermediate-circuit representation. The resulting intermediate-circuit representation, implemented on the FPGA as a hardware module, exposes multiple interfaces on the FPGA. On the input side, heterogeneous sensor data are supplied through a parallel interface as defined in the detector description. On the output side, graph features are accessible through a parallel register interface to provide edge features to successive processing modules.

Considering the application of our module in a latency-sensitive, high-throughput environment like particle experiments, direct access to graph data is required at the hardware level. Therefore, bus architectures employed in general-purpose processors, like AXI or AMBA, are not suitable for our use case. For this reason, our graph building module is connected to subsequent modules via buffered stream interfaces, reducing the routing overhead in the final design.

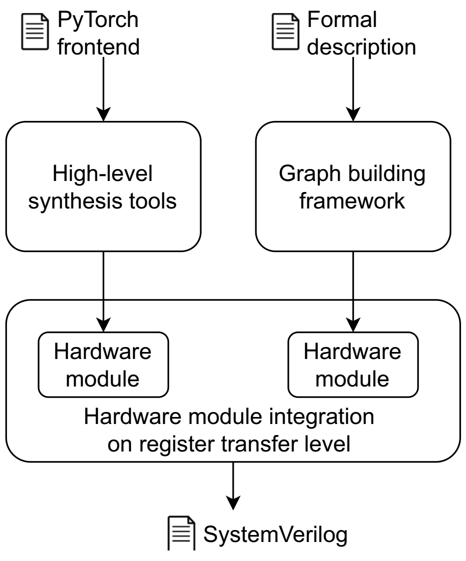


Fig. 4 Exemplary integration of our graph building methodology into a state-of-the-art HLS design flows

Figure 4 depicts exemplary, how our Chisel3-based graph building methodology is combined with state-of-the-art HLS tools, such as HLS4ML [48], FINN [45, 46] or ScaleHLS [51, 52] in order to enable the generation of hardware-accelerated neural networks. The left side of the figure depicts a generic HLS flow converting, for example, a PyTorch [50] neural network model into hardware modules.. The register transfer level description of hardware modules generated by HLS toolchains are composed of discrete registers, wires, and synthesisable operations. In a similar way, the right side of the figure depicts our proposed graph building procedure. The formal detector description and the user-defined graph building *metric* are used as an input to generate a register-transfer level description of the hardware module. As both toolchains are generating hardware descriptions in the register transfer abstraction level, merging the two modules is feasible. Last, a top level design combining both modules in SystemVerilog [53] is generated for an FPGA-specific implementation using commercially available toolchains, for example Vivado ML [54].

Module Architecture

Utilising the generated intermediate graph description, available generator templates, and user-defined hyperparameters, a hardware module is generated at the register-transfer level. The system architecture of the module is depicted in Fig. 5.

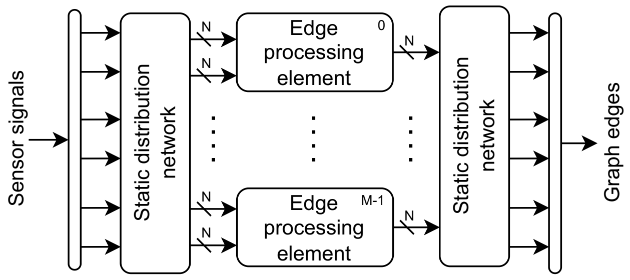


Fig. 5 System architecture of the generated hardware module. Sensor signals are received on the left side of the figure. The resulting graph edges are shown on the right side

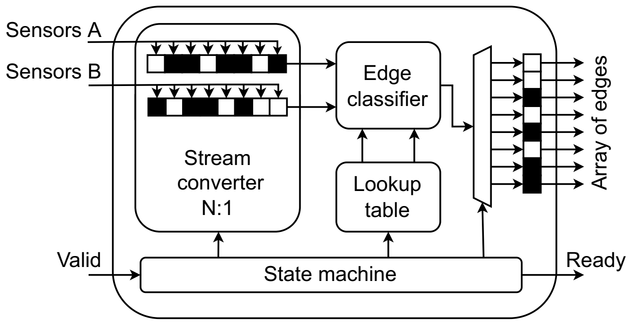


Fig. 6 The edge processing element consists of a stream converter, an edge classifier, and a lookup table. Edge registers are made available through a parallel interface

The total number of graph edges $|E|$ is factorised into M edge processing elements and N graph edges per edge processing element. Time variant readings from the detector sensors, e.g. energy or timing information, are scattered to an array of M edge processing elements via a static distribution network. In this way, each edge processing element has conflict-free access to the sensor data for classifying the respective edges. Every edge processing element builds N graph edges in a time-division multiplex. For each edge which is processed in an edge processing element, data from two adjacent sensors are required which are provided to the edge processing element. Therefore, to process N edges data from $2N$ sensors are required. Consequently, graph edges are built from candidates identified at design time yielding a sparse array of both active and inactive edges. In the described architecture, all generated edges are accessible through parallel registers. In case a serial interface is required for successive algorithms, an interface transformation is achieved by adding FIFO modules.

Figure 6 illustrates the block level diagram of an edge processing element in detail. During design-time, each hardware module is allocated N edges which are built sequentially. Static allocation allows a priori known sensor and edge features, like euclidean distances, to be stored in read-only registers. During run-time, the described module loads static features from the registers, combines them with variable input features, like the deposited energy, and classifies the edge as active or inactive. The online graph building is carried out in three steps. First, a pair of sensor readings is loaded from the shift registers, and static sensor and edge features are loaded from a static lookup table. Second, a Boolean flag is generated based on a neighbourhood condition, e.g. a user-specified metric is fulfilled for two adjacent sensors. Third, the resulting feature vector of the edge is stored in the respective register. Feature vectors of all edge processing elements are routed via a second static distribution network mapping each edge to a fixed position in the output register.

