Reconfigurable Acceleration of Graph Neural Networks for Jet Identification in Particle Physics

Zhiqiang Que, Marcus Loo, Wayne Luk
Imperial College London, UK, {z.que, marcus.loo20, w.luk}@imperial.ac.uk

***Abstract*—This paper presents a novel reconfigurable architecture to accelerate Graph Neural Networks (GNNs) for JEDI-net, a jet identification algorithm in particle physics which achieves state-of-the-art accuracy. The challenge is to deploy JEDI-net for online selection targeting the Large Hadron Collider (LHC) experiments with low latency. This paper proposes custom strength reduction for matrix multiplication operations customised for the GNN-based JEDI-net, which avoids the costly multiplication of the adjacency matrix with the input feature matrix. It exploits sparsity patterns and binary adjacency matrices to increase hardware efficiency while reducing latency. The throughput is further enhanced by a coarse-grained pipeline enabled by adopting column-major order data layout. Evaluation results show that our FPGA implementation is 11 times faster and consumes 12 times lower 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

Graph neural networks (GNNs) have been explored and shown excellent performance for particle physics applications, such as jet identification (tagging) [1], charged particle tracking [2, 3], and calorimeter energy measurements [4]. Jet identification, which identifies the nature of the particle that initiated a given collimated cascade, is an important but challenging problem at the CERN Large Hadron Collider (LHC). It identifies all-hadronic decays of high-momentum heavy particles produced at the LHC and distinguishes them from ordinary jets originating from the hadronization of quarks and gluons. [1] proposes JEDI-net, a GNN-based network, which achieves state-of-the-art accuracy on jet identification. It can process the list of jet constituent features directly without assuming specific properties of the underlying detector geometry, and it is insensitive to the order of the input jet constituents, showing many advantages for the LHC experiments.

However, JEDI-net requires very large amounts of computation and suffers from large inference latency [1], which makes its use problematic for real-time deployment in typical level 1 and high level trigger environments. This work proposes several optimizations to accelerate the GNN-based JEDI-net using high-level synthesis (HLS) on FPGAs. First, we propose custom strength reduction for matrix operations based on the characters of the JEDI-net, which avoids the expensive matrix multiplications of the adjacency matrix with the input feature matrix. It largely reduces the computation cost of JEDI-net and boosts the design throughput since the adjacency matrix is very large and matrix multiplication is costly. Second, we propose to adopt a column-major order instead of a conventional row-major order for the data layout of the intermediate results traversed along the hardware datapath, which benefits the computation of JEDI-net. It enables a coarse-grained pipeline with a low initiation interval (II) to improve the design throughput. With both of the above optimizations, our FPGA implementation of JEDI-net achieves a sub-microsecond initiation interval, which makes the model compatible to LHC conditions that has strict latency constraints, while enabling improved accuracy.

To the best of our knowledge, this is the first FPGA design of a GNN-based JEDI-net for jet identification in particle physics experiments. This work would help improve the performance of next-generation collider trigger systems capable of highly accurate jet identification.

We make the following contributions in this paper:

- A custom strength reduction for matrix multiplications for the GNN-based JEDI-net, which exploits sparsity patterns and binary adjacency matrices to increase hardware efficiency while reducing latency.
- A low latency coarse-grained pipeline design for JEDI-net with novel column-major order based data layout to increase design throughput and reduce design latency for jet identification.
- A comprehensive evaluation of the proposed method and hardware architecture.

II. Background

The JEDI-net can be represented as a graph, $G = \langle I, R \rangle$ with the nodes, I, corresponding to the input features of the physics particle, and the edges, R, to the relations. The input particle features (I) are defined as a $P \times N_O$ matrix, whose columns correspond to the node's P-length state vectors, and N_O corresponds to the number of particles in the 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 Rr is a one-hot vector that indicates the receiver node's index; Rs indicates the sender similarly. JEDI-net utilizes a full interconnection through directional edges and the total number of the edges, N_E, is $N_O \times (N_O - 1)$.

