GreeNX: An Energy-Efficient and Sustainable Approach to Sparse Graph Convolution Networks Accelerators using DVFS

Siqin Liu, *Student Member, IEEE,* Prakash Chand Kuve, and Avinash Karanth *Senior Member, IEEE*

***Abstract*—Graph convolutional networks (GCNs) have emerged as an effective approach to extend deep learning algorithms for graph-based data analytics. However, GCNs implementation over large, sparse datasets presents challenges due to irregular computation and dataflow patterns. Specialized GCN accelerators have emerged to deliver superior performance over generic processors. However, prior techniques that include specialized datapaths, optimized sparse computation, and memory access patterns, handle different phases of GCNs differently which results in excess energy consumption and reduced throughput due to sub-optimal dataflows.**

In this paper, we propose GreeNX, a computation and communication-aware GCN accelerator that uniformly applies three complementary techniques to all phases of GCN. First, we abstract two cascaded sparse-dense matrix multiplications that uniformly process the computation in both aggregation and combination phases of GCNs to improve throughput. Second, to mitigate the overheads of processing irregular sparse data, we develop a dynamic-voltage-and-frequency-scaling (DVFS) scheme by grouping a row of processing elements (PEs) that dynamically changes the applied voltage/frequency (V/F) to improve energy-efficiency. Third, we conduct a comprehensive carbon footprint evaluation, analyzing both embodied and operational emissions for GCNs. Extensive simulation and experiments validate that our GreeNX consistently reduces memory accesses and energy consumption leading to an average $7.3\times$ speedup and $5.6\times$ energy savings on six real-world graph datasets over several state-of-the-art GCN accelerators including HyGCN, AWB-GCN, GCoD, GRIP, IGCN, and LW-GCN.

***Index Terms*—Hardware Accelerator, Graph Convolution Network, Dynamic Voltage and Frequency Scaling, Carbon Footprint.**

I. Introduction

Graph convolutional networks (GCNs) have emerged as a powerful neural network model for representing and processing graph-structured data, finding practical applications across various domains such as social networks, recommendation systems, traffic prediction, knowledge graphs, and more [1]–[9]. Although GCNs leverage the flexible representational capabilities of deep neural networks, they also introduce distinct challenges in computation and dataflow due to their inherently sparse and irregular graph structures.

In GCNs, the bulk of execution latency is typically attributed to computation and memory accesses, which are divided into two main phases: *aggregation* and *combination* [10], [11]. The combination phase, characterized by regular computation, resembles the fully connected layers seen in conventional neural networks. In contrast, the aggregation phase involves irregular computational and memory access patterns due to the sparse nature of the graph structure, posing unique challenges. To address the differing requirements of these phases, existing GCN architectures often employ a hybrid approach, optimizing the accelerator to handle each phase separately [12]–[19]. This approach typically includes specialized datapaths for each phase, advanced sparse computation engines, and a coordinated design that integrates both algorithmic and hardware considerations. Strategies such as preprocessing input graph data and clustering or reshaping inputs are also employed to minimize irregularities in processing [17], [20]. However, some of these methods can present implementation challenges, such as the complexity of preprocessing, or they might result in increased energy consumption and latency due to specialized datapaths. These issues can limit the scalability and efficiency of GCN accelerators.

The computation of graph convolutional network is commonly abstracted to sparse matrix multiplication (SpMM) which is then implemented in GCN accelerators. One of the most distinctive features is the ultra sparsity observed in the matrix operands (e.g., the Nell dataset has 99.9927% sparsity). Prior research has leveraged sparsity by optimizing Sparse Matrix Multiplication (SpMM) by skipping zero elements in outer products, which generally outperforms techniques based on optimizing inner products. However, this method's reliance on generating a large volume of partial outer product results places significant demands on both the capacity and bandwidth of on-chip buffers. Additionally, we have identified that the computation within Graph Convolutional Networks (GCNs) can be conceptualized as two sequential sparse-dense matrix multiplications (SDMMs), where one input matrix is extremely sparse and the other is moderately sparse. Consequently, there is a need for a new SpMM optimization paradigm that efficiently accommodates the outer product approach while also facilitating the effective processing of SDMMs across a range of sparsity levels.

Even with an efficient GCN SDMM operator, challenges persist in handling the irregular non-zeros in graph connections and the corresponding memory accesses during the merge stage of the outer product-based SDMM. These challenges often result in excessive power consumption. While numerous recent studies have developed Dynamic Voltage and Frequency

S. Liu and A. Karanth are with the School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA e-mail: ls847719@ohio.edu and karanth@ohio.edu

P. Kuve is with Timing and Communication Group (TCG) in Microchip Technology Corporation, India e-mail: prakashchand.kuve@microchip.com

Manuscript received August 19, 2024; revised xx, 2024.

Scaling (DVFS) optimizations for DNN accelerators to dynamically adjust to irregular and varying workloads, thereby reducing both static and dynamic power consumption [21]–[26], these approaches predominantly focus on layer-wise and stage-wise optimization. Such optimizations are adept at balancing the varying computational loads across different DNN layers [25] and adjusting to different graph topologies in Coarse-grained Reconfigurable Arrays (CGRA) [26]. However, these methods do not adequately address the unique two-phase computation patterns and dataflow challenges specific to GCNs, which critically impact the efficiency of GCN accelerators [16]. A co-optimization of DVFS with the customized dataflow and computation engine of the GCN accelerator is essential in improving both the energy efficiency and throughput of GCN inferences.

Rapid improvements in computing have been driven by the continuous shrinking of transistor dimensions which has benefited GCN accelerators as well. However, this progress has also contributed to a substantial increase in energy consumption for manufacturing the accelerator itself. With information and computing technologies (ICT) now accounting for nearly 5% of global carbon emissions, a figure projected to rise if not managed properly. As such, a detailed carbon footprint analysis of GCN accelerators is essential, considering both embodied emissions from manufacturing and operational emissions during usage. By optimizing energy efficiency at all stages, from raw material sourcing to active use, we can mitigate the environmental impact of the GCN accelerator design, contributing to more sustainable practices in ICT.

In this paper, we propose **GreeNX**, a *computation and communication-aware* GCN accelerator that uniformly applies three complementary techniques to all phases of GCN. First, we unify the sparse-dense matrix multiplications for GCN variants. In contrast to general SDMM workloads, GCN computation features a highly sparse adjacency matrix, a relatively moderate sparse feature matrix, and a dense weight matrix, making it sub-optimal to conventional SDMM engines. Therefore, we propose an accelerator architecture to implement such SDMM optimization to maximize data reuse and minimize memory access. Second, to mitigate the overheads of processing irregular sparse data, we develop a dynamic-voltage-and-frequency-scaling (DVFS) scheme by grouping a row of processing elements (PEs) that dynamically changes the applied V/F to improve energy-efficiency. To improve workload balance, we propose three DVFS models with corresponding workload percentage thresholds to explore the design trade-off between energy savings and throughput. In addition to energy efficiency, we perform a carbon footprint evaluation of **GreeNX**, considering both embodied and operational carbon emissions. The evaluation leverages a carbon emission model to assess the impact of the proposed DVFS schemes on the overall carbon footprint. We demonstrate how different DVFS modes affect carbon emissions, with the potential to select the most environmentally efficient mode during runtime based on workload intensity. The major contributions of this work are as follows:

1) **Optimize Sparse-dense Matrix Multiplication:** We abstract GCN variants into two unified SDMMs with different sparsity ratios and propose a row-wise SDMM-based hardware architecture to efficiently and uniformly process both aggregation and combination phases. In contrast to conventional SpMM engines [27], [28], we customize the hardware to harness the zero-skipping with the sparse matrix operand and maximize the data reuse with the dense matrix operand, leading to energy-efficient processing of GCN workloads.
2) **Apply DVFS to the SDMM Engine:** We implement a DVFS scheme in GreeNX that applies power gating to PEs during periods of low computation to save static power and dynamically scales voltage and frequency during periods of medium to high computation. This approach alleviates workload imbalance from the SDMM mechanism and further reduces energy consumption.
3) **Conduct Carbon-aware Performance Evaluation:** We conduct a comprehensive evaluation of the carbon footprint for GreeNX, assessing the impact of different DVFS modes on both embodied and operational emissions. This evaluation helps identify the optimal DVFS mode that minimizes the carbon footprint, contributing to more sustainable ICT practices.

