AMPLE: Event-Driven Accelerator for Mixed-Precision Inference of Graph Neural Networks

Pedro Gimenes, Aaron Zhao, George Constantinides
Department of Electrical & Electronic Engineering
Imperial College London
London, United Kingdom
{pg519, a.zhao, g.constantinides}@ic.ac.uk

***Abstract*—Graph Neural Networks (GNNs) have recently gained attention due to their performance on non-Euclidean data. The use of custom hardware architectures proves particularly beneficial for GNNs due to their irregular memory access patterns, resulting from the sparse structure of graphs. However, existing FPGA accelerators are limited by their double buffering mechanism, which doesn't account for the irregular node distribution in typical graph datasets. To address this, we introduce AMPLE (Accelerated Message Passing Logic Engine), an FPGA accelerator leveraging a new event-driven programming flow. We develop a mixed-arithmetic architecture, enabling GNN inference to be quantized at a node-level granularity. Finally, prefetcher for data and instructions is implemented to optimize off-chip memory access and maximize node parallelism. Evaluation on citation and social media graph datasets ranging from 2K to 700K nodes showed a mean speedup of $243\times$ and $7.2\times$ against CPU and GPU counterparts, respectively.**

***Index Terms*—FPGAs, graph neural networks, quantization**

I. Introduction

Graphs serve as powerful representations for capturing relationships between entities, which are represented as nodes connected together by edges. This structure enables modeling complex systems such as social networks [1] and recommendation systems [8]. Graph Neural Networks (GNNs) have emerged as an effective approach for processing graph data by learning complex relational information [4, 7].

Inference on GNN models can be divided into two computational phases, (1) **Aggregation** and (2) **Transformation** [2]. In the Aggregation phase, a permutation-invariant function such as summation or mean is applied over the feature embeddings of a node's neighbors. The results are then utilized in the Transformation phase, which consists of a fully-connected layer used to generate the updated feature embedding for each node. While the Transformation phase presents a *highly regular computational pattern*, which can be effectively accelerated on a parallelized device such as a GPU, the Aggregation phase involves many irregular memory accesses due to the random and sparse nature of typical graph data. Additionally, aggregation latency is a function of a node's degree, which follows a highly non-uniform distribution. As such, an efficiently-designed GNN accelerator needs to alleviate the computational irregularity of the Aggregation phase while leveraging the regularity of the Transformation phase [10].

Although CPU memory systems are a mature and highly optimized technology, the sparse structure of graph data renders traditional cache systems less effective, since node aggregation incurs a high number of accesses to non-contiguous memory ranges. Inference on GPUs offers higher performance due to the deep level of parallelism, however, these devices are limited by high-latency memory management mechanisms. Additionally, there is no support for inter-phase pipelining, meaning aggregation results must be stored into off-chip memory before being re-fetched for the transformation phase. Additionally, modern devices have limited support for computation with low-precision numerical formats.

These considerations have motivated the design of several GNN accelerators, such as HyGCN and GenGNN. However, (i) the double-buffering mechanism deployed in HyGCN is not well suited for graph computation due to the non-uniform distribution of node degrees. Under this paradigm, low degree nodes must wait for higher degree nodes before computation can proceed, causing a high number of pipeline gaps. This highlights the need for an **event-driven programming flow**, where nodes are independently allocated resources and scheduled onto the accelerator. Additionally, (ii) neither accelerator offers hardware support for mixed precision. As observed by Tailor et al. [6], the accuracy cost of quantization in GNNs is predominantly due to the aggregation phase and directly correlated to a node's degree. As such, casting low-degree nodes to lower-precision formats while preserving high-degree nodes in high precision leads to reduced memory cost and resource usage at a low cost to model accuracy. Finally, (iii) existing accelerators require on-chip buffering of node embeddings for the entire input graph. As such, these have limited applicability for inference on large graphs ($> 100k$ nodes) where embeddings cannot feasibly be stored on-chip, highlighting the need for a **node-centric pre-fetching system** to hide memory access latency while the accelerator is busy. We address these shortcomings by introducing a novel GNN accelerator, AMPLE, contributing the following:

- We showcase an event-driven programming model for GNN acceleration, enabling the host to program nodes asynchronously through memory-mapped registers.
- We propose an architecture featuring a pool of multi-precision Aggregation Cores connected through a Network-on-Chip for dynamic node allocation.
- We evaluate AMPLE on large-scale social graph datasets

ranging from 2K to 700K nodes, achieving an average speedup of 243× and 7.2× compared to CPU and GPU baselines, respectively.