The I matrix is multiplied by the R_R and R_S matrices and the two resulting matrices are then concatenated to form the B matrix, having dimension $2P \times N_E$, as shown in Fig. 1. A trainable function $f_R : \mathbb{R}^{2P} \rightarrow \mathbb{R}^{D_E}$ is then applied to each column of B and gives a matrix E. Thus, the cumulative effects of the interactions received by a given vertex are gathered by summing the D_E hidden features over the edges

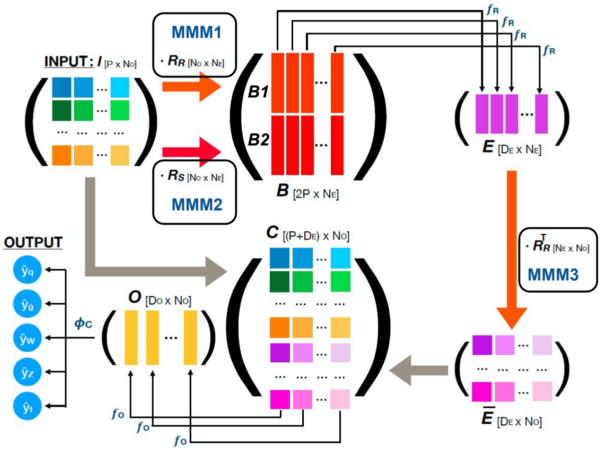

Fig. 1: Overview of the JEDI-net architecture [1].

arriving at it, which is implemented by computing $\bar{E} = ER_R^T$ in the MMM3 unit. The $\bar{E}$ and input matrix I are then concatenated to form the C matrix. Each column of the C matrix represents a constituent in the jet, expressed as a $(P + D_E)$-dimensional feature vector, containing the P input features and the D_E hidden features representing the combined effect of the interactions with all the connected particles. Another trainable function f_O is used to build a post-interaction representation of each jet constituent. It is applied to each column of C to form the matrix O with dimensions $D_o \times N_O$. A final trainable function ϕ_G returns the probability for that jet to belong to each of the five categories. f_R, f_O and ϕ_G are expressed as 3-layer deep neural networks using multi-layer perceptrons (MLPs).

III. Design, Optimization and Implementation

This section introduces several optimizations for accelerating the GNN-based JEDI-net.

A. Strength reduction for matrix multiplications of JEDI-net

The implementation of the JEDI-net in [1] utilizes matrix-matrix multiplication (MMM) operations to compute $B1 = IR_R$ and $B2 = IR_S$, which is costly and time-consuming. One optimization is to exploit the sparsity of R_R and R_S and a dense-sparse matrix multiplication unit could be designed to accelerate the calculation. However, most of the computational operations in these MMMs in JEDI-nets are unnecessary if we exploit the patterns in the R_R and R_S matrices. Both matrices are binary and each of their columns is one-hot. Besides, they have a fixed pattern as shown in Fig. 2. To illustrate the idea, the number of particles is 5 in this figure, but a real design has more particles. The element $(R_R)_{ij}$ is set to 1 when the i^{th} vertex receives the j^{th} edge and is 0 otherwise. Similarly, the element $(R_S)_{ij}$ is set to 1 when the i^{th} vertex sends the j^{th} edge and is 0 otherwise. Because of the fixed patterns and the binary feature, MMMs are not necessary to produce $B1$ and $B2$. First, the multiplications are unnecessary because the R_R and R_S matrices only have binary values. Second, accumulation (addition) operations can be avoided because each column of R_R and R_S is one-hot. Thus, only load and store operations are required to compute $B1 = IR_R$ as well as $B2 = IR_S$. Moreover, the R_R and R_S matrices are not obtained from input; their values are fused into the loops, which reduces memory bandwidth. A similar optimization can be applied to compute $\bar{E} = ER_R^T$ with minimum addition operations. All the zero products are excluded by a careful design using loop and index. Our proposed methods not only eliminate expensive MMM operations to increase the computational efficiency but also avoid the input of the adjacency matrices to improve the memory access efficiency, which reduces the latency. Although this work focuses on the JEDI-net architecture, the proposed custom strength reduction technique can be adapted to optimize other GNN-based networks.