II. Background

A. GCN basics

In several real-world applications, graph data are inherently embedded into object features that describe the connections between the objects. Therefore, it is appealing to extend deep learning capabilities and execution efficiency for graph data via Graph Neural Networks (GNNs). Among the many variants of GNNs, GCNs [29] have seen rapid interest due to the high efficiency of neural network-based graph processing. GCNs follow the neighborhood aggregation scheme, where the feature vector of each vertex is computed by recursively aggregating and transforming the representing vectors of its neighbor.

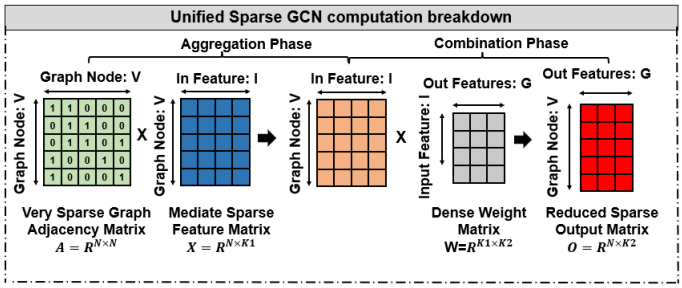

Fig. 1. **GCN Computation Breakdown.** GCN variants can be abstracted into two uniform sparse-dense matrix multiplications (SDMMs) with different sparsity ratios.

Aggregation is the most notable feature that distinguishes GCN from other neural networks. Convolution in a graph aggregates information from the neighboring nodes, applies a specific aggregation function, and embeds a new feature as the output. To be more specific, X is the feature matrix of the initial graph with dimension N $\times$ M. N is the number of vertices and M is the number of features for each vertex. An

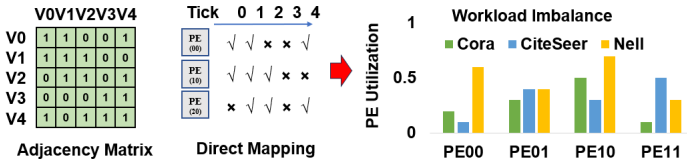

Fig. 2. Illustration of workload imbalance in GCN direct mapping on a 2x2 PE mesh array. The adjacency matrix (left) highlights the sparse and irregular patterns characteristic of the aggregation phase, which mainly contribute to workload imbalance. This imbalance is evidenced by the varying PE utilization across different datasets (right)

adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are connected or not in the graph. One example can be found in Fig. 1(a) where a non-zero element A_{ij} indicates an edge from i to j. The degree matrix D is a diagonal matrix that contains the information about the degree of each vertex i.e. the number of edges attached to each vertex. The layer-wise forward propagation of GCN is formulated as follows:

$$\begin{aligned} Aggregate(X_i) &= \sum_{j=0}^{N-1} A_{ij} X_j \\ Aggregate(X) &= (\bar{D}^{-0.5}\bar{A}\bar{D}^{-0.5})X \\ X^{k+1} &= f(X^k, A) = \sigma(\bar{D}^{-0.5}\bar{A}\bar{D}^{-0.5}X^k W^k) \end{aligned} \tag{1}$$

$$Combination(H) = f(W^k X^k) = \sigma(W^k X^k) \tag{2}$$

where A_{ij} is an element of adjacency matrix A, X_{ij} is an element of vertex feature matrix X, and D is the degree (diagonal) matrix of the current vertex. X^k represents the feature vector of vertex X at the kth layer. The $Aggregate$ function aggregates multiple feature vectors from neighbors whereas the $Combination$ function transforms the feature vector of each vertex to another feature vector using a multi-layer perceptron (MLP) neural network.

A GCN computation example is shown in Fig. 1. Variants of GCN models can be described by using Equation 1. The difference between various GCN models can be derived by transforming the adjacency matrix. GCN model variants can be represented by $X^{k+1} = f(X^k, \bar{A}) = \sigma(\bar{A}X^k W^k)$, where $\bar{A}$ dictates the type of GCN model; X^0, IM, W, X^1 denote the input feature matrix, intermediate output matrix, GCN weight matrix, and the output feature matrix, respectively. This abstraction offers convenient guidance to design efficient GCN accelerators, in which architects only need to consider the characteristics of the adjacency matrix to accommodate different GCN models.

B. Workload Imbalance of GCNs

Sparse matrix multiplications (SpMMs) are ubiquitous in numerous algorithms, including graph analytics, hybrid linear solvers, Markov clustering, searching algorithms, and machine learning. In general, the primary computational operations used in GCNs can be summarized as a series of multiplications between sparse and dense matrices (two cascaded SDMMs). The graph data structure inherently dictates the irregularity and sparsity of these cascaded SDMMs, thus becoming a bottleneck in scaling the performance of GCN accelerators.

Domain-specific architectures can mitigate the challenges posed by sparse zero-valued operands, which hinder parallelism and throughput on general-purpose platforms, by customizing the data structure and implementing efficient indexing algorithms. However, these solutions introduce significant challenges related to workload imbalance. As shown in Figure 2, PEs are only actively used when the corresponding entries of the adjacency matrix are non-zero. Although alternative mapping algorithms can offer optimizations over direct mapping, the inherent features of sparsity and irregularity in GCN computation remain unchanged. An analysis of such workload imbalance on a 2×2 PE mesh array across three GCN datasets is provided in Figure 2. PE utilization not only varies greatly among different PEs but also across different datasets. While the sparsity presents opportunities for hardware architects to optimize power savings by avoiding zero-valued computations, it also poses significant challenges in addressing these workload imbalance issues.

C. Dynamic Voltage and Frequency Scaling (DVFS)

DVFS techniques have been applied to hardware accelerators in prior work [22], [23], [30]. In the GCN accelerator, each PE is a simple but independent computing unit that is responsible for a chunk of computation assigned by the scheduler. DVFS can be applied at different levels of granularity. At the coarse grain, DVFS can be applied where the power manager monitors the GCN workload and adjusts the precision of the GCN model by tuning the DVFS setting of CPUs or GPUs [22]. It trades off both the model accuracy and throughput for energy savings. Applying DVFS to the PE level at finer granularity provides an energy-efficient approach for GCN accelerators regardless of the adjacency matrix irregularity. In GCN architecture, the sparsity in the adjacency matrix leads to numerous irregularly located zeros, which impose challenges for GCN accelerators to achieve optimal speedup and energy efficiency. Instead of processing sparse data in a regular manner with dedicated hardware [16], architects can track the runtime data pattern of both operand matrices. By scaling down the supply voltage and corresponding frequency for sparse workloads and scaling up the voltage and frequency for dense workloads, the DVFS technique can be applied to the PE array to exploit the irregular sparsity feature in order to save energy and address the workload imbalance problem.