II. Background

A. Graph Representation

A graph $G = (\mathcal{V}, \mathcal{E})$ is a set of nodes/vertices $\mathcal{V}$ and edges $\mathcal{E}$. The set of feature representations at layer l is denoted by matrix $X^{(l)} \in \mathcal{R}^{N \times D}$, where $N = |\mathcal{V}|$ is the number of nodes and D is the feature size. An element $e_{i,j} = (v_i, v_j)$ present in $\mathcal{E}$ indicates that there is a connection between nodes v_i and v_j, meaning node v_j is contained in the set of i's neighbors, $\mathcal{N}_i$, and v_i is contained in $\mathcal{N}_j$. In an undirected graph, the edge element $e_{i,j}$ corresponds to $e_{j,i}$. The connections in a graph can be represented using an $N \times N$ adjacency matrix, where each element $A_{i,j}$ represents an edge between nodes i and j.

B. Graph Neural Networks (GNNs)

Within a GNN, graph data is transformed over several layers to perform classification and/or regression tasks on the full graph or individual nodes/edges. GNNs can be represented through the Message Passing Mechanism [2], which generalizes the node update law as follows.

$$\mathbf{x}_i^{l+1} = \gamma(\mathbf{x}_i^l, \mathcal{A}_{j \in \mathcal{N}(i)}(\phi(\mathbf{x}_i^l, \mathbf{x}_j^l, e_{i,j}^l))) \tag{1}$$

In the general case, each node aggregates incoming messages produced by a function ϕ, which is equivalent to aggregating neighboring embeddings when $\phi = \mathbf{x}_j^l$. Messages are aggregated through an arbitrary permutation-invariant aggregation function $\mathcal{A}_{j \in \mathcal{N}(i)}$ over the neighborhood of node i, and and an arbitrary transformation function $\gamma(\mathbf{x}_i^l, \mathbf{m}_i^l)$, where $\mathbf{m}_i^l$ is the result of aggregation (i.e. $\mathbf{m}_i = \mathcal{A}_{j \in \mathcal{N}(i)} \phi(\mathbf{x}_i^l, \mathbf{x}_j^l, e_{i,j}^l)$).

1) Graph Convolutional Networks (GCN): emerged as a solution analogous to Convolutional Neural Networks in the computer vision domain [4]. The node update law is as follows.

$$\mathbf{x}_i^{l+1} = W \left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j^l \right) \tag{2}$$

Here, $\mathcal{A}$ is taken as the summation $\mathcal{A} = \sum_{j \in \mathcal{N}_i} \phi(\mathbf{x}_j, e_{i,j})$, with $\gamma(\mathbf{x}_i, \mathbf{m}_i) = W\mathbf{m}_i$, as proposed by Kipf and Welling [4]. The scaling factors are $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$.

2) Graph Isomorphism Networks (GIN): this architecture can provably generate distinct feature updates for two graphs that can be shown to be non-isomorphic through the Weisfeiler-Lehman test [5], maximizing its representational capacity [9].

$$\mathbf{x}_i^{l+1} = MLP \left((1 + \epsilon) \cdot \mathbf{x}_i^l + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j^l \right) \tag{3}$$

As shown in Equation 3, the same aggregation $\mathcal{A}$ is used as in GCN, with $\gamma = MLP\left[(1+\epsilon)\mathbf{x}_i^l + \mathbf{m}_i\right]$, where MLP represents a Multi-Layer Perceptron. In contrast to GCN, GIN does not make use of normalization factors in aggregation (i.e. $\phi = \mathbf{x}_j$), and a residual connection is added after aggregation, which is equivalent to a self-connection in the graph's adjacency matrix. Note ϵ is a small scalar.