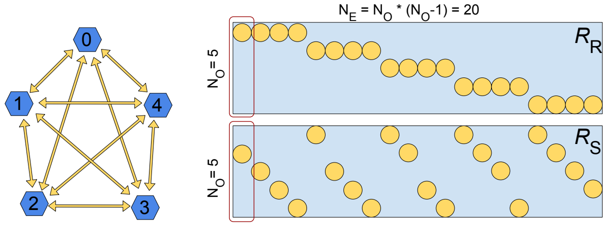

Fig. 2: An example graph with 5 fully connected vertices ($N_O = 5$) and the corresponding 20 uni-directional edges ($N_E = N_O \times (N_O - 1) = 20$) (left) and its receiving matrix R_R as well as sending matrix R_S (right). Each yellow circle is 1 and 0 for all the others in the matrices. The first columns of R_R and R_S matrices have been highlighted using a red frame to show that they are one-hot.

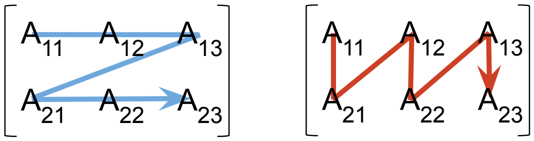

Fig. 3: Row-major (left) and column-major (right) orders.

B. Row-major and column-major orders

The intermediate results in the JEDI-net architecture are captured using two dimensional (2D) arrays representing a matrix as shown in Fig. 1. Row-major and column-major orders (Fig. 3) are two data layout methods which are critical for correctly passing arrays between hardware units. It is important to select an appropriate data layout, since the choice has an impact on performance. When mapping a 2D structure onto a one dimensional (1D) structure (i.e. memory) using a high level synthesis tool (e.g., Xilinx Vivado HLS), often the default data layout is row-major order. However, row-major order for JEDI-net will lead to poor spatial locality and will hinder parallelism since the functions f_R and f_O are applied to each column of the input matrix. With a row-major format, the input data of these functions are not contiguous in memory so it is very time-consuming to collect all the data in a column. However, if the memory is arranged in a column-major layout, iterating over each column brings benefits because the data are accessed sequentially. Thus, this work adopts the column-major order for accelerating the JEDI-net.

Algorithm 1: The pseudocode of the custom matrix operations for JEDI-net.

```
Function MMM_$B$( $I$, $B1$, $B2$):
    for $i = 0$ to $N_O$ do
        for $k = 0$ to $N_O - 1$ do
            for $j = 0$ to $P$ do
                $B1[k + i * (N_O - 1)][j] = I[i][j];$
                $index = (k < i)?k : (k + 1);$
                $B2[k + i * (N_O - 1)][j] = I[index][j];$
            end
        end
    end
End Function
```

C. Implementation of the hardware accelerator

The detailed implementation of the custom strength reduction for the matrix operations is illustrated using pseudocode in Algorithm 1. The computation of $B1$ and $B2$ is shown in the same function in the pseudocode to save space but they are implemented as MMM1 and MMM2 units respectively as shown in Fig. 1. The multiplications and additions have been removed when computing $B1$ and $B2$ as discussed in Section III-A. Furthermore, R_R and R_S are not obtained from the input to reduce memory bandwidth. Their values are statically fused into the loop index.

In addition, this work splits the whole JEDI-net into several sub-layers and adopts a layer-wise hardware architecture [5, 6] to map all the sub-layers on-chip which is flexible to take full advantage of the customizability of FPGAs. Besides, different sub-layers run in a fashion of coarse-grained pipelining to further increase the design throughput. Moreover, we perform the calculation for different sub-layers on their own unit using separate optimization to achieve low latency and high design throughput. This work achieves low latency by using as many hardware resources as possible, such as fully unrolling each layer in the DNNs (f_R, f_O, ϕ_O). We also deploy N_{fR} copies of the f_R hardware unit, each processing a vector of the B matrix. Thus, N_{fR} column vectors can be processed simultaneously, which improves parallelism. Note that the preceding hardware units of f_R in JEDI-net are also updated so that they can provide the N_{fR} vectors in each cycle.

