Optimizing Graph Neural Networks for Jet Tagging in Particle Physics on FPGAs

Zhiqiang Que*, Marcus Loo*, Hongxiang Fan*, Maurizio Pierini†, Alexander Tapper‡, Wayne Luk*
*Department of Computing, Imperial College London, UK, {z.que,marcus.loo20, h.fan17, w.luk}@imperial.ac.uk
† European Organization for Nuclear Research (CERN), Geneva, Switzerland, maurizio.pierini@cern.ch
‡ Department of Physics, Imperial College London, UK, a.tapper@imperial.ac.uk

***Abstract*—This work proposes a novel reconfigurable architecture for reducing the latency of JEDI-net, a Graph Neural Network (GNN) based algorithm for jet tagging in particle physics, which achieves state-of-the-art accuracy. Accelerating JEDI-net is challenging since it requires low latency to deploy the network for event selection at the CERN Large Hadron Collider. This paper proposes an outer-product based matrix multiplication approach customized for GNN-based JEDI-net, which increases data spatial locality and reduces design latency. It is further enhanced by code transformation with strength reduction which exploits sparsity patterns and binary adjacency matrices to increase hardware efficiency while reducing latency. In addition, a customizable template for this architecture has been designed and open-sourced, which enables the generation of low-latency FPGA designs with efficient resource utilization using high-level synthesis tools. Evaluation results show that our FPGA implementation is up to 9.5 times faster and consumes up to 6.5 times less power than a GPU implementation. Moreover, the throughput of our FPGA design is sufficiently high to enable deployment of JEDI-net in a sub-microsecond, real-time collider trigger system, enabling it to benefit from improved accuracy.**

I. Introduction

Real-time data processing from high-energy proton collisions at the CERN Large Hadron Collider (LHC) is challenging since the particle detectors around the LHC ring produce hundreds of terabytes of data per second [1] from collisions that occur every 25 ns. The large data produced from the detectors are reduced by a real-time processing system, known as the trigger, which keeps interesting collision events while discarding the others. In the trigger system, jet tagging is an important but challenging task. It identifies the decays of high-momentum heavy particles produced at the LHC and distinguishes them from ordinary jets which come from the hadronization of quarks and gluons. High accuracy in the trigger is crucial to keep only the most interesting events while keeping the output bandwidth low. [1, 2] utilize Multi-Layer Perceptrons (MLP) networks for jet tagging on FPGAs, which achieve an accuracy of around 75%. [3] presents JEDI-net, a Graph Neural Network (GNN) based algorithm, which achieves the state-of-the-art accuracy of over 80% for jet tagging and is in high demand in the trigger system.

However, the GNN-based JEDI-net is more complex than MLPs. It involves three MLP networks and three matrix-matrix multiplication units with large adjacency matrices. It requires large amounts of computation and suffers from large inference latency, which makes it impossible to be deployed in real-time in the level-1 trigger (L1T) system of an LHC experiment [3]. The L1T, using only FPGAs, requires processing latencies of applications in a fixed time, within ~1μs, in the updated High-Luminosity Large Hadron Collider (HL-LHC) [4]. If algorithm latency exceeds the limit, data or interesting events are lost. Hence, accelerating JEDI-net inference using reconfigurable accelerators such as FPGAs is essential in the LHC since it would enable sophisticated processing to run in real time on the data stream from detectors. This work proposes several optimizations to accelerate the GNN-based JEDI-net using high-level synthesis (HLS) on FPGAs. We also open-source the HLS templates of the graph neural network of JEDI-net for the community.

To efficiently accelerate the GNN-based JEDI-net, this work proposes an outer-product based matrix-matrix multiplication (MMM) approach for the GNN aggregation function computation, which inputs and outputs the data in a column-major order to increase the data spatial locality and reduce the design latency. The design pipeline also adopts column-major order instead of the conventional row-major order for the data layout of the intermediate results traversed along the hardware datapath. Working together with the column-major order representation, the outer-product based MMM enables a coarse-grained pipeline with a low Initiation Interval (II) to improve the design throughput. Additionally, this approach is further enhanced by a custom strength reduction for the matrix multiplication operations based on the characters of the adjacency matrices of JEDI-net. It not only avoids expensive multiplication and involves just a few additions, but also removes the input of the adjacency matrix to save memory bandwidth. The outer-product MMM enhanced by the strength reduction largely reduces the computation cost and memory access as well as the power consumption of JEDI-net to improve the hardware design throughput. Moreover, this work introduces a two-level parallelism scheme which explores potential design parallelism. Furthermore, design space exploration is performed to identify the appropriate parallelism parameters. Finally, a fine-tuning step is conducted to find the design with not only low II but also low end-to-end latency. Our FPGA implementation of JEDI-net achieves a sub-microsecond initiation interval, which makes the algorithm compatible to LHC conditions where there are strict latency constraints, while enabling improved accuracy.