3) GraphSAGE: proposed as an inductive framework to generate feature embeddings with high representational capacity for unseen nodes and/or sub-graphs [3].

$$\mathbf{x}_i^{l+1} = W_1 \mathbf{x}_i + W_2 \cdot \left(\underset{j \in \mathcal{N}(i)}{\text{mean}} \sigma(W_3 \mathbf{x}_j^l + \mathbf{b}) \right) \tag{4}$$

As shown in Equation 4, the message passing function ϕ is taken as a fully-connected layer with activation σ over the neighbouring embeddings $\mathbf{x}_j$, $\mathcal{A}$ is taken as the mean, and the transformation $\gamma(\mathbf{x}_i, \mathbf{m}_i) = W_1 \mathbf{x}_i + W_2 \mathbf{m}_i$ where W_1, W_2 are linear projection matrices. The projection parameterized by W_1 can be seen as a scaled residual connection.

C. Neural Network Quantization

Quantization has been widely explored as a method for reducing memory requirements of neural networks. In general, activations are quantized following Equation 5, where q_{min}, q_{max} form the chosen range of representable values, s is the scaling factor to place x into the required range, z is the zero-point (floating point equivalent of the value 0 in the quantized space) and the brackets represent rounding.

$$x_q = \min(q_{max}, \max(q_{min}, \left\lfloor \frac{x}{s} + z \right\rceil)) \tag{5}$$

The $\min$ and $\max$ functions are in place to show that any values beyond the specified range assume the fixed-point value at the limit. Following this, values can be de-quantized by $\hat{x} = (x_q - z)s$, such that $\hat{x}$ is an approximation of the original floating-point value.

1) Quantized Graph Neural Networks: Degree-Quant, proposed by Tailor *et al.*, was one of the first works suggesting mixed precision quantization of GNNs [6]. The authors suggest that the aggregation phase of GNNs is the predominant source of quantization error, which can be observed more heavily in nodes with higher in-degrees, as the absolute magnitude of aggregation grows with the number of neighbors. The growth in aggregation for high-degree nodes affects the q_{max} and q_{min} values, reducing the quantization resolution due to these outliers in the distribution of aggregation results. The authors address this issue by stochastically applying a protection mask at each layer following the Bernoulli distribution. Protected nodes operate in floating-point, while non-protected nodes operate in fixed-point. A node's probability of protection is a function of its degree, interpolated within a parametrizable range $[p_{min}, p_{max}]$, where the graph nodes with minimum/maximum neighbor counts are assigned the limit probabilities.

III. Architecture

The architecture of the AMPLE accelerator is shown in Figure 1. In this section, we describe how event-driven programming, mixed-precision arithmetic and large graph processing are achieved at the circuit level.

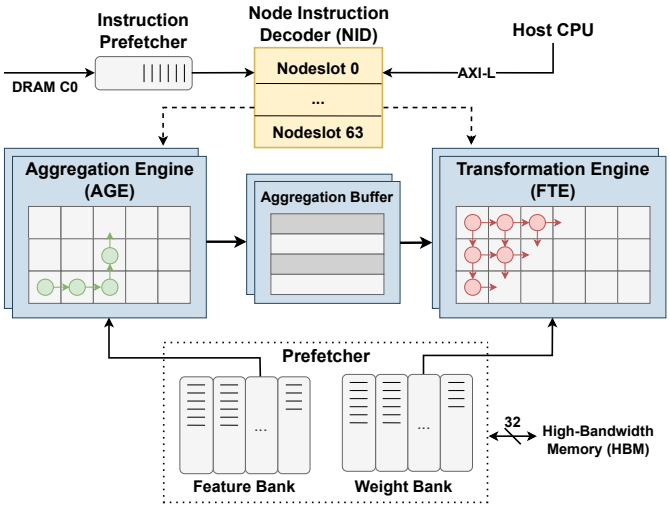

Fig. 1: Architecture of the **AMPLE** accelerator. The NID handles communication with the host device and driving other functional units. The Prefetcher fetches and stores weights and features in local memories. The AGE performs neighbour aggregation through its Network-on-Chip (NoC) design. The FTE performs matrix multiplication between weights and aggregation results in a systolic array.