III. Proposed Architecture

A. Microarchitecture

The proposed architecture of GreeNX is designed to optimize the processing of graph convolutional networks through a coordinated microarchitecture that effectively leverages dynamic voltage and frequency scaling (DVFS) and sparse density matrix multiplication (SDMM). As depicted in Fig. 3(a), the architecture centers around a Processing Element (PE) array that is intricately connected through a hierarchical mesh Network-on-Chip (NoC), enabling data communication required by the GCN computation. The Global Buffer acts as the main storage area and is segmented to accommodate different types of matrix data in GCNs. Adjacent to the buffers,

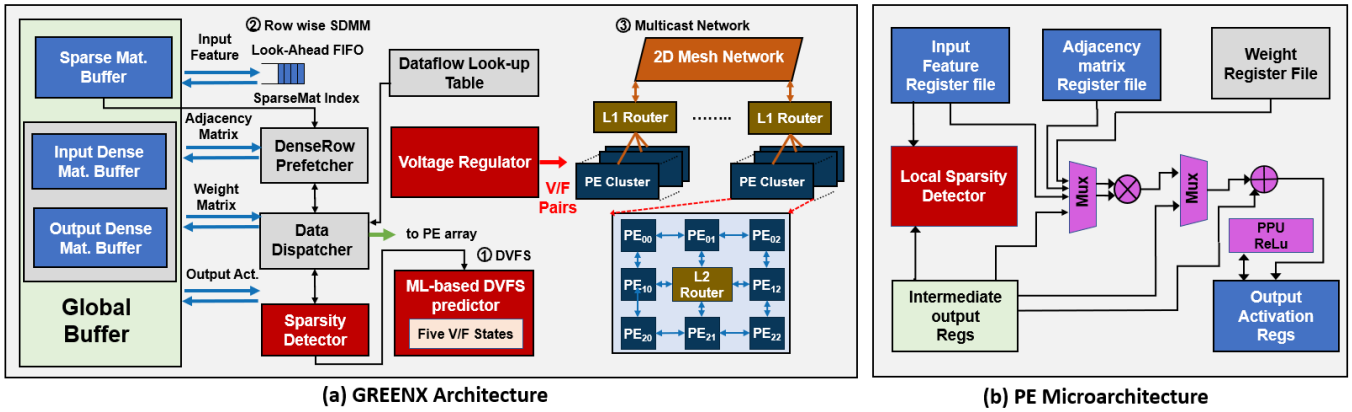

Fig. 3. (a) An overview of GreeNX architecture consisting of the three optimization techniques, (1) DVFS, (2)row-wise SDMM, and (3) multicast network. (b) shows the internal PE microarchitecture consisting of an adjacency matrix, weight, input feature, intermediate output, output activation registers, a sparsity detector, and arithmetic units.

the *DenseRow Prefetcher* plays a role in pre-loading data into the PE array to minimize idle times due to indexing overheads of the sparse representation. The *Data Dispatcher* manages the dataflow within the architecture, ensuring that data is efficiently routed and exhaustively deployed between the buffers and the processing elements according to the computation requirements.

The *Sparsity Detector* and the *Dynamic V/F Generator* are critical to implementing DVFS within the architecture. The Sparsity Detector analyzes the workload dynamically to identify opportunities for voltage and frequency scaling based on the sparsity of the data being processed. The Dynamic V/F Generator then adjusts the voltage and frequency settings for each PE cluster accordingly, optimizing power consumption without compromising performance. The PE array is connected in a mesh-based network. Each PE cluster has an independent power and clock supply to support fine-grained or coarse-grain DVFS scheme.

The PE microarchitecture, illustrated in Fig. 3(b), features distinct local memory units tailored to different data types, facilitating parallel processing of various dataflow variants that necessitate specialized storage. The datapath is segmented into three pipeline stages: DVFS prediction, buffer access, and computation. The computation stage is equipped with a 32-bit multiplier and a 64-bit accumulator, designed to execute a multiplication and accumulation (MAC) operation within a single cycle efficiently. The post-processing unit (PPU) activates to apply the ReLu activation function to the output following the accumulation phase. If the accumulation is not complete, the PPU functions merely as a conduit for the intermediate output, forwarding it to adjacent PEs for further processing.

B. DVFS control system

Implementation of DVFS Control: In our proposed architecture, the grouping of PEs for DVFS is implemented at the hardware layer. Specifically, each PE cluster, containing 8 PEs, shares a common independent V/F supply, as shown in Figure 3(a). A detailed schematic of the voltage regulator and power multiplexer design for each PE cluster is presented in Figure 5. Within each cluster, an L2 router with a full-switching crossbar interconnects the PEs to support efficient intra-cluster communication. DVFS control is managed on the basis of the PE cluster as the basic unit. Clusters are further interconnected through an L1 router, forming a 2D mesh topology. Since different PE clusters operate under independent V/F domains, asynchronous FIFOs are used in each channel of the L1 router to support data communication across V/F domains.

We propose three DVFS models with corresponding workload percentage thresholds to explore the design trade-off between energy savings and throughput. We further extend the proposed DVFS scheme with coarse-grained configuration to explore the design trade-off between DVFS performance and implementation overhead. Each DVFS model consists of one inactive state (power-gated) and three active states. In an inactive state, the voltage supply to the specific PE and its outgoing interconnection is reduced to 0 V with no clock applied to the PE. V/F pair for one PE may switch for each epoch, which has to be short enough to maintain the granularity of DVFS control, and long enough to allow the power supply switching latency. Determining the optimal epoch size is a challenge. After testing several epoch sizes, we set 50 cycles for a relatively balanced design point [31]. PE in an active state can operate in any one of the three available voltage levels. The V/F pairs used are $0.8V/2.75ns, 1.0V/2.25ns, 1.2V/1.8ns$, which are numbered as V/F modes 2-4 with a power-gated state as mode 1. These voltage and frequency pairs are commonly configured in DVFS-supported processors or accelerators [32]. Due to the highly sparse nature of GCNs, we introduce a baseline mode that is maintained constantly at a high voltage level and update the threshold values to fit in the GCN workload distribution as shown in Table I.

Prediction algorithm: Instead of setting up the expected V/F pair for the current workload, Our DVFS scheme predicts the V/F pair requirement of the next epoch based on the workload information of the current epoch. We devise a neural network machine learning method simply based on the previous workload distribution to predict the next epoch. The implementation overheads are a low number of integer

TABLE I
WORKLOAD DISTRIBUTION AMONG DIFFERENT MODELS AND DVFS MODES.

DVFS Models	0 V	0.8V/ 2.75ns	1.0V/ 2.25ns	1.2V/ 1.8ns
Baseline	0%	/	/	>0%
Power-Saving	<10%	10-85%	85-90%	>90%
Balanced	<10%	10-30%	30-80%	>80%
Performance	<10%	10-45%	45-50%	>50%

multiplication, rather than millions of MAC operations in reinforcement learning methods [33].

Algorithm 1 details our DVFS controlling procedures, which utilizes a compact multi-layer perceptron neural network to dynamically adjust voltage and frequency settings based on predicted workload characteristics for each epoch. The perception network consists of an input layer with 7 neurons, a fully connected middle layer with 128 hidden neurons, and a classification output layer equipped with 5 Softmax outputs for predicting V/F mode, which collectively approximate the relationship between input feature sets and labeled voltage/frequency (V/F) modes. The perception is trained offline using runtime data derived from simulations of real-world GCN datasets such as Cora, Citeceer, Ogbn-ArXiv, Reddit, Pubmed, and Nell mapped onto the PE array. The learned parameters are directly used as inference by the DVFS predictor in GreeNX.