We make the following contributions in this paper:

- An outer-product based matrix multiplication approach for the GNN-based JEDI-net with custom strength reduction as well as column-major order data layout, which exploits the sparse adjacency matrix and the nature of the column-based algorithm architecture to increase hardware efficiency while reducing design latency.
- A scalable and low latency JEDI-net template which enables the generation of low-latency FPGA designs with efficient resource utilization by HLS tools. We open-source the template[1] for the community.
- A comprehensive evaluation of the proposed method and hardware architecture.

II. Background

A. Interaction Network

The JEDI-net is a graph neural network based on interaction network [5] architecture. The interaction network is a powerful graph based framework for reasoning about objects and relations in complex and dynamic systems. The input to an interaction network is a graph of objects and the relations between them. It learns to capture the complex interactions that can be used to predict future states and abstract physical properties. The acceleration of interaction-network based GNNs has also been studied for charged particle tracking at the CERN LHC on FPGAs [6].

B. JEDI-net for Jet Tagging

The JEDI-net can be represented as a graph, $G = \langle I, R \rangle$ with the nodes, I, corresponding to physics particles, and the edges, R, to the relations. The input of nodes (I) is defined as a $P \times N_o$ matrix, whose columns represent the node's P-length feature vectors, and N_o is the number of particles in a jet. The relations are a triplet, $R = \langle R_r, R_s, {R_r}^T \rangle$, where R_r and R_s are $N_o \times N_E$ binary matrices which index the receiver and sender nodes, respectively. Each column of R_r and R_s is a one-hot vector and it indicates the receiver node's index; R_s indicates the sender similarly. The number of the edges, N_e, is $N_o \times (N_o - 1)$ since JEDI-net is a fully connected graph with directional edges.

Fig. 1 shows the dataflow of JEDI-net. The input I matrix is multiplied by the R_r and R_s matrices and the results are then concatenated to form a B matrix, having dimension $2P \times N_e$. A trainable deep neural network (DNN) function $f_R : \mathbb{R}^{2P} \rightarrow \mathbb{R}^{D_e}$ is then applied to each column of B and produces a matrix E. Then $\bar{E} = ER_r^T$ is conducted in the MMM3 to gather the cumulative effects of interactions received by a given node. The $\bar{E}$ and I are then concatenated to form the C matrix. Each column of the C matrix represents a constituent in the jet, containing P input features and D_e hidden features, representing the combined effect of all the interactions between particles. Another trainable function f_O is introduced to build a post-interaction representation of each jet constituent. It is applied to each column of C to produce the O matrix, having dimension $D_o \times N_o$. A final trainable function ϕ_O returns the probability for that jet to belong to each of the five categories. f_R, f_O and ϕ_O are implemented as 3-layer DNNs using Multi-Layer Perceptrons (MLPs).

[1]https://github.com/walkieq/GNN-JEDInet-FPGA

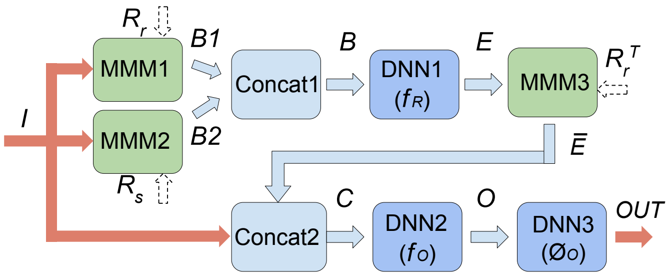

Fig. 1: Overview of the JEDI-net architecture

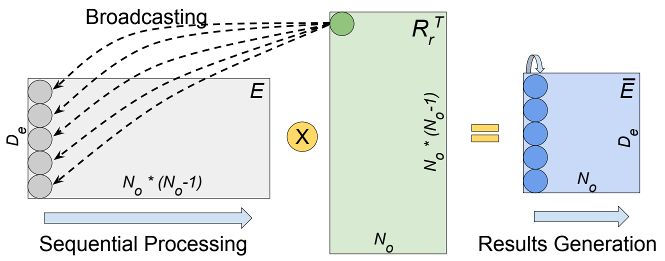

Fig. 2: Outer-product based matrix multiplication

III. Design and Optimization

This section introduces several optimizations to accelerate the GNN for JEDI-net. Although this work focuses on the architecture of JEDI-net for jet tagging, the proposed techniques could be adapted to optimize other GNN-based networks with applications beyond jet tagging.