The proposed architecture takes advantages of distributed lookup tables and registers on the FPGA in two ways. First, due to the independence of the edge processing elements space-domain multiplexing is feasible on the FPGA even for large graphs. Second, static features of the graph edges and vertices are stored in distributed registers allowing logic minimisation algorithms to reduce the required memory [55].

To conclude, we developed an architecture for online graph building which is well suited for the latency-constrained environment of low level trigger systems in particle physics experiments. The variable output interface allows for an easy integration of successive trigger algorithms and leaves ample room for application-specific optimisation. The number of output queues is controlled by the parameter N which yields a flexible and efficient design supporting variable degrees of time-domain multiplexing.

Case Study: Belle II Trigger

To demonstrate the working principle of our concept, we adapt our graph building methodology for the first-level (L1) trigger of the Belle II experiment. The implementation focuses on the CDC (see sect. "Simulation and Dataset") that is responsible for all track-based triggers.

Environment

The aim of the trigger system is to preselect collision events based on their reconstructed event topologies. In order to filter events, a multi-stage trigger system is employed. As a result, the effective data rate and thus the processing load of the data acquisition systems is reduced.

To give an overview of the constraints and requirements imposed by the experiment, the existing system is briefly described in the following. The L1 track triggers are shown schematically in Fig. 7. They perform real-time filtering with a strict latency requirement of $5\,\mu$s [20]. The sense wires

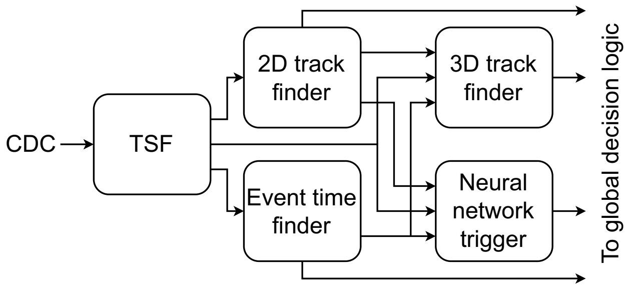


Fig. 7 Flowchart of the L1 trigger system at the Belle II experiment, limited to systems that use the wire hit information from the CDC [56]

inside the CDC are sampled with 32MHz and wire hits are accumulated for approximately 500ns. In order to process all available input signals concurrently, a distributed FPGA-based platform is employed.

To obtain a trigger decision, track segments are generated from incoming events in parallel by performing space-division multiplexing. Based on the output of the track segment finder (TSF), multiple algorithms including conventional 2D and 3D track finding algorithms as well as a Neural Network Trigger [14] generate track objects of varying precision, efficiency, and purity for a Global Decision Logic [56].

The integration of GNNs in the L1 trigger system requires an online-graph building approach that is optimised for both latency and throughput. In this case study, we employ our proposed toolchain to generate an application-specific graph-building module as described in the previous section while adhering to constraints in the challenging environment of the Belle II experiment.

Graph Building

The wire configuration of the CDC is mapped onto the formal detector definition from sect. "Graph Building", using wires as discrete sensors. These sensors are called nodes or vertices in the following. Inside the L1 trigger system, three signals are received per wire: a *hit identifier*, the *TDC readout* and the *ADC readout*, where TDC is the output of a time-to-digital converter measuring the drift time, and ADC is the output of an analogue-to-digital converter measuring the signal height that is proportional to the energy deposition in a drift cell. Cartesian coordinates of the wires inside the detector are known during design time and used as static sensor features. Additionally, the distance between two vertices, which is also known during design-time, is considered as an edge feature.

Illustrating the working principle our graph building approaches, Fig. 8 depicts four cut-outs of the CDC in the x-y plane for $z = 0$.

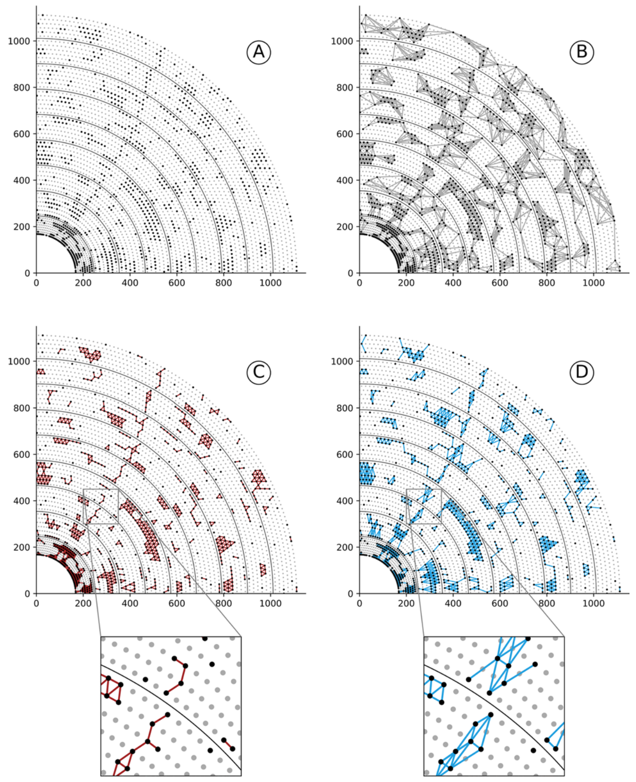


Fig. 8 Typical event display of the CDC for various graph building approaches. Quadrants show Ⓐ all hits, Ⓑ k-NN graph building (k=6), Ⓒ ϵ-NN graph building (ϵ=22 mm), and Ⓓ p-NN graph building (see Fig. 9). The inserts show zooms to a smaller section of the CDC