In Algorithm 1, All PEs are initialized by a voltage of 1.2V and a clock cycle of 1.8ns (Line 1). The input features are collected through the Sparsity detector of each PE and calculated as an indicator of workload sparsity and computational intensity (Line 2 to 8). These collective data are fed into the perceptron neural network that classifies and predicts the optimal V/F mode (Line 10 to 25). As per the predictions from the trained neural network, DVFS settings are adaptively modified during runtime to optimize power consumption and maintain high computational efficiency across different PE islands (Line 9).

Dynamic voltage and frequency generation: The overview of the DVFS system architecture is shown in Fig. 4(a). The frequency and power modules generate voltages and frequencies for the PE array based on Table I, configured via signals F_{con} and V_{con} from the controller. The frequency module's sub-architecture is illustrated in Fig. 4(b), where a Phase-Locked Loop (PLL) control block, implemented as a finite state machine (FSM), regulates the local clock domain by driving the PLL request signal. The feedback clock (fb) ensures correct phase locking, and clock buffers (depicted as triangles) serve as entry points to the clock distribution network.

Previous works have proposed hierarchical power delivery systems to optimize system performance and latency [34]. However, equipping each PE with a dedicated voltage multiplexer introduces latency. To address this, we propose a global Single-Input-Multiple-Output (SIMO) converter and PE-wise power multiplexers, as shown in Fig. 5. We assume a 1.5V off-chip voltage supply. The SIMO converter, similar to an inductor-based buck converter, generates multiple output voltages using a single inductor (L1) and capacitors

Algorithm 1 DVFS Control Algorithm for Sparse GCN Workloads

Input: Graph (V, E); input features $(X_v, \forall v \in V)$; GCN depth K; weight matrices (W^k); aggregator function; neighborhood function.
Output: Vector representation z_v for all $v \in V$, and voltage/frequency settings for each PE cluster.
1: Initialize voltage to 1.2V and clock frequency to 1.8ns for all PE clusters.
Feature Collection:
2: $W_t = \sum_i w_i$ **where** $(w_i \neq 0)$
3: $A_t = \sum_j a_j$ **where** $(a_j \neq 0)$
4: $M_t = W_t \times O^2 - F \times (F - X_t)$ {F and O are input/output feature sizes}
Edge Update:
5: **for** each vertex v in the active vertex list **do**
6: Update edge $e(u, v) = \text{EdgeUpdate}(h_v^{k-1}, h_u^{k-1}, W_{\text{edge}})$
7: Store $e(u, v)$ and update vertex state.
8: **end for**
DVFS Control in Aggregation Phase:
9: **for** each PE cluster h **do**
10: Set default $V_h = 1.2$V and highest frequency.
11: Perform SDMM computation: $X_l = \text{SDMM}(A_t, W_k, X_t)$
12: Detect sparsity via Sparsity Detector: $AM_{ij} = \text{SparsityDetect}(W_t, A_t, M_t)$
13: **if** $AM_{ij} = 0$ for all j **then**
14: Power-gate the current PE cluster.
15: **else**
16: Set $V_h = 1.0$V for moderate workload.
17: **end if**
18: **if** $\sum_j AM_{ij} < \alpha$ **then**
19: Set $V_h = 0.8$V for light workload.
20: **else**
21: Maintain $V_h = 1.2$V for heavy workload.
22: **end if**
23: **end for**

(C1–C3), either on-chip or off-chip. Reference voltages Vref1 (0.8V), Vref2 (1.0V), and Vref3 (1.2V) are derived from a resistor divider. Comparators A1–A3 regulate output voltages (Vout1–Vout3) relative to the references and control the PWM switch controller. The controller senses input current and generates switching signals (SP, SN, S1–S3) to maintain voltage regulation. The regulated outputs are distributed to PE arrays via Power-MUXes, enabling voltage selection per cluster. To balance DVFS overhead and scalability, we implement a coarse-grained DVFS scheme by grouping PEs into rows or columns.

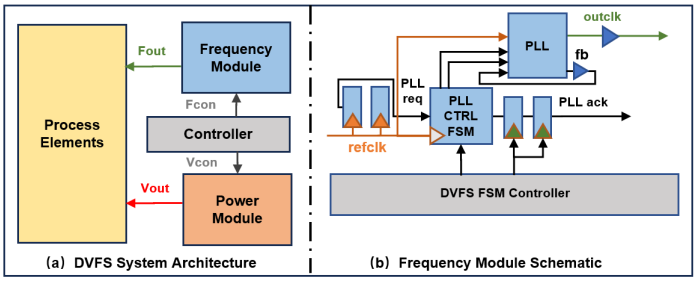

Fig. 4. (a) DVFS system architecture overview (b) Phase-locked-loop (PLL) based DVFS frequency generation schematic.

C. SDMM optimization for DVFS implementation

This section presents the unified SDMM for both the aggregation and combination phases of GCNs introduced in Section II-B. Our approach significantly enhances memory

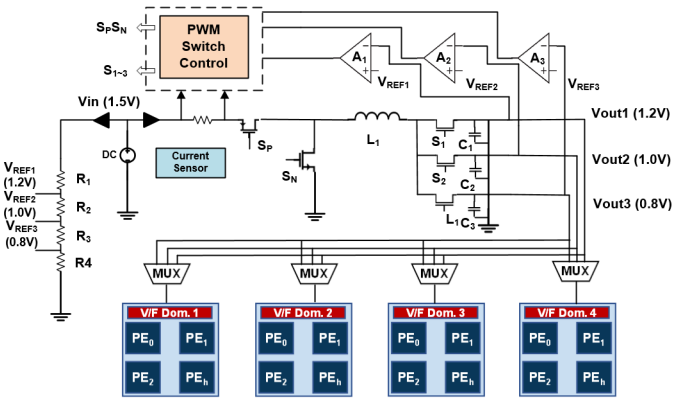

Fig. 5. Single-input-multiple-output (SIMO) based DVFS Power generation and delivery schematic.

efficiency by maximizing the reuse of non-zero elements and eliminating redundant accesses. A key challenge in sparse inner product multiplication lies in the selective performance of multiplications across matched non-zero indices, leading to inefficiencies. Our proposed architecture addresses this issue by leveraging Dynamic Voltage and Frequency Scaling (DVFS) at various levels of granularity. The optimized SDMM, enhanced with DVFS control, outperforms traditional matrix multiplication methods by avoiding the need for index matching. This results in increased reuse of non-zero elements and reduces the frequency of fetching columns or rows from memory.

Fig. 6 shows an example GCN computation and the corresponding sparse adjacency matrix A, input feature matrix X^0, intermediate output matrix IM, weight matrix W, and the output feature matrix X^1, which can be used as inputs for the next layer. We break the SDMM into three DVFS implementation scenarios, that is, element-wise granularity, row-wise granularity, and column-wise granularity. In element-wise DVFS implementation, each element of adjacency matrix A in a row is indexed step by step and multiplied with the corresponding feature matrix elements X^0 to generate the intermediate output matrix IM. The sparsity detector identifies zero-valued elements in A and subsequently skipping the dataloading of the associated input feature elements from X^0. This selective loading significantly reduces unnecessary memory accesses and computational demands, enhancing the overall efficiency of the matrix multiplication operation.

The second SDMM of IM and W follows the same pattern to generate the output matrix X^1. Without complex sparse data encoding and decoding, element-wise SDMM can thoroughly eliminate zero-valued operation by zero-skipping as shown in the $T3$ time stamp. Then, the DVFS system controls the current PE to the power-gating V/F stage to further reduce both dynamic and static power consumption. Due to the large size of the adjacency matrix in graph datasets and stringent on-chip memory footprints, only a chunk of data (tile) can be processed parallel on the PE. Therefore, we highlight the tile size in grey color for both the SDMMs. In row-wise DVFS implementation, we increase the throughput by processing a row of matrix A at a time. Each row of A is reused inside the PE array until all feature matrix elements are traversed, avoiding redundant off-chip memory access. Instead of skipping zero-valued operation for each index, row-wise SDMM treats each row of matrix A as a set of workloads and scales the voltage and frequency supply based on the sparsity detected in the current working epoch. The column-wise DVFS implementation is implemented when the row dimension of the adjacency matrix is too large to fit into the tile. As shown at the bottom of Fig. 6, each column is fetched as sparse working loads and processed by the PE array with scaled V/Fs accordingly.