A. Outer-product based matrix multiplication with strength reduction for JEDI-net

To multiply a matrix by another matrix we often do the inner-product of rows from the 1st matrix and columns from the 2nd matrix. For example, to compute the $\bar{E} = ER_r^T$ in the MMM3 unit, it requires a whole row of the E matrix and a whole column of R_r^T to perform the inner-product for each entry of $\bar{E}$. However, in JEDI-net, the input matrix of the MMM3 unit comes from the output of f_R, as shown in Fig. 1, which produces the results column by column. With an inner-product based MMM in the MMM3 unit, this unit needs to wait for a long time until a whole row of E matrix is ready, resulting in long latency. To solve this issue, this work proposes an outer-product based matrix multiplication for MMM3 to process the $\bar{E} = ER_R^T$ for JEDI-net. Instead of using a whole row from E matrix, now a whole column of E matrix is multiplied by one element from R_r^T matrix to generate the partial result of the first column of result matrix $\bar{E}$ as shown in Fig. 2. The partial result will then be accumulated to form the column of the result matrix. Since the E matrix is generated column by column, MMM3 can start as soon as the first column of E is ready. It largely reduces the waiting time of the MMM3 unit and reduces the design latency. It also

Algorithm 1: The pseudocode of the outer-product based MMM with strength reduction for JEDI-net.

```
Function MMM3( $E$, $\bar{E}$):
    for $i = 0$ to $N_o$ do
        for $k = 0$ to $N_o - 1$ do
            // Reduced from $N_o \times (N_o - 1)$ to $N_o - 1$
              because of the 1-hot feature.
            for $j = 0$ to $D_e$ do
                $index = i \times (N_o - 1) + k$;
                $tmp = (k == 0)\ \ ?\ \ 0 : acc[j]$;
                $acc[j] = tmp + E[index][j]$;
                // Multiplications can be avoided since
                  $R_r^T$ is binary.
            end
        end
        for $m = 0$ to $D_E$ do
            $\bar{E}[i][m] = acc[m]$;
        end
    end
End Function
```

enables a low initiation interval coarse-grained pipeline for the whole design.

To efficiently support the outer-product based MMM, this work adopts a column-major order data format for representing the intermediate results (i.e., 2D matrix arrays) in the JEDI-net instead of the conventional row-major order, which can increase the spatial locality and design performance. Generally a C/C++ based HLS tool, e.g., Xilinx Vivado/Vitis HLS, utilizes a row-major order format when mapping a 2D matrix onto memories, in which the consecutive elements of a row reside next to each other. However, in JEDI-net, the DNN functions f_R and f_O process the input matrix column by column as well as output the result matrix column by column. Besides, the proposed outer-product based MMM also performs the calculation based on columns of input and output matrices. Hence, if the data sits in memory in the default row-major order, it is very time-consuming to collect a whole column for these functions since elements in a column are not contiguous in memory. But conversely, with the column-major order, the input elements can be grouped as a vector (i.e., a whole column as a vector) and can be processed efficiently in these functions with high parallelism because the data can be fetched sequentially.

Moreover, this work performs code transformation using strength reduction to enhance the outer-product matrix multiplication of $\bar{E} = ER_r^T$, which exploits the sparsity patterns of the adjacency matrix of R_r^T as well as the binary feature. It avoids costly multiplications but involves only load and store operations with a small number of additions. The detailed pseudocode of the strength reduction enhanced outer-product based MMM3 has been illustrated in Algorithm 1. The input R_r matrix is binary and each of its columns is one-hot as introduced in Subsection II-B. Thus, the multiplication operations are unnecessary since R_r^T is binary. Besides, it

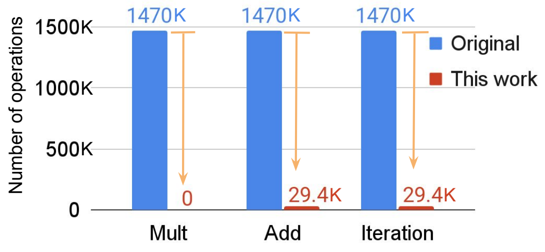

Fig. 3: The reduction in the number of multiplications, additions and iterations for MMM3 in JEDI-net-50p model.

has a fixed pattern, in which the element $(R_r)_{ij}$ is set to 1 when the i^{th} node receives the j^{th} edge and is 0 otherwise. Hence, only $\frac{1}{N_o}$ of total additions are required. Fig. 3 shows that the number of additions can be reduced to 29.4k from 1470k while the multiplications can be totally removed for the $\bar{E} = ER_r^T$ in a JEDI-net-50p model with the proposed approach. Moreover, the input of the adjacency matrices can be also avoided to save memory bandwidth since their patterns can be statically fused into the loop index. The technique of strength reduction can also be applied to compute the MMM1 ($B1 = IR_r$) and MMM2 ($B2 = IR_s$) as shown in our previous work [7]. Our proposed methods not only eliminate the expensive matrix multiplication operation and reduce the iterations but also avoid the input of the adjacency matrices to improve the memory access efficiency, which reduces the design latency and increases the throughput as well as the hardware efficiency.