A. Event-Driven Programming through the NID

The Node Instruction Decoder (NID) is a memory-mapped register bank comprised of a configurable number of nodeslots. Each nodeslot contains the information required to perform a node's aggregation and transformation steps, and a state machine is maintained indicating each node's state. The host device runs concurrently with the accelerator to offload the GNN workload. First, the NID is programmed with a number of global and layer-wise parameters, including node/feature counts and aggregation functions. Subsequently, the host programs the nodeslots and updates values in a mask $available\_nodeslots \in \{0,1\}^n$ where n is the number of nodeslots. While a node is programmed, the accelerator performs aggregation and transformation over previously-programmed nodes. The $available\_nodeslots$ mask is then deasserted independently by the accelerator when the computation is finished. As such, the accelerator supports a node-wise, event-driven computation paradigm.

After a nodeslot is programmed, the NID drives the Prefetcher, AGE and FTE to perform the computation, and updates the node's internal state machine after each step. No further intervention is required from the host, and an interrupt is sent to indicate the nodeslot can be reused. Typical graph datasets often display high variance in execution time per node, depending on neighbour count and numerical precision. Whenever a nodeslot finishes its computation, it can be immediately reprogrammed by the host with the next node. This event-driven control flow requires the host to run concurrently with the accelerator to monitor its state and drive further work when resources are available. Within the NID, nodes running concurrently are serviced with round-robin arbitration to grant access to shared resources within the Aggregation and Transformation Engines.

B. Mixed-precision Arithmetic at Node Granularity

Processing elements of various numerical precisions are arranged in a Network-on-Chip (NoC) architecture within the Aggregation Engine, with the ratio of PEs allocated to each precision being configurable at compile time. Each PE is coupled to a router responsible for transferring packets over the network. Each packet is comprised of a head flit carrying routing payloads, an arbitrary number of body flits carrying data, and a tail flit. Since there is no requirement for communication between PEs of different precisions, these are placed within isolated sub-networks as shown in Figure 2, reducing packet congestion.

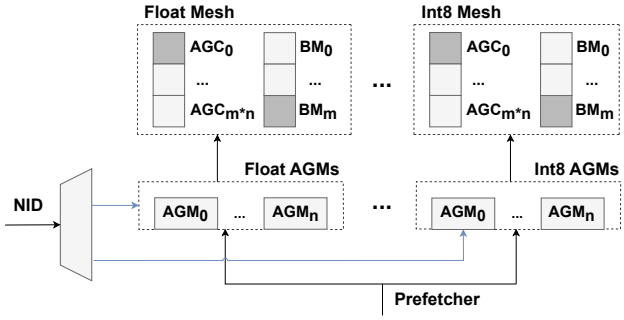

Fig. 2: Architecture of the Aggregation Engine (AGE). Work for each node is handled by an Aggregation Manager (AGM), which allocates a number of Aggregation Cores (AGCs) according to the runtime requirements. Each AGC receives instructions from the Node Instruction Decoder (NID) and feature embeddings from the Prefetcher. Embeddings are transferred to the AGCs through the NoC, and results are buffered by the Buffering Managers (BMs).

As discussed in Section I, static pipelining through the double buffering mechanism leads to pipeline gaps when computing over graphs with high variance in node degree, since low-degree nodes must wait for high-degree nodes to release resources. This is alleviated in the AGE by dynamically allocating processing elements within each aggregation sub-network according to a node's feature count and precision. As such, nodeslots are allocated resources independently of any other ongoing workload, and these resources can be immediately reused upon completion, forming an event-driven programming model.