IV. Carbon Footprint Analysis

The carbon footprint of the hardware accelerator consists of the operational and embodied emissions. The operational emissions are computed as the product of the energy consumed by running the target neural network workloads and the carbon intensity on the hardware. The embodied carbon footprint of hardware is measured as the product of the die area and carbon emitted per unit area manufactured. The proposed DVFS not only reduces energy consumption but also enhances carbon efficiency based on the intensity of runtime workloads.

A. Operational Carbon Footprint

The operational carbon footprint of a hardware accelerator during a DNN inference task depends on multiple factors, including the algorithm's efficiency, the number and power of processing elements (PEs), the duration of processing, and the efficiency of the data center's power and cooling systems. The carbon footprint can be estimated by the following formula:

$$\text{Carbon}_{\text{footprint}} = \text{Energy}_{\text{consumed}} \times \text{CO2e}_{\text{intensity}} \tag{3}$$

Energy Consumption: The energy consumed by the system can be modeled as a function of runtime, the number of active cores, their utilization, memory usage, and the power consumption of these components, adjusted by the Power Usage Effectiveness (PUE) of the data center:

$$\text{Energy}_{\text{consumed}} = t \times (n_c \times P_c \times u_c + n_m \times P_m) \times \text{PUE} \tag{4}$$

where t is the runtime (in seconds), n_c is the number of processing cores, u_c represents the core utilization factor (ranging from 0 to 1), P_c is the power consumption of a core (in watts), n_m is the memory size (in gigabytes), and P_m is the power consumption of memory (in watts). The PUE accounts for the efficiency of the data center's power usage.

B. Embodied Carbon Footprint

To assess the embodied carbon footprint (ECF) of a hardware accelerator, emissions are considered at the component level for each processing element, Network-on-Chip (NoC), memory (DRAM), and storage element. Each integrated circuit (IC) also contributes additional emissions due to packaging. The embodied carbon footprint for a hardware platform can be estimated as follows:

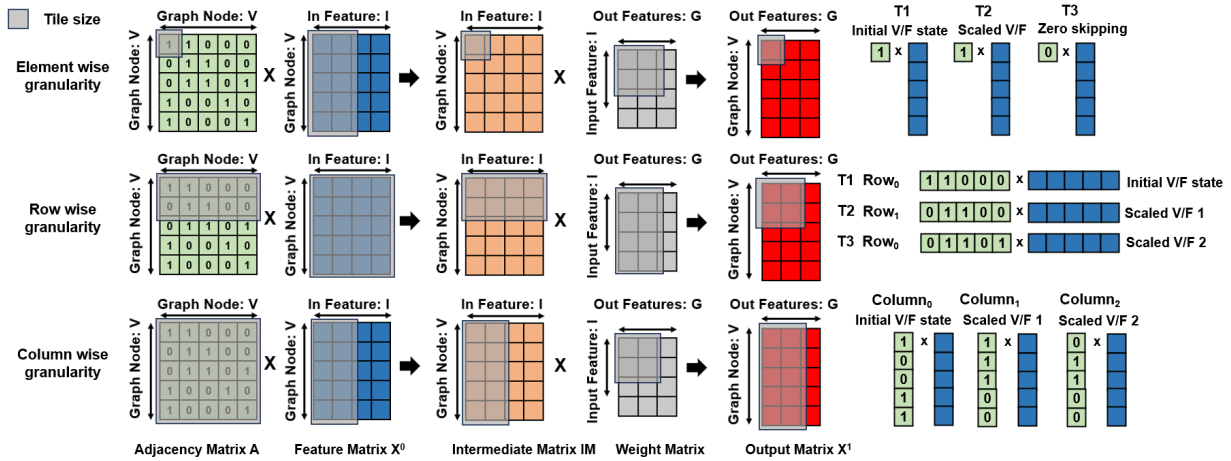

Fig. 6. An example to explain the customized SDMM with DVFS controlling in element-wise, row-wise, and column-wise granularities.

TABLE II
SUMMARY OF THE GRAPH DATASET STATISTICS.

Dataset	Nodes	Edges	Features	Classes	Storage
Cora	2,708	5,429	1,433	7	15 MB
Citesser	3,312	4,372	3,703	6	47 MB
Pubmed	19,717	44,338	500	3	38 MB
Nell	65,755	266,144	5,414	210	1.3 GB
Ogbn-ArXiv	169,343	1,166,243	128	40	103 MB
Reddit	232,965	114,615,892	602	41	1.8 GB

TABLE III
SUMMARY OF THE GCN MODEL SPECIFICATIONS.

Model	Hidden Dim.	Aggregation	Major Features
GCN	2	Mean Aggregator	/
GIN	3	Addition	All hidden layers concatenated
GraphSAGE	2	Generalized Aggregator	Subsampled neighbors as GCN

$$\mathrm{ECF} = K_{\text{packaging}} + \sum_{r}^{\text{PE, Memory, NoC, Storage}} \mathrm{EC}_r \tag{5}$$

where r represents the different components, including processing elements, memory modules, and storage devices, and $K_{\text{packaging}}$ represents the packaging footprint, which is typically estimated based on industry standards (e.g., 0.15 kg CO2 per IC).

V. PERFORMANCE EVALUATION

A. Simulation Environment

Graph-based Datasets and Algorithms: We evaluate GreeNX on three representative GCN algorithms, including GCN [29], GraphSAGE [35], and GIN [36], and six graph datasets, i.e., Cora, Citeceer, Ogbn-ArXiv, Reddit, Pubmed, and Nell. The statistics for the targeted datasets and GCN algorithms are summarized in Table II and III. These datasets cover a wide range of graph-structure data tensors.

Hardware Evaluation: We use Design Compile Ultra from Synopsys [37] to compile and synthesize the RTL design to obtain the power consumption and latency of the hardware components. FreePDK45, a 45nm technology node design kit is used as the target library. We manually modify the voltage and frequency settings to obtain the hardware metrics under the predefined V/F pairs. We use Cacti [38] to estimate the area, power, and access latency of the on-chip buffers and FIFOs. Table V shows the area, static power, dynamic power, and propagation latency of each submodule of the GCN accelerator under a 1.2V/1.8ns setting for 45nm technology. To evaluate the overall energy consumption and execution latency of the benchmarks, we built a cycle-accurate simulator in Python. The simulator models the microarchitectural design of each module, counts the exact number of operations, and estimates the energy consumption and latency.

State-of-the-Art GCN Accelerators: To compare our proposed GreeNX with state-of-the-art (SOTA) GCN acceleration, we consider a total of eight baselines: PyTorch Geometric (PyG) [39] with Intel Xeon E5-2680 v3 CPUs and NVIDIA RTX 8000 GPUs, Deep Graph Library (DGL) [40] with NVIDIA RTX 8000 GPUs, HyGCN [15], AWB-GCN [41], GCoD [17], GRIP [18], IGCN [42], and LW-GCN [19]. The system configurations of the baselines and our GreeNX are summarized in Table IV. We evaluate all the above platforms in terms of speedups, energy consumption, and required off-chip memory bandwidth and memory accesses. For example, HyGCN is an ASIC design that uses 4608 fixed-point MAC units running at 1GHz; AWB-GCN is an FPGA design that uses 4096 floating-point MAC units (running at 330MHz). To provide a fair comparison, GreeNX accelerator adopts 4096 PEs with a 32-bit fixed point precision.