The latency could be further reduced by inputting multiple columns of E. But this requires multiple f_R hardware units to be deployed and all the previous hardware units can process the corresponding number of columns of their input matrix in each cycle using more hardware resources. We quantify the trade-off between the design initiation interval (II) and the hardware resources from supporting 1-column to several columns in our evaluation section.

B. *Two-level parallelism*

The trade-off between latency, throughput and FPGA resource usage is determined by the parallelization of the design. This work exploits a two-level parallelism scheme. First, we adopt the reuse factor [1] to fine tune the parallelism of the MLPs in a JEDI-net model. The reuse factor is configured to set the number of times a multiplier is used in the computation of a module. The code transformation is performed manually using strength reduction to optimize the matrix multiplications in JEDI-net to avoid multiplications. Hence, only the three MLPs (f_R, f_O, ϕ_O) consume multipliers in the design. We apply the reuse factors R_{fR}, R_{fO} and $R_{\phi O}$ to these three MLPs. This work always tries to achieve extremely low latency by using as many hardware resources as possible, such as unrolling all the layers in the MLPs by adopting a reuse factor value of 1.

Second, this work deploys multiple copies of the f_R unit to further increase the design parallelism. The f_R is applied to each column of the B matrix, as mentioned in Section II-B,

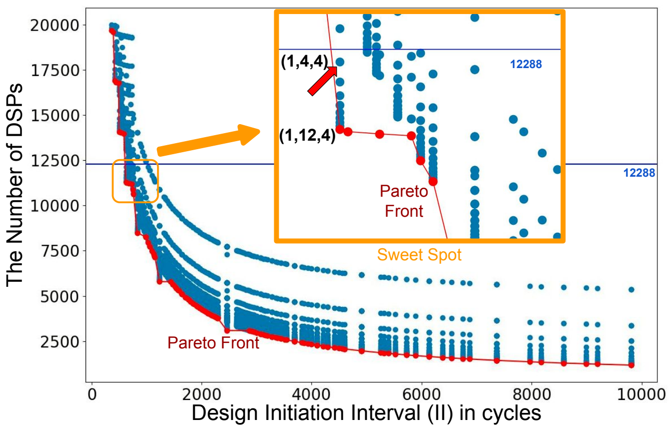

Fig. 4: Design space exploration of JEDI-net-50p model targeting Xilinx U250 FPGAs with various configurations of parallelism parameters (R_{fR}, R_{fO}, N_{fR}).

resulting in a significant number of iterations since there are $N_O \times (N_O - 1)$ columns in the B matrix. Fully unrolling all the iterations requires thousands of hardware copies of f_R, leading to a large hardware resource consumption that will easily exceed a given FPGA. Hence, this work partially unrolls it with a factor, N_{fR}, resulting in N_{fR} copies of the f_R hardware units, each processing a vector of B matrix.

C. Implementation of the hardware accelerator

Fig. 4 shows the results of the design space exploration of the JEDI-net-50p [3] model based on reuse factors and N_{fR} from value 0 to 15. For simplicity, we set the same reuse factor for f_O and ϕ_O which are cascaded. Additionally, both N_{fO} and $N_{\phi O}$ are set to 1 since only DNN1 is the bottleneck. Each blue dot is an explored design while the red dots are Pareto designs, forming the Pareto front. The sweet spot shows the explored design candidates which have a good trade-off between the Initiation Intervals (IIs) and the total DSPs on the targeted U250 FPGA which has 12288 DSPs.

This work splits the whole JEDI-net into several sub-layers and adopts a layer-wise hardware architecture [1, 8, 9, 10, 11, 12] to map all the sub-layers on-chip which is flexible and able to take full advantage of the customizability of FPGAs. In addition, different sub-layers run in a fashion of coarse-grained pipeline to further increase the design throughput. Moreover, we perform the calculation for different sub-layers on their own units using dedicated optimization to achieve low latency and high design throughput.

IV. Evaluation and Analysis

This section presents the evaluation results of the GNN-based JEDI-net on FPGAs demonstrating the scalability of the proposed optimization for GNNs.