In sector Ⓐ, *hit identifier* received by the detector for an exemplary event are indicated by black markers. The other three sectors show one graph building approach each: Sector Ⓑ depicts a k-NN graph for of $k = 6$, as there are up to six direct neighbours for each wire. The k-NN graphs connects wires that are widely separated. Sector Ⓒ shows an ϵ-NN graph for $\epsilon = 22$ mm. The specific value for ϵ is chosen, because 22 mm is in the range of one to two neighbour wires inside the CDC. This graph building approach connects hits in close proximity only, yielding multiple separated graphs. In addition, more edges are detected in the inner rings compared to the outer rings of the detector due to the higher wire density in this region. Finally, sector Ⓓ shows a p-NN graph using the pattern described in Fig. 9. The pattern extends the existing pattern [57–59] of the currently implemented TSF in the L1 trigger system by taking neighbours in the same superlayers into account. When comparing the ϵ-NN graphs and the p-NN graphs with each other, it is observed that the degrees[2] of p-NN vertices are more evenly distributed (see inserts in Fig. 8).

[2] The degree of a vertex of a graph is the number of edges that are connected to the vertex.

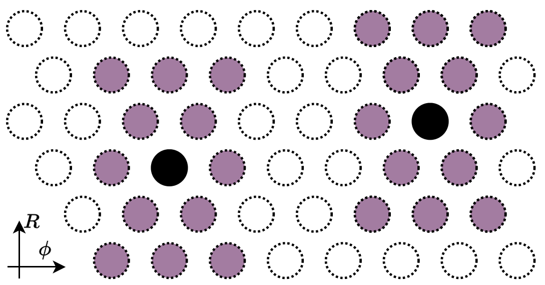


Fig. 9 Two query vertices illustrate the neighbourhood pattern in hourglass shape used for the Belle II detector case study. The superlayer is rolled off radially and an exemplary cut-out is shown. Vertices which are considered neighbour candidates of the respective query vertex are shown as purple-filled markers

Parameter Exploration

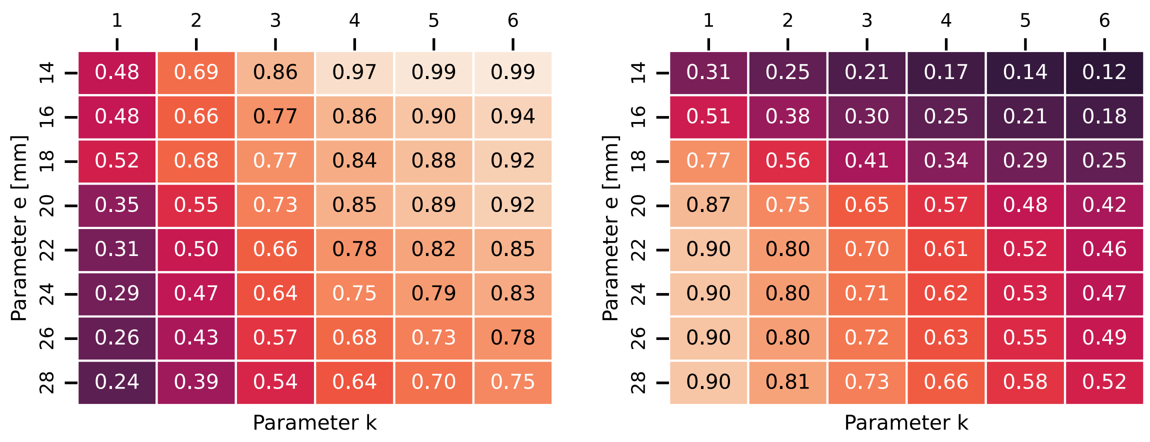
In general, k-NN, ϵ-NN and p-NN algorithms generate different graphs for an identical input event. However, to replace k-NN graph building with a locally constrained graph building approach, the graphs should ideally be identical. As the generated graphs depend strongly on the chosen hyperparameters, on the geometry of the detector, and on the background distribution of the events under observation, a quantitative measure of the similarity of the generated graphs between k-NN graphs and locally constrained graphs, such as ϵ-NN or p-NN graphs, is necessary. The optimal choice of the hyperparameter ϵ^* is the one that maximises the similarity for any k. For this optimisation, we use simulated events as described in sect. "Simulation and Dataset". We generate both the k-NN graphs and the locally constrained graphs on the dataset considering the neighbourhood of wires inside the detector. Edges of the k-NN graphs are labelled E_k, whereas the edges of observed locally constrained graphs are labelled E_l. We measure the similarity between the two graphs using the binary classifications metrics recall and precision defined as

$$\text{recall} = \frac{|E_k \cap E_l|}{|E_k|}, \tag{3}$$

$$\text{precision} = \frac{|E_k \cap E_l|}{|E_l|}. \tag{4}$$

To perform the evaluation, we automate the parameter exploration using Python 3.10. We vary k between 1 and 6 and ϵ between 14 and 28 mm, as the minimal distance between two wires in the CDC is approximately 10 mm. Precision and recall scores are calculated for every pair of k and ϵ parameters and show mean value over 2000 events in Fig. 10. As expected, the precision score increases monotonically when parameter k is increased. In addition, it increases if the parameter ϵ is reduced. The recall score behaves in the opposite way: It monotonically decreases when parameter k is increased. In addition, it decreases if the parameter ϵ is decreased. Similarity is defined as the ratio between recall and precision, where an optimal working point also maximises recall and precision themselves. We observe that we do not find high similarity for all values of k. Maximal similarity is found for $k = 3$ and $\epsilon = 22$ mm, and $k = 4$ and $\epsilon = 28$ mm, respectively. The corresponding precision and recall on the underlying data set are around 65–70%.