IV. Evaluation and Analysis

A. Experimental setup

This study is based on JEDI-net [1] on a dataset of 30 particles [7]. To study the performance and limitations of the proposed optimizations and hardware architecture, the design is implemented using Xilinx Vivado HLS 19.2 on a U250 FPGA 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 input of JEDI-net are quantized to 24-bit: one sign bit, 8 integer bits and 15 fractional bits. It achieves the same accuracy as the floating-point model.

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 [↓]	810k	205k	924	7417
$(N_{fR} = 8)$	Utilized [%, ↓]	46	5	17	60

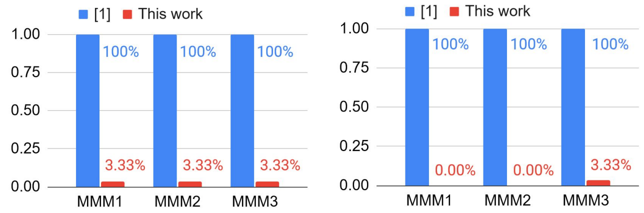

Fig. 4: Number of iterations reduction (left) and number of addition/multiplication operations reduction (right) for the matrix matrix multiplication in JEDI-net.

B. Resource Utilization

Table I shows the resource utilization of our design on the U250 FPGA when the N_{fR} is 1 and 8. The input particle number N_O is 30 and the input feature P is 16, which are defined in the dataset. The number of edges, N_E, increases dramatically when N_O increases. It is $N_O * (N_O - 1)$ which equals 870 when $N_O = 30$. To achieve low latency, the MLPs of f_R, f_O and ϕ_O are all unrolled with a reuse factor [5] of 1 for each dense layer in these models. The design with $N_{fR} = 8$ consumes 4.05 times more DSP blocks than the one with $N_{fR} = 1$.

C. Performance and analysis

We perform code transformation using strength reduction to optimize matrix multiplications, transforming multiplications to only load and store operations. Fig. 4 (right) shows that all the additions for MMM1/2 units are removed while only 3.33% of the additions of the original implementation [1] are required for the MMM3 unit. Besides, strength reduction also decreases the number of the total loop iterations to 3.33% for all the MMM units, which largely reduces the design latency. Furthermore, the run time of C synthesis using Vivado HLS is also reduced by over 3 times when compared with the HLS design without strength reduction.

To achieve low latency and high throughput, each of the dense layers in all the f_R, f_O, ϕ_O hardware units are fully unrolled. Besides, N_{fR} copies of the f_R unit are deployed to reduce the design latency and II (initiation interval) with the improved hardware units in JEDI-net described in Section III-C. When N_{fR} increases, both the design II and latency reduce as shown in Fig. 5. The II reduces from 600 cycles (3μs) to 120 cycles (0.6μs) when N_{fR} increases from 1 to 8. There is a cost: the large design needs 4.05 times more DSP blocks and 2.67 times more LUTs than the small one, as shown in Table I.

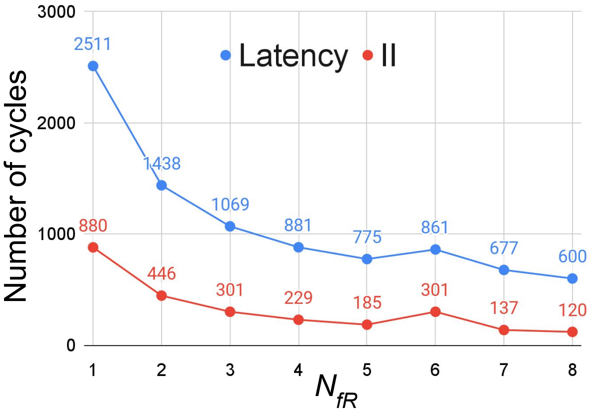

Fig. 5: End-to-end latency and initiation interval (II) cycle numbers with various N_{fR}.