A. Experimental setup

This study focuses on JEDI-net-30p [3] models targeting a dataset of 30 particles [13] and JEDI-net-50p models targeting a 50 particles dataset [14]. To study the performance and

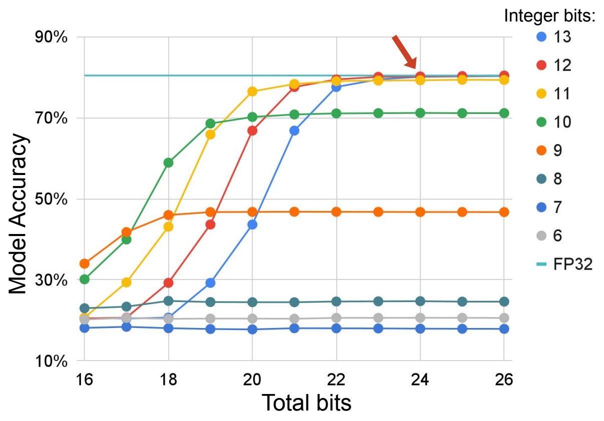

Fig. 5: The JEDI-net model accuracy with various bits

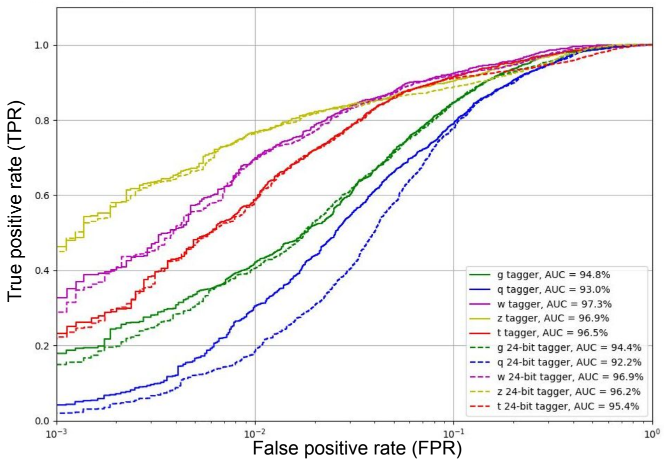

Fig. 6: The AUCs of five jet taggers

limitations of the proposed optimizations and hardware architecture, the design is implemented using Xilinx Vivado HLS 19.2 on a Xilinx Alveo U250 board to do the evaluation and comparison with other implementations. It runs at 200MHz so each cycle is 5ns. FPGA power consumption is reported by the Xilinx Vivado tool. Besides, the weights and inputs of the JEDI-net are quantized to 24-bit: one sign bit, 11 integer bits and 12 fractional bits. It achieves the same accuracy as the floating-point model.

B. Model quantization and accuracy

To find a proper fixed-point precision that can achieve no reduction in the physics performance of the algorithm, we scan the fixed-point precision with total bit widths from 16 to 26 bits and integer bits from 6 to 13, including the sign bit, as shown in Fig. 5. For simplicity, a unified bitwidth is applied. With 24 total bits and 12 integer bits, the fixed-point model effectively achieves the same accuracy as the FP32 floating-point counterpart. In addition, JEDI-net achieves much higher accuracy than the previous work based on DNNs [1, 2] with an accuracy below 75%. We also evaluate the Receiver Operating Characteristic (ROC) curves with the area under the curve (AUC) for the 5 jet classifiers, including gluon, light quarks, W boson, Z boson and top quark, as shown in Fig. 6. The AUC

TABLE I: Resource utilization

Task		LUT	FF	BRAM	DSP
	Available	1728k	3456k	5376	12288
JEDI-net 30P	Used [↓]	303k	104k	284	1831
$(N_{fR} = 1)$	Utilized [%, ↓]	17	3	5	14
JEDI-net-30P	Used [↓]	1158k	246k	1392	11504
$(N_{fR} = 13)$	Utilized [%, ↓]	67	7	25	93
JEDI-net-50P	Used [↓]	886k	31k	1511	7342
$(N_{fR} = 1)$	Utilized [%, ↓]	51	8	28	59
JEDI-net-50P	Used [↓]	1515k	533k	1607	12284
$(N_{fR} = 4)$	Utilized [%, ↓]	87	15	29	99

TABLE II: Performance comparison of JEDI-net-50p FPGA implementations with various parallelism parameters

	U1	**U2**	**U3**	**U4**	**U5**
NN Model / FPGA	JEDI-net-50p / U250				
R_{fR}	1	1	1	1	1
R_{fO}	1	1	1	12	4
N_{fR}	1	2	3	4	4
DSP used	7342 (59%)	10062 (81%)	12894 (104%)	11440 (93%)	12284 (94%)
Latency (cycles)	6519	3474	2493	2565	2131
Latency (μs)	32.60	17.37	12.47	12.83	10.66
II (cycles)	2462	1242	854	650	650
II (μs)	12.31	6.21	4.27	3.25	3.25

of the light quarks tagger (blue lines) using 24-bit fixed-point data representation seems different from the floating-point one, but note there is a logarithmic scale on the x-axis of Fig. 6 and the AUC loss of the q tagger is less than 1%.