C. Large Graph Processing

Inference over large graphs is enabled by the Prefetcher, which contains a storage element named "Fetch Tag" for each nodeslot in the NID. Fetch tags for all nodeslots make memory requests concurrently in a two-stage process - first, the list of neighbouring node IDs for each node is stored in the Address Queue, and these are then used as pointers for the neighbouring feature embeddings, which are stored in the Message Queue.

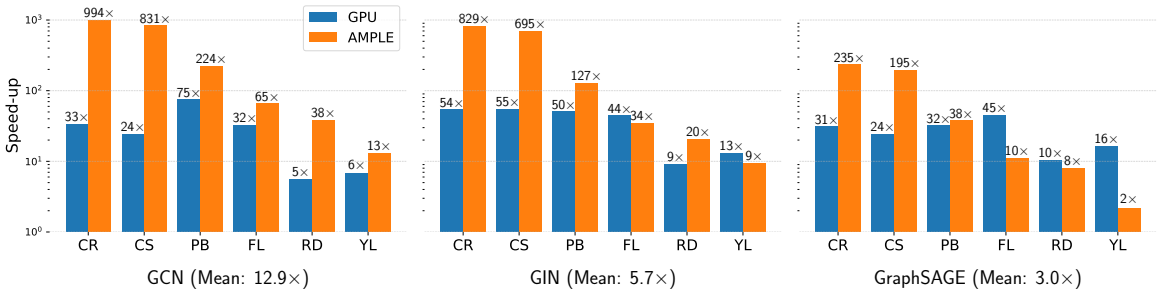

Fig. 3: Inference speedup compared to Intel Xeon CPU baseline obtained on the RTX A6000 GPU and AMPLE simulation. The GPU shows an average speedup of 29.8×, 37.8× and 26.7× across all datasets for GCN, GIN and GraphSAGE, respectively. Equivalent speedups on AMPLE were 361.3×, 285.8× and 81.7×.

The large graph use case is supported a **partial response mechanism**. For nodes with degree higher than the Fetch Tag capacity, the Fetch Tag fills the Message Queue and directly unblocks the AGE to begin the aggregation process. Once aggregation begins, the Fetch Tag is re-enabled and continues to fetch the remaining neighbours, hiding the memory access latency. This mechanism leads to lower storage requirement per nodeslot, allowing a higher number of Fetch Tags in the Feature Bank, i.e. deeper node parallelism.

IV. Experiments

The described accelerator was benchmarked on GCN, GIN and GAT networks (see Section II) over 6 graph datasets were chosen: three small citation graphs and three larger social media graphs. The DegreeQuant algorithm was used to assign the precision for each node in each dataset. In each evaluation, the design was configured to have a number of nodeslots allocated to low precision relative to the ratio of low precision nodes determined by DegreeQuant. Due to the high sparsity in high precision nodes, it was found that allocating a single nodeslot to floating-point was enough to meet the requirement for task accuracy while maximising hardware node parallelism.

A. Performance Analysis

Each model was first benchmarked on the Intel Xeon CPU and RTX A6000 GPU across all datasets, with randomly initialized node features and layer weights. In each case, the mean latency was obtained over 100 trials to account for runtime jitter due to non-deterministic processes. The GPU cache was emptied prior to each prediction step such that latency readings include off-chip memory access for features and weights. GPU warm-up time was not included, meaning inference times are taken after driver initialization is complete. Finally, inference latency on AGILE was obtained from Modelsim 19.2 simulation results at a frequency of 200MHz, obtained for the Alveo U280 card using the Vivado 23.1 toolflow. As shown in Figure 3, AMPLE led to an improvement in mean inference time compared to the CPU/GPU baselines across all models. Table I shows the obtained values for latency and node throughput for GCN.