The similarity between k-NN and ϵ-NN graphs can be interpreted in relation to the mathematical statement from Ref. [43] (compare sect. "Graph Building"). Based on the background noise and the large number of hits per event, we assume that the *hit identifiers* in the dataset are approximately uniformly distributed. Therefore, we expect that pairs of k-NN and ϵ-NN graphs exist that exhibit a high degree of similarity, e.g. precision and recall scores close to one.



(a) Precision (b) Recall

Fig. 10 Precision and recall for the comparison of the k-NN and ϵ-NN graph building approaches

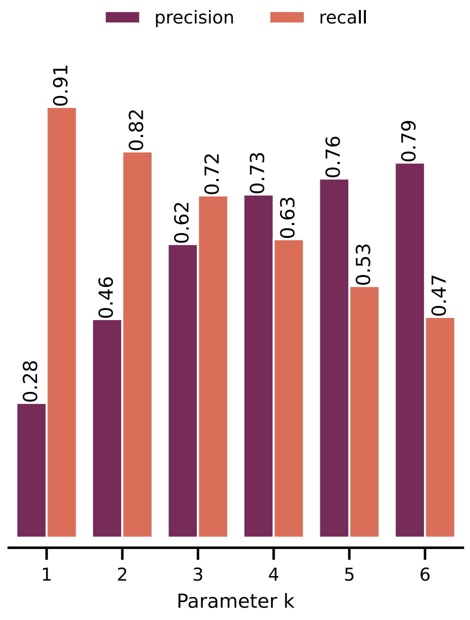


Fig. 11 Precision and recall for the comparison between the p-NN graphs (for the pattern see in Fig. 9) and the k-NN graphs

Our expectation is only partially met as the trade-off point reaches only about 65–70 %. The achieved metrics indicate, that the k-NN graph-building approach from high-level trigger algorithms may be replaced by the ϵ-NN graph-building approach in the first-level trigger and behave qualitatively similar.

We perform the same comparison between the k-NN and the p-NN graph building approach as shown in Fig. 11. We achieve similar results in comparison to the ϵ-NN comparison: the recall score is monotonically decreasing for a larger parameter k, and the precision score is monotonically increasing for larger parameter k. For k between three and four, precision and recall scores are approximately similar and around 70 %.

Again, our expectation of a high degree of similarity is only partially met. This similarity is to be expected, as the chosen pattern is also locally constrained and approximately ellipsoid.

Prototype Setup

For the implementation of the proposed algorithm into a hardware prototype, the CDC is partitioned into 20 partially overlapped, independent sectors in ϕ and radial distance r for the L1 trigger. Each Φ-r-sector is processed physically isolated by one FPGA platform, the overlapping of the sectors ensures that no data is lost. The overlapping sectors must be merged in subsequent reconstruction steps that are not part of the graph-building stage. In the following, the graph-building module is implemented on the Belle II Universal Trigger Board 4 (UT4) featuring a Xilinx Ultrascale XCVU160WE-2E. The UT4 board is currently used in the Belle II L1 Trigger and therefore serves as a reference for for future upgrades of the L1 trigger system.

To implement the online graph building module, we generate JSON databases for every Φ-sector of the CDC. Each database represents a formal detector containing the positions of the wires and information about sensor features as described in sect. "Graph Building". Sensor features are composed of 1bit for the binary *hit identifier*, 5bit for the *TDC readout*, 4bit for the *ADC readout*, and the Cartesian coordinates of the wires. Additional edge features containing information about the wire distances of two adjacent vertices are included as well. The resolution of the euclidean features can be arbitrarily chosen and is therefore considered a hyperparameter of the module implementation.

The sector database and a function describing the pattern as illustrated in Fig. 9 is provided as an input to our proposed toolchain which is implemented in Python 3.10. An intermediate graph representation is generated as a JSON database, containing a type definitions of all vertices, edges and their respective features. In addition, features known at design-time, such as Cartesian coordinates, are rounded down, quantised equally spaced, and included in the intermediate graph representation. By generating the databases for all 20 sectors, we identify the smallest and largest sector of the CDC to provide a lower and an upper bound for our problem size. The maximum number of edges in each sector is determined by the pattern from Fig. 9. The smallest sectors are located in superlayer two containing 498 vertices and 2305 edges, while the largest sectors are located in superlayer six containing 978 vertices and 4545 edges.

To demonstrate our graph building approach, we synthesise the previously generated intermediate graph representation into a hardware module targeting the architecture of the UT4. We provide the JSON database as an input for the hardware generator, which is a set of custom modules implemented in Chisel 3.6.0. In addition, we provide a Scala function that performs the online classification of edge candidates based on the *hit identifier*: an edge candidate is considered valid, if the *hit identifiers* of both adjacent vertices are hit. For the edge processing elements we choose the number of edges per edge processing element N of eight. Therefore, eight edges are processed sequentially in every edge processing element as described in sect. "Toolchain". Based on the required throughput of 32MHz, a system frequency of at least 256MHz is required to achieve the desired throughput. By starting the generator application, edges and features are extracted from the intermediate graph representation and scheduled on edge processing elements. After completion, the hardware generator produces a SystemVerilog file containing the graph-building hardware module [53].

Table 2 Overview of the features of the sensors used to define the edges

Feature	Type	Occurrence	Width
Hit identifier	Dynamic	2	1 bit
ADC readout	Dynamic	2	4 bit
TDC readout	Dynamic	2	5 bit
X coordinate	Static	2	4 to 16 bit
Y coordinate	Static	2	4 to 16 bit
Distance	Static	1	4 to 16 bit

The occurrence indicates how often the respective feature is represented in an edge

Implementation Results

For further evaluation, the SystemVerilog module implementing the presented p-NN graph building is synthesised out-of-context for the UT4 board using Xilinx Vivado 2022.2. During synthesis, the target frequency f_{sys} is set to 256MHz, for which no timing violations are reported by the tool. In addition, functional tests are performed to validate the algorithmic correctness of the module. In the following, we perform two series of measurements to validate the feasibility of the proposed implementation on the Xilinx Ultrascale XCVU160WE-2E FPGA.

