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 low inference latency [1], which makes its use problematic for real-time deployment on 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 deployable at the collider trigger system that has strict latency constraints, while enabling the system to benefit from 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, $G = (I, R)$ 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, $R = (R_R, R_S, R_R^T)$, 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 $R_R$ is a one-hot vector that indicates the receiver node’s index; $R_S$ 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.
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 sparse matrix multiplication unit could be designed to accelerate the operations. However, most of the computational operations in these MMMs in JEDI-net 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 then input to reduce memory bandwidth; their values are fused into the loops. A similar optimization can be applied to compute $E = ER_R^T$ with minimum addition operations. All the zero products are excluded by a careful design using loop and index. Our proposed methods not only eliminate the expensive MMM operation 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.

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 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.

### 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 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 ($f_R$, $f_O$, $\phi_O$). 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 on a dataset of 30 particles. 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.

**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 $\phi_O$ are all unrolled with a reuse factor 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 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 with no strength reduction.

To achieve low latency and high throughput, each of the dense layers in all the $f_R$, $f_O$, $\phi_O$ 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 earlier. When $N_{fr}$ increases, both the design II and latency reduce as shown in Fig. 5. The II reduces from 600 cycles (3mus) to 120 cycles (0.64us) 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.

---

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

```plaintext
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
```

---

**TABLE I: Resource utilization**

<table>
<thead>
<tr>
<th>Task</th>
<th>LUT</th>
<th>FF</th>
<th>BRAM</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available</td>
<td>1728k</td>
<td>3456k</td>
<td>5376</td>
<td>12288</td>
</tr>
<tr>
<td>JEDI-net 30P</td>
<td>Used [ ]</td>
<td>303k</td>
<td>104k</td>
<td>284</td>
</tr>
<tr>
<td>(N_{fr} = 1)</td>
<td>Utilized [%]</td>
<td>17</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>JEDI-net 30P</td>
<td>Used [ ]</td>
<td>810k</td>
<td>205k</td>
<td>924</td>
</tr>
<tr>
<td>(N_{fr} = 8)</td>
<td>Utilized [%]</td>
<td>46</td>
<td>5</td>
<td>17</td>
</tr>
</tbody>
</table>

---

**Fig. 4:** Number of iterations reduction (left) and number of addition/multiplication operations reduction (right) for the matrix matrix multiplication in JEDI-net.
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 [15] 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.

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


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

<table>
<thead>
<tr>
<th>Platform</th>
<th>CPU</th>
<th>GPU</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>3.4 GHz</td>
<td>1.62 GHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>Technology</td>
<td>32 nm</td>
<td>16 nm</td>
<td>28 nm</td>
</tr>
<tr>
<td>Power (W)</td>
<td>30.5</td>
<td>233</td>
<td>19.43</td>
</tr>
<tr>
<td>Precision</td>
<td>F32</td>
<td>F32</td>
<td>24 Fixed</td>
</tr>
<tr>
<td>Batch Size</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Latency per Event (us)</td>
<td>70.4</td>
<td>6.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Throughput (MEPS)</td>
<td>0.014</td>
<td>6.8</td>
<td>1.67</td>
</tr>
<tr>
<td>Power Efficiency (MEPS/W)</td>
<td>0.0005</td>
<td>0.0006</td>
<td>0.086</td>
</tr>
</tbody>
</table>