To compare the performance of the proposed design on FPGA with other platforms, we run the JEDI-net model implemented in [1] on Intel Xeon E5-2620 CPU and on NVIDIA TITAN X GPU based on PyTorch framework. The CuDNN libraries are used for optimizing the hardware performance on GPUs. Each batch has 1000 events (samples) according to [1], so we set the same batch size for all the hardware platforms for a fair comparison. We adopt MEPS (Million Events Per Second), which denotes the number of event inferences that run per second, as an indicator of throughput. Compared with the JEDI-net implementation on GPU, our FPGA design is 11 times faster and consumes 12 times less power. In terms of the power efficiency, which is denoted as MEPS per watt, our design is 143 times higher than the GPU implementation. When compared to the CPU implementation, our FPGA implementation is 17 times faster and consumes 1.57 times less power. Besides, our design achieves 172 times higher power efficiency than the CPU implementation. Our FPGA design is faster and more efficient because it is tailor-made for the JEDI-net based on coarse-grained pipelining with low II. This work uses the same GPU implementation as the one in [1]. We believe that the GPU and CPU implementations can also benefit from the proposed optimization of custom strength reduction but the latency profiling shows that the three MMMs cost less than 15% of the total latency. Further CPU and GPU optimisations are beyond the scope of this paper.

V. Related work

There has been much work exploring GNNs [1, 2, 3, 4, 8, 9, 10] for particle physics applications. [3] explores automatic translation of GNN-based algorithms into FPGA firmware for charged particle tracking using the hls4ml [5] tool. Besides, a GNN-based GarNet [10] is proposed for calorimeter energy regression and deployed on FPGAs using hls4ml. There are also some general GNN accelerations [11, 12, 13]. [12] presents an efficient graph sampling accelerator targeting FPGAs with high bandwidth memory (HBM) for training GNNs. In contrast, we focus on optimizing and implementing the GNN-based JEDI-net.

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

	CPU	GPU	This work
Platform	Intel E2620	TITAN X	U250
Frequency	3.4 GHz	1.62	200 MHz
Technology	32 nm	16 nm	28 nm
Power (W)	30.5	233	19.43
Precision	F32	F32	24 Fixed
Batch Size	1000		
Average Latency per Event (us)	70.4	6.8	0.60
Throughput (MEPS)	0.014	0.15	1.67
Power Efficiency (MEPS/W)	0.0005	0.0006	0.086

VI. Conclusions and Future Work

This paper presents a novel approach for minimizing the latency for the execution of the GNN-based JEDI-net by optimizing the matrix operations and coarse-grained pipelining. The approach benefits next-generation low-latency collider trigger systems for many fundamental physics experiments including jet identification. Results show latency reduction of up to 11 times over the existing GPU-based JEDI-net design. Current and future work includes exploring the use of new FPGA resources such as the AI Engines [14] and the AI Tensor Blocks [15], and incorporating the proposed approach into the design of the data analysis 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 is gratefully acknowledged.

References

[1] E. A. Moreno *et al.*, "Jedi-net: a jet identification algorithm based on interaction networks," *The European Physical Journal C*, 2020.

[2] X. Ju *et al.*, "Performance of a geometric deep learning pipeline for HL-LHC particle tracking," *The European Physical Journal C*, 2021.

[3] A. Elabd *et al.*, "Graph Neural Networks for Charged Particle Tracking on FPGAs," *arXiv preprint arXiv:2112.02048*, 2021.

[4] S. R. Qasim *et al.*, "Learning representations of irregular particle-detector geometry with distance-weighted graph networks," *The European Physical Journal C*, 2019.

[5] J. Duarte, S. Han *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, 2018.

[6] Z. Que *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.

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

[8] J. Duarte and J.-R. Vlimant, "Graph neural networks for particle tracking and reconstruction," *arXiv preprint arXiv:2012.01249*, 2020.

[9] A. M. Deiana *et al.*, "Applications and techniques for fast machine learning in science," *arXiv preprint arXiv:2110.13041*, 2021.

[10] Y. Iiyama *et al.*, "Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics," *Frontiers in big Data*, 2021.

[11] B. Zhang *et al.*, "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.

[12] C. Su *et al.*, "Graph Sampling with Fast Random Walker on HBM-enabled FPGA Accelerators," in *31st FPL*. IEEE, 2021.

[13] T. Geng *et al.*, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

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

[15] M. Langhammer *et al.*, "Stratix 10 NX Architecture and Applications," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021.