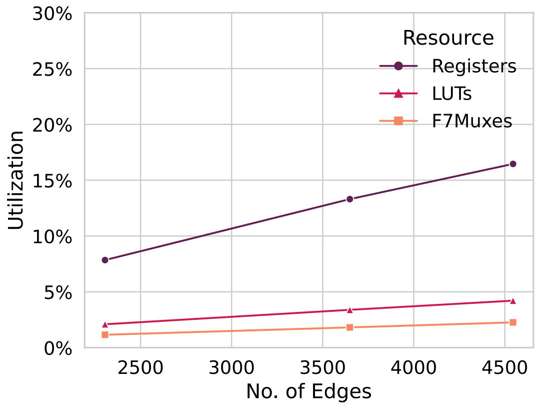
Figure 12 depicts the results of the two evaluation series, reporting the utilisation on the UT4 board for the respective resource types. The first series of three synthesised versions is shown in Fig. 12a, varying the input graph size in a suitable range between the 2305 and 4545 edges. The highest occupancy is reported for registers, amounting up to 16.46 % for the largest input graph, as opposed to 7.84 % for the smallest graph. For all other resource types, the utilisation is lower than 5 %. In general, it is observed that the resource utilisation scales linearly with the number of edges in the input graph.

For the second series, a variation in resolution of the underlying edge features is considered. An overview of all utilised features is given in Table 2. The width of features that are received as inputs from the CDC, namely *hit identifier*, *ADC readout*, and *TDC readout*, are exemplary chosen in a way which is supported by the current readout system. As an example, the *TDC readout* quantisation of 5bit derives from the drift time resolution of 1ns at a trigger data input rate of 32MHz. The resolution of euclidean coordinates and distances can be optimised at design-time.

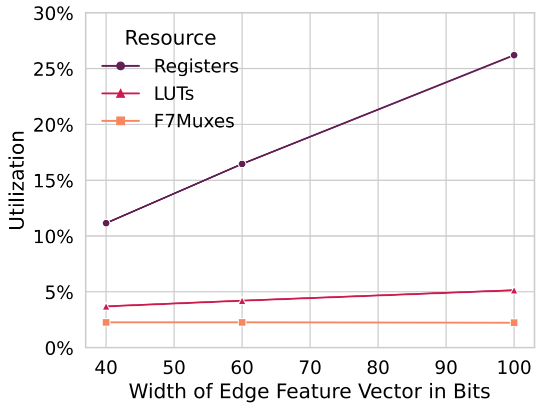
In the following, we choose a resolution between 4 to 16 bit which results in a quantisation error for the euclidean coordinates in the range 34.4 to 0.017 mm. 4 bit per coordinate result in a total edge width of 40bit, whereas a resolution of 16bit per coordinate results in a total edge width of 100bit.

The implementation utilisation of all three synthesised modules is shown in Fig. 12b, varying the resolution of euclidean coordinates and distances in the generated edges.

Similar to the previous measurement, the highest utilisation is reported for registers, taking up between 11.1% and 26.1% depending on the width of the edges. It can be seen, that the implementation size scales linearly with the width of the graph edges. Increasing the resolution of a parameter, e.g. the *TDC readout*, therefore leads to a proportionally higher utilisation of the corresponding resource on the



(a) Utilization for a variable graph size $|E|$. The queue length parameter is set to eight, each edge is composed of 60 bits.



(b) Utilisation for a variable edge width. The queue length parameter is set to eight, the input graph is composed of 4545 edges.

Fig. 12 Resource utilisation reported after out-of-context synthesis on the UT4 platform using Vivado 2022.2 for registers, lookup tables (LUTs) and multiplexers (F7MUXes). Measurement are indicated by dots and connected by lines through linear interpolation to guide the eye. Unreported resource types are not utilised in the implementation

Table 3 Utilisation for variable graph size |*E*|, |*V*| and edge width

No. of vertices	No. of edges	Width of edge	Registers		LUTs		F7Muxes	
			Abs.	%	Abs.	%	Abs.	%
498	2305	60 bit	145333	7.84	19370	2.09	5760	1.15
786	3649	60 bit	246511	13.30	31360	3.39	9120	1.81
978	4545	40 bit	206573	11.15	34252	3.70	11360	2.26
978	4545	60 bit	304919	16.46	38968	4.21	11360	2.26
978	4545	100 bit	485473	26.20	47642	5.14	11200	2.23

Numerical implementation results are identical to the values shown in Fig. 12

FPGA. For a better overview, the overall evaluation results are also presented in Table 3.

Based on the presented results, the implementation of the graph building module is considered feasible on the UT4 board. By experimental evaluation we show that our hardware architecture can be implemented semi-automatically for the L1 trigger of the Belle II experiment, enabling the deployment of GNNs in the latency-constrained trigger chain. The feature vectors of the edges are provided via a parallel output register, where the address of every edge is statically determined at design time. Depending on successive filtering algorithms, any number of output queues can be provided. To conclude, our toolchain allows for a flexible and resource efficient design of online graph building modules for trigger applications. In the presented implementation, our module is able to achieve a throughput of 32 million samples per second at total latency of 39.06ns, corresponding to ten clock cycles at f_{sys}. As the reported latency is well below the required $\mathcal{O}(1\mu s)$, our graph building module leaves a large part of the latency and resource budget on FPGAs to the demanding GNN solutions.