C. Resource Utilization

Table I shows the resource utilization of our designs on the U250 FPGA with different parallelism parameters. For the model JEDI-net-30p, the input particle number N_O is 30 with a feature size P as 16, which are defined in the dataset. For JEDI-net-50p, N_O is 50 with the same size of P. The number of edges, N_E, increases dramatically when N_O increases. It is $N_O * (N_O - 1)$ which equals 870 when $N_O = 30$ and 2450 when $N_O = 50$. The two models also have different sized f_R, f_O and ϕ_O.

D. Performance and analysis

To achieve low latency and high throughput, each of the layers in the f_R, f_O, ϕ_O units are firstly fully unrolled. Besides, the proposed strength reduction enhanced outer-product MMM is applied with a column-major data format. The initiation interval of our design reduces from sub-millisecond to a few microseconds. To further improve the latency and II, multiple f_R hardware units are deployed since it is the design bottleneck. But simply increasing the copy of f_R could work but may not be able to result in the optimal design. The II decreases from 12.31μs to 6.21μs when N_{fR} increases from 1 to 2 as shown in Table II. When N_{fR} increases to

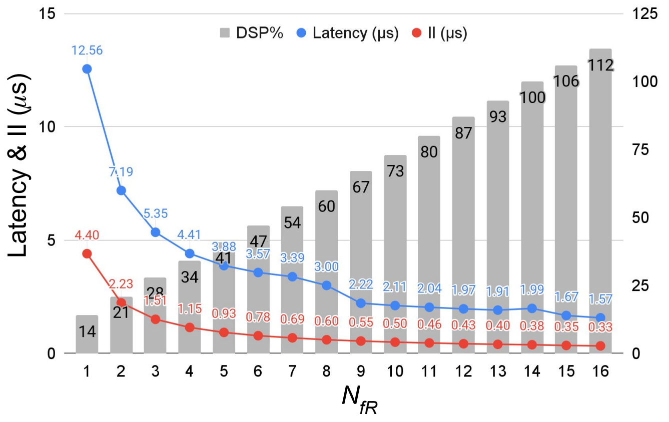

Fig. 7: The latency and initiation interval (II) with various N_{fR} for JEDI-net-30p.

3, the required number of DSPs has exceeded the total DSPs on this FPGA. To solve this issue, we re-allocate some DSP blocks from DNN2 (f_O) and DNN3 (ϕ_O) to DNN1 (f_R) to decrease the design II. TO get the appropriate values of the parallelism parameters (R_{fR}, R_{fO}, N_{fR}), we conduct the design space exploration as shown in Fig. 4. The candidate design is (1,12,4), which is shown as design U4 in Table II.

Although the II is often the most important metric of the design since it decides the throughput, some other metrics are also important, such as the end to end latency. Besides, the number of the DSPs on an FPGA is fixed and if we do not use up all the DSPs, the left ones will be wasted. Thus, if we take the latency into consideration, an extra fine-tuning step can be applied to achieve a better design, e.g., the design U5. First, the U4 is found, which is on the Pareto Front. And then we check if there is a design which has the same II and demands more DSP blocks but does not exceed the total DSPs of the given FPGA. Since the Pareto Front line is based on the II and DSP blocks, it always gives the minimum number of DSP blocks that can achieve the lowest II. But there might be some unused DSPs in this FPGA, which are not sufficient to be used to reduce the II. However, they can still be used to achieve a better latency. Thus, the final design is based on the (R_{fR}, R_{fO}, N_{fR}) of (1,4,4), which achieves not only a smaller II than U3 but also a smaller latency. The JEDI-net-30p model is much smaller than the large one, JEDI-net-50p. With the (R_{fR}, R_{fO}, N_{fR}) as (1,1,1), it only cost 14% DSPs while the JEDI-net-50p costs near 60% DSPs with the same parameters. With the proposed approach, the optimal II of JEDI-net-30p is 0.40μs based on the (R_{fR}, R_{fO}, N_{fR}) of (1,1,13) as shown in Fig. 7.

When the jet tagging is part of the whole processing in the trigger, our approach can still lead to an appropriate set of parameters to get the optimal II and latency with a given hardware budget, as shown in Fig. 7 and Table II. Besides, in a realistic use case, the cardinality of the input dataset might be much smaller. In that case, one would be able to speed up the algorithm even more than what we show in this work, as well as to reduce the resource utilization.