V. Conclusion

We presented AMPLE, an FPGA-based accelerator for GNNs addressing irregular node distribution in graphs through a novel event-driven programming model. The accelerator's mixed-precision architecture, featuring isolated sub-networks for numerical precisions, maximizes hardware utilization through node-level quantization. Additionally, a partial response mechanism allows for efficient processing of large graphs by hiding memory access latency. Experimental results across six diverse datasets demonstrated average speedups of 361.3×, 285.8×, and 81.7× compared to CPU baselines for GCN, GIN, and GraphSAGE models respectively, while also outperforming GPU implementations by 12.9×, 7.5×, and 3.0×. These results establish AMPLE as a promising solution for high-performance, memory-efficient GNN inference.

TABLE I: Inference time for evaluated datasets using a single-layer GCN model. Mean latency is reported over 100 iterations.

	CPU (Intel Xeon)		GPU (RTX A6000)		AMPLE @200MHz			
	Mean Latency [ms]	Throughput [nodes/ms]	Mean Latency [ms]	Throughput [nodes/ms]	Mean Latency [ms]	Throughput [nodes/ms]	Latency Gain (CPU)	Latency Gain (GPU)
Cora	244.4	11.1	7.2	376.3	0.246	11,022.0	994.8×	29.3×
CiteSeer	244.3	13.6	10.1	330.0	0.294	11,318.6	831.2×	34.3×
PubMed	362.4	54.4	4.8	4,099.5	1.617	12,193.2	224.1×	3.0×
Flickr	475.4	187.8	14.5	6,146.2	7.227	12,350.0	65.8×	2.0×
Reddit	953.3	244.4	171.0	1,362.0	24.6	9,463.6	38.7×	6.9×
Yelp	760.8	942.2	110.9	6461.6	57.5	12,471.7	13.2×	1.9×
Average	506.8	242.2	53.1	3,129.3	15.2	11,469.9	**361.1**×	**12.9**×

References

[1] Andry Alamsyah, Budi Rahardjo, and Kuspriyanto. "Social Network Analysis Taxonomy Based on Graph Representation". In: (Feb. 2021). URL: https://arxiv.org/abs/2102.08888v1.

[2] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

[3] William L. Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Neural Information Processing Systems*. 2017. URL: https://api.semanticscholar.org/CorpusID:4755450.

[4] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (Sept. 2016). DOI: 10.48550/arxiv.1609.02907. URL: https://arxiv.org/abs/1609.02907v4.

[5] Adrien Leman. "THE REDUCTION OF A GRAPH TO CANONICAL FORM AND THE ALGEBRA WHICH APPEARS THEREIN". In: 2018. URL: https://api.semanticscholar.org/CorpusID:49579538.

[6] Shyam A. Tailor, Javier Fernandez-Marques, and Nicholas D. Lane. "Degree-Quant: Quantization-Aware Training for Graph Neural Networks". In: (Aug. 2020). DOI: 10.48550/arxiv.2008.05000. URL: https://arxiv.org/abs/2008.05000v3.

[7] Petar Veličković et al. "Graph Attention Networks". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (Oct. 2017). DOI: 10.48550/arxiv.1710.10903. URL: https://arxiv.org/abs/1710.10903v3.

[8] Shoujin Wang et al. "Graph Learning based Recommender Systems: A Review". In: *IJCAI International Joint Conference on Artificial Intelligence* (May 2021), pp. 4644–4652. ISSN: 10450823. DOI: 10.24963/ijcai.2021/630. URL: https://arxiv.org/abs/2105.06339v1.

[9] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *ArXiv* abs/1810.00826 (2018). URL: https://api.semanticscholar.org/CorpusID:52895589.

[10] Mingyu Yan et al. "HyGCN: A GCN Accelerator with Hybrid Architecture". In: *Proceedings - 2020 IEEE International Symposium on High Performance Computer Architecture, HPCA 2020* (Jan. 2020), pp. 15–29. DOI: 10.48550/arxiv.2001.02514. URL: https://arxiv.org/abs/2001.02514v1.