Conclusion

In our work, we analysed three graph building approaches on their feasibility for the real-time environment of particle physics machine-learning applications. As the k-NN algorithm, which is favoured by state-of-the-art GNN-tracking solutions, is unsuitable for the strict sub-microsecond latency constraints imposed by trigger systems, we identify two locally constrained nearest neighbour algorithms ϵ-NN and p-NN as possible alternatives. In an effort to reduce the number of design-iterations and time-consuming hardware debugging, we develop a generator-based hardware design methodology tailored specifically to online graph-building algorithms. Our approach generalises graph-building algorithms into an intermediate-graph representation based on a formal detector description and user-specified metrics. The semi-automated workflow enables the generation of FPGA-accelerated hardware implementation of locally constrained nearest neighbour algorithms. To demonstrate the capabilities of our toolchain, we perform a case study on the trigger system of the Belle II detector. We implement an online graph-building algorithm which adapts the pattern of the current track segment finder, demonstrating the feasibility of our approach in the environment of particle physics trigger applications. The code used for this research is available open source under Ref. [60].

Nearest neighbour algorithms presented in this work achieve a $\mathcal{O}(1)$ time complexity and a $\mathcal{O}(|E|)$ space complexity, compared to a $\mathcal{O}(|D|)$ time complexity in approximate k-NN algorithms or a $\mathcal{O}(k|D|\log(|D|)$ complexity in the sequential case [23, 25]. As a result, our semi-automated methodology may also be applied to other detectors with heterogeneous sensor arrays to build graphs under latency constraints, enabling the integration of GNN-tracking solutions in particle physics.

During the evaluation of our similarity metric, we found a non-negligible difference between k-NN graphs and locally constrained NN-graphs. For the complete replacement of k-NN graphs with our proposed ϵ-NN and p-NN graphs, the differences must be taken into account to achieve optimal performance when designing successive trigger stages. For this reason, we consider the future development of methods for algorithm co-design essential for integrating GNNs into real-world trigger applications. Careful studies of possible difference between simulated data are another main direction of future work.

Acknowledgements The authors would like to thank the Belle II collaboration for useful discussions and suggestions on how to improve this work. It is a great pleasure to thank (in alphabetical order) Greta Heine, Jan Kieseler, Christian Kiesling and Elia Schmidt for discussions, and Tanja Harbaum, Greta Heine, Taichiro Koga, Florian Schade, and Jing-Ge Shiu for feedback and comments on earlier versions of the manuscript.

Author contributions M. N., J. B., P. D., T. F, L. R., S. S. and K. U. contributed to the design and implementation of the research. M. N. contributed to the analysis of the results. M. N. and T. F. contributed to the writing of the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The datasets generated during and analysed during the current study are property of the Belle II collaboration and not publicly available.

Code availability The code used for this research is available open source under Ref. [59].

Declarations

Competing interests The authors declare that they have no competing interests.