B. Simulation Results

Speedups: We first evaluate GreeNX against both the general platforms in terms of speedup, off-chip memory bandwidth requirement, and the number of off-chip memory accesses. Fig.7 shows the overall performance of our GreeNX over the baselines. We can see thatGreeNX on average achieves 8562x and 1107x speedups over PyG-CPU and DGL-GPU, respectively. The superior GreeNX improvements validate the effectiveness of the proposed innovations: (1) GreeNX's unifying SDMM largely alleviates the irregularity of the graph adjacency matrices, leading to more consecutive accesses of off-chip memory; and (2) GreeNX accelerator utilizes DVFS

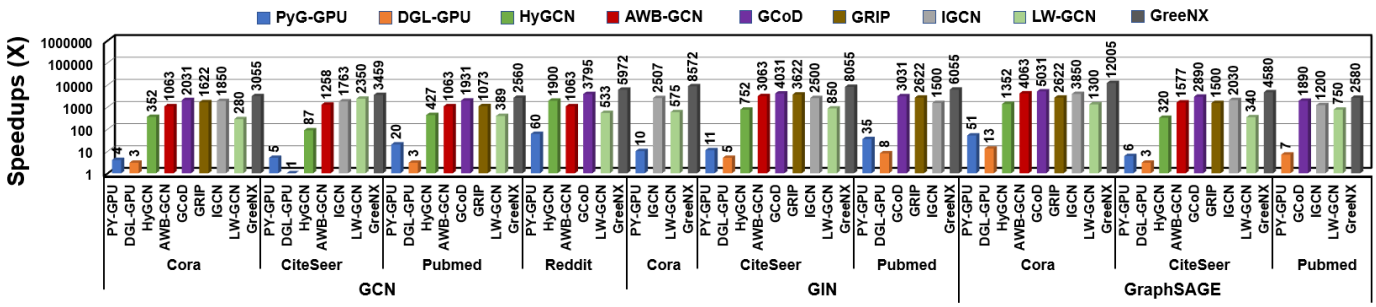

Fig. 7. The normalized inference speedups (w.r.t. PyG-CPU) achieved by our GreeNX framework over eight SOTA baselines on three variants of GCN models and representative citation graph datasets.

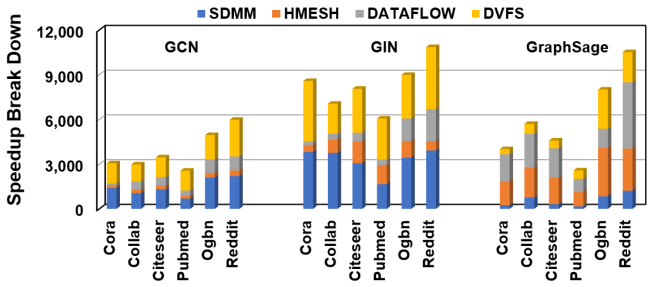

Fig. 8. Speedup breakdown of GreeNX accelerator

TABLE IV
HARDWARE PARAMETER SET UP AND PERFORMANCE COMPARISON WITH CPU, GPU, AND STATE-OF-THE-ART ACCELERATORS.

Design	Compute Unit	Off-chip Memory	Die Area (mm^2)	Power (W)
PyGCPU	2.5GHz@24 cores	1365 GB/s DDR4	-	150
DGLGPU	1.35GHz@4352 cores	616 GB/s GDDR6	754 (12nm)	250
HyGCN	1GHz@32 SIMD 8 systolic array	256 GB/s HBM 1.0	7.8 (12nm)	6.7
AWB-GCN	330MHz@Intel D5005 FPGA	76.8 GB/s DDR4	-	215
GoD	330MHz VCU128	460 GB/s HBM	-	180
GRIP	1GHz 4096 PEs	128 GB/s HBM	6.51 (45nm)	0.365
IGCN	4096MACs@330MHz Intel Stratix 10SX	76.8 GB/s DDR4	-	13.5
GreeNX	1GHz 4096 PEs	64 GB/s DDR4	4.55 (45nm)	0.258

to enable more balanced workloads while increasing the utilization of the PE array by allowing more data reuse along with lower on-chip storage.

We further compare GreeNX with SOTA GCN accelerators. As shown in Fig.7, GreeNX on-average achieves 9.2x, 3.4x, 1.3x, 2.1x, 2.3, and 11.8x speedups over HyGCN, AWB-GCN, GCoD, GRIP, IGCN, and LW-GCN (light version) respectively. GreeNX benefits can be attributed to the three optimization schemes. Specifically, HyGCN adopts fine-grained block-wise scheduling while GreeNX adopts coarse-grained DVFS and adaptive row-wise SDMM; AWB-GCN realizes workload balance via on-the-fly autotuning while GreeNX leverages DVFS to achieve a naturally balanced workload. We further provide the improvement breakdown in Fig.8, from which we can see that the improvement is mostly attributed to GreeNX's DVFS and row-wise SDMM enabled accelerator that contributes to on-average 75.6% speedups, while multicast optimized network further provides 14.3% speedups. The coarse-grain DVFS scheme in GreeNX efficiently processes sparse workload by scaling or shutting down the supply voltage and frequency of each PE to save energy. The uniform control avoids the workload imbalance and irregularity problems that are incurred in the two accelerator architectures.

As for the speedup for specific datasets and GCN algorithms, GreeNX performs on average 1.5x better and 2.7x better on GIN over GCN and GraphSAGE. This is because the aggregation computation requires a different range of neighborhoods to be aggregated in GCN and GraphSAGE. GreeNX benefits from the unified SDMMs for the two computation phases, thereby performing better in the average aggregation of GraphSAGE compared to the normalized aggregation of GCN. Datasets also have an impact on the overall speedup of GreeNX against other GCN accelerators. For example, GreeNX achieves a speedup of 12005x on Cora but only 2580x on Pubmed for GraphSAGE since the execution order exploration only reduces memory access by 1.4x on Pubmed dataset. Besides, the density of the feature matrix in Pubmed is higher than that of other datasets, which impedes the performance of DVFS in GreeNX because most of the working epochs are active in high voltage settings. For other accelerators exploiting sparsity to increase energy efficiency, the high density of the input feature matrix in Pubmed also hinders the speedup (such as GRIP and IGCN). Only DGL-GPU can maintain the speedup in Pubmed due to a large number of parallel GPU cores, though the overall speedups are limited on other benchmarks.