TABLE III: Comparison of the FPGA, CPU and GPU designs

Platform	CPU Gold 6154		GPU GeForce 2080Ti		FPGA U250	
Frequency	3.00 GHz		1.63 GHz		200 MHz	
Technology	14 nm		12 nm		28 nm	
Precision	F32		F32		24 Fixed	
NN Model JEDI-net	50p	30p	50p	30p	50p	30p
Power (W)	103	106	250	245	61.3	37.9
Batch Size	1000					
Average Lat. (μs)	593.1	56.9	16.8	3.8	3.25	0.40
Throughput (KGPS)	1.69	17.6	59.52	263.2	307.7	2500
Power Effic. (KGPS/W)	0.02	0.17	0.24	1.07	5.02	65.96

E. Comparison with GPUs and CPUs

To compare the performance of the proposed design on FPGA with other platforms, we run the JEDI-net models implemented in [3] on Intel Xeon Gold 6154 CPU and NVIDIA GeForce RTX 2080 Ti (CUDA 10.2) based on PyTorch (1.8.1) framework. The CuDNN libraries are used for optimizing the hardware performance on GPUs. Each batch has 1000 graph events (samples) according to [3], so we set the same batch size for all the hardware platforms for a fair comparison. CPU power consumption is measured by the pcm-power utility [15], excluding the DRAM power consumption. GPU power consumption is measured using nvidia-smi utility. We adopt KGPS (Kilo Graphs Per Second), which denotes the number of graph events inferences that run per second, as an indicator of throughput. This work uses the same CPU and GPU implementations like the one in [3]. Compared with the JEDI-net implementation on GPU, our FPGA design is 5.2$\sim$9.5 times faster and consumes 4.1$\sim$6.5 times less power. In terms of the power efficiency, which is denoted as KGPS per Watt, our design is 21$\sim$62 times higher than the GPU implementation. When compared to the CPU implementation, our FPGA implementation is 142$\sim$185 times faster. In addition, our design achieves 306$\sim$397 times higher power efficiency than the CPU implementation. We believe the proposed custom strength reduction can also be applied to CPU and GPU implementations, but the latency profiling shows that the three MMMs cost less than 15% of the total latency. We leave that for future work since it has a limited impact on the conclusions in this paper. The FPGA implementation is faster and more efficient because it is unrolled on-chip with a coarse-grained pipeline and benefits from tailor-made optimizations for the JEDI-net based on the proposed approach.

V. Related work

There have been some studies exploring GNNs for particle physics applications, such as jet tagging [3], charged particle tracking [16], and calorimeter energy measurements [17]. To achieve low latency, FPGAs are involved. [6] extends the hls4ml [1] tool to translate GNNs into FPGA firmware automatically for charged particle tracking. Besides, GarNet [18], a GNN-based algorithm, is proposed for calorimeter energy regression. Our previous work [7] only focuses on accelerating the JEDI-net-30p with an II reduced to 0.6μs. With the newly proposed technique, the II in this work is 50% better.

There are also many studies about general GNN accelerations [19, 20, 21, 22, 23]. AWB-GCN [19] is based on a column-wise-product architecture for GCN acceleration. The updated version I-GCN [22] proposes Islandization, a new runtime graph restructuring algorithm, which can increase the regularity of the adjacency matrix to improve data locality for better performance. BoostGCN [20] presents a novel hardware-aware Partition-Centric Feature Aggregation (PCFA) scheme to enable pipelined execution of GCN inferences on FPGAs. [21] proposes to accelerate GCNs using HLS with several hardware-friendly optimizations. [23] proposes a model-architecture co-design with a light-weight algorithm for temporal GNN inferences on FPGAs. Some previous studies focus on accelerating GNN training [24, 25, 26]. GraphACT [24] introduces an FPGA-based accelerator with a subgraph-based algorithm for Graph Convolutional Networks (GCNs) training. [25] presents an efficient graph sampling accelerator on high bandwidth memory based FPGAs for training GNNs. [26] proposes HP-GNN which maps GNN training on the CPU-FPGA platform automatically. All of these studies utilize the single engine architecture. This work focuses on layer-wise architecture and proposes several novel optimizations for ultra low latency and high throughput. These previous studies are orthogonal to our proposed approach and hardware architecture. Their techniques can be complementary to our approach, which could be extended in future to achieve even lower latency.

VI. Conclusions and Future Work

This paper presents a novel approach for minimizing the initiation interval and latency for the GNN-based JEDI-net on an FPGA. It involves execution optimizing the matrix operations and coarse-grained pipeline to support next-generation low-latency collider trigger systems, key to many fundamental physics experiments including jet tagging. We propose an outer-product based matrix multiplication approach enhanced by code transformation of strength reduction and column-major order representation. Results show up to 9.5 times reduction in latency over the existing GPU-based JEDI-net implementation. Our future work includes exploring the use of new FPGA resources such as the AI Engines [27] and the AI Tensor Blocks [28], and incorporating the proposed techniques into the design and implementation of the data processing architecture for next-generation collider trigger systems.

Acknowledgement — The support of the United Kingdom EPSRC (grant numbers EP/V028251/1, EP/L016796/1, EP/N031768/1, EP/P010040/1, and EP/S030069/1), CERN and Xilinx are gratefully acknowledged.

References

[1] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[2] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, 2021.