References

1. Ju X, Nachman B (2020) Supervised jet clustering with graph neural networks for Lorentz boosted bosons. Phys Rev D 102(7):075014. https://doi.org/10.1103/PhysRevD.102.075014
2. Wemmer F et al (2023) Photon reconstruction in the Belle II Calorimeter using graph neural networks. arXiv:2306.04179 [hep-ex]
3. DeZoort G, Thais S, Duarte J, Razavimaleki V, Atkinson M, Ojalvo I, Neubauer M, Elmer P (2021) Charged particle tracking via edge-classifying interaction networks. Comput Softw Big Sci 5(1):26. https://doi.org/10.1007/s41781-021-00073-z
4. Duarte J, Vlimant JR (2022) Graph neural networks for particle tracking and reconstruction. in: artificial intelligence for high energy physics, Chap. 12, pp 387–436. https://doi.org/10.1142/9789811234033_0012
5. Ju X, Farrell S, Calafiura P, Murnane D, Prabhat Gray L, Klijnsma T, Pedro K, Cerati G, Kowalkowski J (2020) Graph neural networks for particle reconstruction in high energy physics detectors. In: 33rd Annual Conference on Neural Information Processing Systems. https://doi.org/10.48550/arXiv.2003.11603
6. Mikuni V, Canelli F (2020) ABCNet: an attention-based method for particle tagging. Eur Phys J Plus 135(6):463. https://doi.org/10.1140/epjp/s13360-020-00497-3
7. Qu H, Gouskos L (2020) ParticleNet: jet tagging via particle clouds. Phys Rev D 101(5):056019. https://doi.org/10.1103/PhysRevD.101.056019
8. Pata J, Duarte J, Vlimant J-R, Pierini M, Spiropulu M (2021) MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks. Eur Phys J C 81(5):381. https://doi.org/10.1140/epjc/s10052-021-09158-w
9. Elabd A, Razavimaleki V, Huang S-Y, Duarte J, Atkinson M, DeZoort G, Elmer P, Hauck S, Hu J-X, Hsu S-C (2022) Graph neural networks for charged particle tracking on FPGAs. Front Big Data 5:828666. https://doi.org/10.3389/fdata.2022.828666
10. Qasim SR, Kieseler J, Iiyama Y, Pierini M (2019) Learning representations of irregular particle-detector geometry with distance-weighted graph networks. Eur Phys J C 79(7):608. https://doi.org/10.1140/epjc/s10052-019-7113-9
11. Iiyama Y, Cerminara G, Gupta A, Kieseler J, Loncar V, Pierini M, Qasim SR, Rieger M, Summers S, Onsem GV (2020) Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics. Front Big Data 3:598927. https://doi.org/10.3389/fdata.2020.598927
12. Aad G (2020) Operation of the ATLAS trigger system in Run 2. JINST 15(10):10004. https://doi.org/10.1088/1748-0221/15/10/P10004
13. Sirunyan AM (2020) Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV. JINST 15(10):10017. https://doi.org/10.1088/1748-0221/15/10/P10017
14. Unger KL, Bähr S, Becker J, Knoll AC, Kiesling C, Meggendorfer F, Skambraks S (2023) Operation of the neural z-vertex track trigger for belle ii in 2021—a hardware perspective. J Phys Conf Ser 2438(1):012056. https://doi.org/10.1088/1742-6596/2438/1/012056
15. Liang S, Wang Y, Liu C, He L, Li H, Xu D, Li X (2021) Engn: a high-throughput and energy-efficient accelerator for large graph neural networks. IEEE Trans Computers 70(9):1511–1525. https://doi.org/10.1109/TC.2020.3014632
16. Zhang B, Kuppannagari SR, Kannan R, Prasanna V (2021) Efficient neighbor-sampling-based GNN training on CPU-FPGA heterogeneous platform. In: 2021 IEEE High Performance Extreme Computing Conference (HPEC), pp 1–7. https://doi.org/10.1109/HPEC49654.2021.9622822
17. Karle CM, Kreutzer M, Pfau J, BeckerJ (2022) A hardware/software co-design approach to prototype 6G mobile applications inside the GNU radio SDR ecosystem using FPGA hardware accelerators. In: International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies. ACM, New York. pp 33–44. https://doi.org/10.1145/3535044.3535049
18. Thais S, Calafiura P, Chachamis G, DeZoort G, Duarte J, Ganguly S, Kagan M, Murnane D, Neubauer MS, Terao K (2022) Graph neural networks in particle physics: implementations, innovations, and challenges. arXiv. https://doi.org/10.48550/arXiv.2203.12852
19. Shlomi J, Battaglia P, Vlimant JR (2021) Graph neural networks in particle physics. Mach Learn Sci Technol 2(2):021001. https://doi.org/10.1088/2632-2153/abbf9a
20. Abe T et al (2010) Belle II technical design report. arXiv. https://doi.org/10.48550/arXiv.1011.0352
21. Rossi M, Vallecorsa S (2022) Deep learning strategies for ProtoDUNE raw data denoising. Comput Softw Big Sci 6(1):2. https://doi.org/10.1007/s41781-021-00077-9
22. Hartmann F (2020) The phase-2 upgrade of the CMS level-1 trigger. Technical report, CERN, Geneva. https://cds.cern.ch/record/2714892
23. Vaidya PM (1989) AnO(n logn) algorithm for the all-nearest-neighbors Problem. Discrete Comput Geom 4(2):101–115. https://doi.org/10.1007/BF02187718
24. Callahan PB, Kosaraju SR (1995) A decomposition of multidimensional point sets with applications to K-nearest-neighbors and n-body potential fields. J ACM 42(1):67–90. https://doi.org/10.1145/200836.200853
25. Connor M, Kumar P (2008) Parallel construction of k-nearest neighbor graphs for point clouds. In: IEEE/ EG Symposium on Volume and Point-Based Graphics. https://doi.org/10.2312/VG/VG-PBG08/025-031
26. Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of the 25th International Conference on Very Large Data Bases. VLDB '99, San Francisco, CA, USA. pp 518–529
27. Hajebi K, Abbasi-Yadkori Y, Shahbazi H, Zhang H (2011) Fast approximate nearest-neighbor search with k-nearest neighbor graph. In: Proceedings of the Twenty-Second International Joint