Memory Bandwidth Consumption and Accesses: Fig.9 shows the evaluation results in terms of off-chip memory bandwidth consumption. We can see that GreeNX only requires on-average 18% and 67% off-chip memory bandwidth compared to AWB-GCN, GCoD, respectively. The high bandwidth of AWB-GCN is attributed to the required high-degree parallelism, whereas GreeNX accelerator's row-wise SDMM and multicast optimized H-mesh NoC enable more frequent data reuses, largely alleviating the off-chip bandwidth requirement. Fig.9(b) reports the measured off-chip memory accesses comparison for processing GCNs with GreeNX, GCoD, and AWB-GCN. The DRAM access reduction varies across the datasets

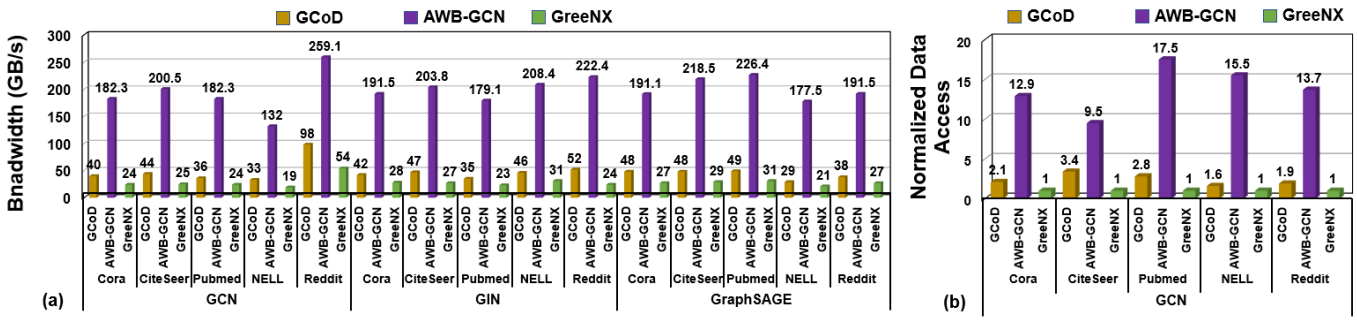

Fig. 9. (a) Bandwidth requirement of GreeNX, GCoD, and AWB-GCN and (b) normalized data access of GreeNX, GCoD, and AWB-GCN. Note that we record the peak bandwidth in (a), so the relative scale of (a) and (b) are slightly different.

and GCN algorithms. Specifically, GreeNX reduces DRAM accesses by up to a factor of 2.7x over GCoD, and 14.6x over AWB-GCN. Since GCoD and AWB-GCN use inefficient GCN execution order, they involve more data movement from the global buffer and off-chip memory into the PE array. AWB-GCN optimizes the reuse of the intermediate matrix by buffering the data between the two engines. However, it sacrifices the input data reuse since it lacks dataflow management. Moreover, the network-on-chip is not carefully tailored in AWB-GCN and GCoD accelerator.

Energy consumption and breakdown. Fig.10 shows the normalized energy consumption of GreeNX over general computation platforms and GCN accelerators. GreeNX consumes only 0.04%, 20.5%, and 85.1% energy on average compared to PyG-GPU, AWB-GCN, and GCoD, respectively. GreeNX achieves the worst energy efficiency on GraphSAGE model, which is also the case for most other comparators. The main reason is that GraphSAGE has additional computations and data accesses when performing aggregation, which cannot be optimized by hardware architectures. In terms of datasets, GreeNX achieves optimal energy efficiency in graphs/kJ on the Cora dataset. The underlying reason is that although the Cora dataset possesses the lowest sparsity ratio, it has the least number of graph dimensions, which requires less amount of computations and data accesses compared to other datasets. Fig.11 shows the energy breakdown with the contribution of computations, on-chip memory accesses, and the three proposed optimization components (i.e. SDMM, DVFS, and H-mesh NoC). We can see that memory accesses and computations consume most of the energy indicating the low overheads of the proposed techniques in GreeNX. Another observation is that on-chip memory access occupies the least energy on Reddit dataset for all the GCN models. This is because Reddit dataset offers the highest sparsity ratio (99.989%) and the most irregular data distribution among other datasets, verifying the effectiveness of GreeNX in exploiting sparsity and alleviating irregularity through SDMM and DVFS. Energy-delay product is used to verify that the optimization does not achieve high energy efficiency by sacrificing processing parallelism. The delay is calculated as the reciprocal of the number of execution cycles.

Impact of DVFS: To evaluate the performance of the proposed DVFS scheme of GreeNX, we compare the baseline configuration with the three DVFS models as depicted in Table I. In

TABLE V
HARDWARE CHARACTERISTICS OF GREENX.

	Area(mm^2)	%	Power(mW)	%
Total	4.55	100.00%	258.0	100.00%
PE Array	0.46	7.13%	65.4	23.62%
Sparsity Detector	0.14	2.36%	5.7	3.14%
Dynamic V/F Generator	0.07	1.07%	5.3	2.95%
DenseRow Prefetcher	0.12	1.85%	7.6	4.04%
Data Dispatcher	0.51	6.25%	19.4	9.58%
Network Router	0.03	0.48%	2.1	1.88%
SparseMat Memory	2.63	40.51%	78.5	36.72%
DenseMat Memory	0.75	13.28%	41.3	28.94%

Fig.12, energy consumption, execution latency, and the energy-delay-product (EDP) are presented for various benchmarks. All numbers are normalized to the baseline configuration that operates at the highest voltage. The coarse-grain DVFS scheme benefits most in terms of energy consumption by trading off latency for most benchmarks. We use EDP to verify that the DVFS scheme does not achieve high energy efficiency by simply sacrificing parallelism. Cora dataset achieves the best performance in the power-saving DVFS model with 59.3% energy consumption reduction, 17.8% latency increase, and 49.2% EDP reduction. However, we observed that the Nell dataset only obtains subtle energy reduction and even increased EDP results. This is because the Nell dataset demonstrates extremely sparse connections in the graph (0.0073% data density). In most working epochs, the Sparsity Detector outputs a power-gated state, which drives the DVFS generator to shut down the voltage supply for the PE array frequently. The accumulated workloads get congested in these inactive epochs, resulting in excessive execution latency.

Power and Area: Table V illustrates the hardware resources in GreeNX with 4K PEs. GreeNX consumes 0.258 Watts of total power while occupying 4.55 mm^2. The dynamic V/F controlling logic accounts for 9.68% of the entire accelerator (including Sparsity Detector, Dynamic V/F Generator, and Data Dispatcher).

Carbon Performance Evaluation: Utilizing the carbon emission as discussed in Section IV, we assessed the impact of our four proposed DVFS modes, as detailed in Table I, with the results illustrated in Figure 13. We compared the baseline GreeNX design, which operates at a fixed voltage and frequency without DVFS optimization, against three proposed DVFS implementations: GreeNX_PS, GreeNX_BA, and

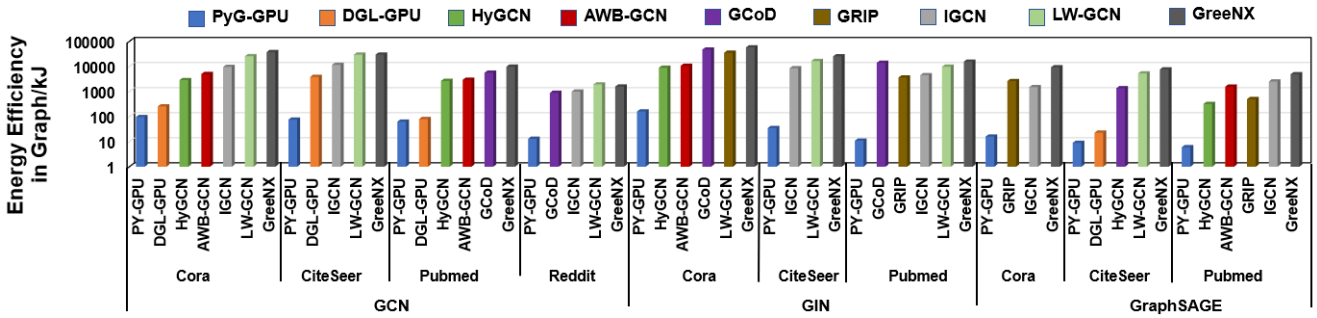

Fig. 10. Energy Efficiency in Graph/kJ achieved by our GreeNX framework over eight SOTA baselines on three variants of GCN models and representative citation graph datasets.