[3] E. A. Moreno, O. Cerri, J. M. Duarte, H. B. Newman, T. Q. Nguyen, A. Periwal, M. Pierini, A. Serikova, M. Spiropulu, and J.-R. Vlimant, "JEDI-net: a jet identification algorithm based on interaction networks," *The European Physical Journal C*, vol. 80, no. 1, pp. 1–15, 2020.

[4] C. Collaboration, "The Phase-2 Upgrade of the CMS Level-1 Trigger," *CMS Technical Design Report CERN-LHCC-2020-004. CMS-TDR-021*, 2020. [Online]. Available: https://cds.cern.ch/record/2714892

[5] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," *Advances in neural information processing systems*, vol. 29, 2016.

[6] A. Elabd, V. Razavimaleki, S.-Y. Huang, J. Duarte, M. Atkinson, G. DeZoort, P. Elmer, J.-X. Hu, S.-C. Hsu, B.-C. Lai *et al.*, "Graph Neural Networks for Charged Particle Tracking on FPGAs," *arXiv preprint arXiv:2112.02048*, 2021.

[7] Z. Que, M. Loo, and W. Luk, "Reconfigurable Acceleration of Graph Neural Networks for Jet Identification in Particle Physics," in *2022 IEEE 4rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022.

[8] S. Tridgell, M. Kumm, M. Hardieck, D. Boland, D. Moss, P. Zipf, and P. H. Leong, "Unrolling ternary neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 4, pp. 1–23, 2019.

[9] H. Nakahara *et al.*, "High-Throughput Convolutional Neural Network on an FPGA by Customized JPEG Compression," in *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020.

[10] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 535–547.

[11] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "DNNExplorer: a framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[12] Z. Que, E. Wang, U. Marikar, E. Moreno, J. Ngadiuba, H. Javed, B. Borzyszkowski, T. Aarrestad, V. Loncar, S. Summers *et al.*, "Accelerating recurrent neural networks for gravitational wave experiments," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2021, pp. 117–124.

[13] J. Duarte *et al.*, "HLS4ML LHC Jet dataset (30 particles)," January, 2020. [Online]. Available: doi:10.5281/zenodo.3601436

[14] J. Duarte *et al.*, "HLS4ML LHC Jet dataset (50 particles)," January, 2020. [Online]. Available: doi:10.5281/zenodo.3601443

[15] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[16] X. Ju, D. Murnane, P. Calafiura, N. Choma, S. Conlon, S. Farrell, Y. Xu, M. Spiropulu, J.-R. Vlimant, A. Aurisano *et al.*, "Performance of a geometric deep learning pipeline for HL-LHC particle tracking," *The European Physical Journal C*, vol. 81, no. 10, pp. 1–14, 2021.

[17] S. R. Qasim, J. Kieseler, Y. Iiyama, and M. Pierini, "Learning representations of irregular particle-detector geometry with distance-weighted graph networks," *The European Physical Journal C*, vol. 79, no. 7, pp. 1–11, 2019.

[18] Y. Iiyama, G. Cerminara, A. Gupta, J. Kieseler, V. Loncar, M. Pierini, S. R. Qasim, M. Rieger, S. Summers, G. Van Onsem *et al.*, "Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics," *Frontiers in big Data*, p. 44, 2021.

[19] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt *et al.*, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 922–936.

[20] B. Zhang, R. Kannan, and V. Prasanna, "BoostGCN: A framework for optimizing GCN inference on FPGA," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 29–39.

[21] Y. C. Lin, B. Zhang, and V. Prasanna, "GCN Inference Acceleration using High-Level Synthesis," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2021, pp. 1–6.

[22] T. Geng, C. Wu, Y. Zhang, C. Tan, C. Xie, H. You, M. Herbordt, Y. Lin, and A. Li, "I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1051–1063.

[23] H. Zhou, B. Zhang, R. Kannan, V. Prasanna, and C. Busart, "Model-Architecture Co-Design for High Performance Temporal GNN Inference on FPGA," *arXiv preprint arXiv:2203.05095*, 2022.

[24] H. Zeng and V. Prasanna, "Graphact: Accelerating gcn training on cpu-fpga heterogeneous platforms," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 255–265.

[25] C. Su, H. Liang, W. Zhang, K. Zhao, B. Ai, W. Shen, and Z. Wang, "Graph Sampling with Fast Random Walker on HBM-enabled FPGA Accelerators," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 211–218.

[26] Y.-C. Lin, B. Zhang, and V. Prasanna, "HP-GNN: Generating High Throughput GNN Training Implementation on CPU-FPGA Heterogeneous Platform," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022, pp. 123–133.

[27] "Xilinx AI Engines and Their Applications," in *WP506(v1.1)*, July 10, 2020.

[28] M. Langhammer, E. Nurvitadhi, B. Pasca, and S. Gribok, "Stratix 10 NX Architecture and Applications," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 57–67.