Conference on Artificial Intelligence, IJCAI-11, pp 1312–1317 . https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-222
28. Harwood B, Drummond T (2016) FANNG: fast approximate nearest neighbour graphs. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 5713–5722 https://doi.org/10.1109/CVPR.2016.616
29. Malkov YA, Yashunin DA (2020) Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans Pattern Anal Mach Intell 42(4):824–836. https://doi.org/10.1109/TPAMI.2018.2889473
30. Besta M, Fischer M, Kalavri V, Kapralov M, Hoefler T (2021) Practice of streaming processing of dynamic graphs: concepts, models, and systems. IEEE Trans Parallel Distrib Syst. https://doi.org/10.1109/TPDS.2021.3131677
31. Gui C-Y, Zheng L, He B, Liu C, Chen X-Y, Liao X-F, Jin H (2019) A survey on graph processing accelerators: challenges and opportunities. J Computer Sci Technol 34(2):339–371. https://doi.org/10.1007/s11390-019-1914-z
32. Liu C, Liu H, Zheng L, Huang Y, Ye X, Liao X, Jin H (2023) FNNG : a high-performance FPGA-based accelerator for k-nearest neighbor graph construction. In: Ienne P, Zhang Z (eds) Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ACM, New York. pp 67–77. https://doi.org/10.1145/3543622.3573189
33. Nurvitadhi E, Weisz G, Wang Y, Hurkat S, Nguyen, M, Hoe JC, Martinez JF, Guestrin C (2014) GraphGen: an FPGA framework for vertex-centric graph computation. In: IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines. IEEE, Boston. pp 25–28. https://doi.org/10.1109/FCCM.2014.15
34. Zeng H, Prasanna, V Graphact (2020) In: Neuendorffer S, Shannon L (eds) Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, New York, pp 255–265.https://doi.org/10.1145/3373087.3375312
35. Yan M, Deng L, Hu X, Liang L, Feng Y, Ye X, Zhang Z, Fan D, Xie Y (2020) HYGCN: a GCN accelerator with hybrid architecture. In: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, San Diego, pp 15–29. https://doi.org/10.1109/HPCA47549.2020.00012
36. Geng T, Li A, Shi R, Wu C, Wang T, Li Y, Haghi P, Tumeo A, Che S, Reinhardt S, Herbordt MC (2020) AWB-GCN: a graph convolutional network accelerator with runtime workload rebalancing. In: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, Athens, pp 922–936. https://doi.org/10.1109/MICRO50266.2020.00079
37. Agostinelli S (2003) GEANT4—a simulation toolkit. Nucl Instrum Methods Phys Res A 506:250–303. https://doi.org/10.1016/S0168-9002(03)01368-8
38. Kuhr T, Pulvermacher C, Ritter M, Hauth T, Braun N (2019) The Belle II core software. Comput Softw Big Sci 3(1):1. https://doi.org/10.1007/s41781-018-0017-9
39. Kou E (2019) The Belle II Physics Book. PTEP 2019(12):123–01 arXiv:1808.10567 [hep-ex]. https://doi.org/10.1093/ptep/ptz106
40. Abe T et al (2010) Belle II technical design report. Technical report, Belle-II. arXiv:1011.0352
41. Liptak ZJ (2022) Measurements of beam backgrounds in SuperKEKB Phase 2. Nucl Instrum Methods A 1040:167168 arXiv:2112.14537 [physics.ins-det]. https://doi.org/10.1016/j.nima.2022.167168
42. Natochii A (2022) Beam background expectations for Belle II at SuperKEKB. In: Snowmass 2021
43. Prokhorenkova L, Shekhovtsov A (2020) Graph-based nearest neighbor search: from practice to theory. In: Proceedings of the 37th International Conference on Machine Learning, vol. 119, pp 7803–7813 https://doi.org/10.48550/arXiv.1907.00845
44. Pfau J, Figuli SPD, Bähr S, Becker J (2018) Reconfigurable FPGA-based channelization using polyphase filter banks for quantum computing systems. In: Applied Reconfigurable Computing. Architectures, Tools, and Applications. Lecture Notes in Computer Science, vol. 10824, pp 615–626 . https://doi.org/10.1007/978-3-319-78890-6_49
45. Umuroglu Y, Fraser NJ, Gambardella G, Blott M, Leong P, Jahre M, Vissers K (2017) FINN: a framework for fast, scalable binarized neural network inference. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp 65–74. https://doi.org/10.1145/3020078.3021744
46. Blott M, Preußer TB, Fraser NJ, Gambardella G, O'Brien K, Umuroglu Y, Leeser M, Vissers K (2018) FINN-R: an end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Trans Reconfig Technol Syst 11(3):1–23. https://doi.org/10.1145/3242897
47. Duarte J, Han S, Harris P, Jindariani S, Kreinar E, Kreis B, Ngadiuba J, Pierini M, Rivera R, Tran N (2018) Fast inference of deep neural networks in FPGAs for particle physics. JINST 13(07):07027. https://doi.org/10.1088/1748-0221/13/07/P07027
48. FastML Team (2023) hls4ml. Zenodo. https://doi.org/10.5281/zenodo.1201549
49. Bachrach J, Vo H, Richards B, Lee Y, Waterman A, Avižienis R, Wawrzynek J, Asanović K (2012) Chisel: constructing Hardware in a Scala Embedded Language. In: Proceedings of the 49th Annual Design Automation Conference, pp. 1216–1225. https://doi.org/10.1145/2228360.2228584
50. Paszke A (2019) Pytorch: an imperative style, high-performance deep learning library. In: NeurIPS 2019
51. Li M, Liu Y, Liu X, Sun Q, You X, Yang H, Luan Z, Gan L, Yang G, Qian D (2021) The deep learning compiler: a comprehensive survey. IEEE Trans Parallel Distrib Syst 32(3):708–727. https://doi.org/10.1109/TPDS.2020.3030548
52. Ye H, Jun H, Jeong H, Neuendorffer S, Chen D (2022) ScaleHLS: a scalable high-level synthesis framework with multi-level transformations and optimizations. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, New York. pp. 1355–1358. https://doi.org/10.1145/3489517.3530631
53. 1800-2017—IEEE standard for SystemVerilog–unified hardware design, specification, and verification language (2018) Technical report. https://doi.org/10.1109/IEEESTD.2018.8299595
54. AMD Vivado ML. https://www.xilinx.com/products/design-tools/vivado.html. Accessed 10 July 2023.
55. Harbaum T, Seboui M, Balzer M, Becker J, Weber M (2016)A content adapted FPGA memory architecture with pattern recognition capability for L1 track triggering in the LHC environment. In: 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp 184–191. https://doi.org/10.1109/FCCM.2016.52
56. Lai Y-T, Bühr S, Chang M-C, Iwasaki Y, Kim J-B, Kim K-T, Kiesling C, Koga T, Lu P-C, Liu S-M (2018) Level-1 track trigger with central drift chamber detector in belle ii experiment. In: 2018 IEEE Nuclear Science Symposium and Medical Imaging Conference, pp. 1–4. https://doi.org/10.1109/NSSMIC.2018.8824506
57. Pohl S (2018) Track reconstruction at the first level trigger of the Belle II experiment. PhD thesis, Ludwig-Maximilians-Universität München. https://doi.org/10.5282/edoc.22085
58. Unger KL, Bähr S, Becker J, Iwasaki Y, Kim K, Lai Y-T (2020) Realization of a state machine based detection for track segments in the trigger system of the belle ii experiment. In: Proceedings of Topical Workshop on Electronics for Particle Physics—PoS(TWEPP2019), vol. 370, p. 145. https://doi.org/10.22323/1.370.0145

59. Unger KL, Neu M, Becker J, Schmidt E, Kiesling C, Meggendorfer F, Skambraks S (2023) Data-driven design of the Belle II track segment finder. JINST 18(02):02001. https://doi.org/10.1088/1748-0221/18/02/C02001
60. Neu M et al Online graph building on FPGAs for machine learning trigger applications in particle physics. https://github.com/realtime-tracking/graphbuilding. Accessed 15 June 2023