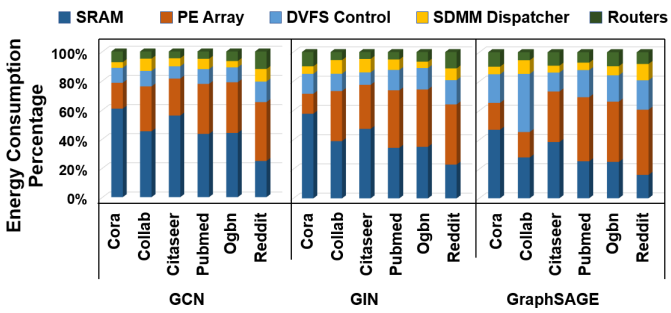

Fig. 11. The energy breakdown of the GreeNX framework when evaluated on the six datasets and the three GCN models.

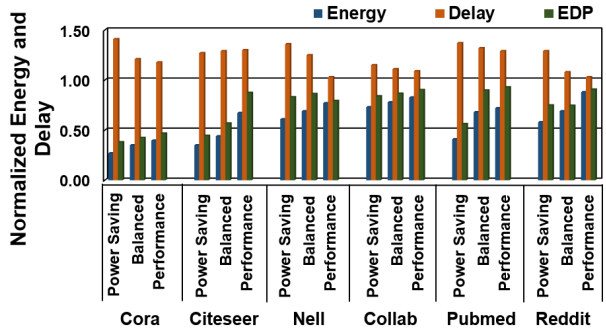

Fig. 12. EDP results for coarse-grained DVFS scheme on various datasets.

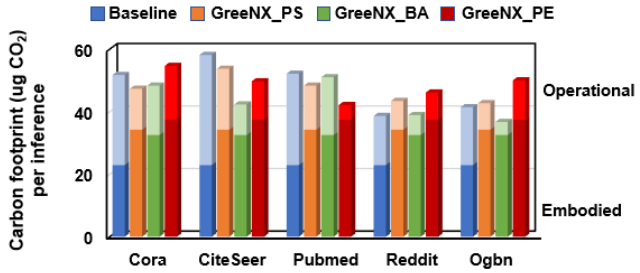

Fig. 13. Carbon footprint per inference for different datasets under various configurations: Baseline, GreeNX under Power_saving, Balance, and Performance DVFS modes.

GreeNX_PE.

As shown in Figure 13, the carbon footprint per inference varies across different datasets (Cora, CiteSeer, Pubmed, and Reddit) due to the distinct strategies employed by each DVFS mode. The figure differentiates between embodied and operational carbon contributions, highlighting that while all DVFS modes achieve significant reductions in operational energy consumption, this does not uniformly translate to lower carbon emissions. This discrepancy arises because the DVFS modes introduce additional overheads, reflected in the embodied carbon costs. By implementing carbon-aware control during runtime, the DVFS controller can be optimized to select the mode that minimizes overall carbon emissions, thereby enhancing environmental efficiency.

VI. Related Work

Graph Inference Accelerators. The ultra sparsity in real-world graphs (e.g., Nell dataset has 99.988% sparsity vs. sparse DNNs have 88.9% 92.3% sparsity [43]) imposes challenges on the dynamic and irregular data accesses for GCNs' feature aggregation, which has a different execution pattern with DNN accelerators, leading to dedicated GCNs. GraphABCD [44] is an asynchronous heterogeneous graph analytic framework that offers algorithm and architectural supports for asynchronous execution, without undermining its fast convergence properties. Recently, RAHP [45] introduces a redundancy-aware approach specifically for HyperGNN inference, optimizing computation and memory access patterns to significantly reduce redundant calculations. Furthermore, NeuraChip [14] employs a hash-based decoupled spatial accelerator design that effectively balances load and optimizes data caching strategies to improve performance and energy efficiency in GNN computations. Although these accelerators deliver considerable performance and energy efficiency improvement, they are inefficient when handling GCNs because even though they are designed to alleviate the irregularity of graph data, they do not leverage the regularity in GCNs.

Neural Network Accelerators. Numerous research efforts have been dedicated to enhancing the speed of neural networks, as demonstrated by works such as [46]–[52]. In the case of dense neural networks, the focus of accelerators is primarily on utilizing vast parallelism to enhance performance and utilization. For instance, TPU [47] and Eyeriss [46] are two examples of such accelerators. To improve the hardware implementation efficiency of compressed DNN models, accelerators have been developed to exploit the inherent sparsity of neural networks, thereby reducing the number of operations required [48]–[52]. However, GCNs contain two-phase matrix

multiplications that enable new kinds of parallelisms and data reuse patterns that are not exploited in these neural network accelerators. Although we can extend CNN accelerators to run SpMMs by equalizing the input and filter dimensions, it weakens the advantages of CNN accelerators since they are specialized for convolutions rather than matrix multiplications.

VII. Conclusions

In this paper, we introduce GreeNX, an energy-efficient GCN accelerator that optimizes the use of Dynamic Voltage and Frequency Scaling (DVFS) and Sparse-Dense Matrix Multiplication (SDMM) within a hierarchical mesh network to enhance throughput and energy efficiency. GreeNX capitalizes on sparse graph connections with a novel row-wise SDMM approach that improves data reuse and enables high parallelism, addressing workload imbalances by dynamically adjusting processing elements' operating voltages in real-time. Despite these advancements, the reliance on DVFS could introduce complexity in real-time operational tuning and may limit scalability under varying workload conditions. Future research will focus on refining DVFS algorithms to enhance adaptability and exploring advanced machine learning techniques for predictive workload management. A comprehensive carbon footprint analysis—covering both embodied and operational emissions—underscores the environmental benefits of our strategies, positioning GreeNX as a sustainable solution in large-scale deployments. Performance tests on six real-world graph datasets show that GreeNX achieves significant gains, with a $7.3\times$ speedup and $5.6\times$ energy savings over contemporary state-of-the-art accelerators such as HyGCN, AWB-GCN, GCoD, GRIP, IGCN, and LW-GCN.

Acknowledgments

This research was partially supported by NSF grants CCF-1703013, CCF-1901192, CCF-1936794, CCF-2324645, and CCF-2311544.

References

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[2] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: a comprehensive graph neural network platform," *arXiv preprint arXiv:1902.08730*, 2019.

[3] L. Wu, P. Sun, R. Hong, Y. Fu, X. Wang, and M. Wang, "Socialgcn: An efficient graph convolutional network based model for social recommendation," *arXiv preprint arXiv:1811.02815*, 2018.

[4] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[5] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 922–929.

[6] H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo, "Knowledge graph convolutional networks for recommender systems," in *The world wide web conference*, 2019, pp. 3307–3313.

[7] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International conference on machine learning*. PMLR, 2020, pp. 8459–8468.

[8] Y. Wang, S. Qian, J. Hu, Q. Fang, and C. Xu, "Fake news detection via knowledge-driven multimodal graph convolutional networks," in *Proceedings of the 2020 international conference on multimedia retrieval*, 2020, pp. 540–547.

[9] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang, "Graph neural networks and their current applications in bioinformatics," *Frontiers in genetics*, vol. 12, p. 690049, 2021.

[10] M. Yan, Z. Chen, L. Deng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Characterizing and understanding gcns on gpu," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 22–25, 2020.

[11] Z. Zhang, J. Leng, L. Ma, Y. Miao, C. Li, and M. Guo, "Architectural implications of graph neural networks," *IEEE Computer architecture letters*, vol. 19, no. 1, pp. 59–62, 2020.

[12] Y. Li, T.-Y. Yang, M.-C. Yang, Z. Shen, and B. Li, "Celeritas: Out-of-core based unsupervised graph neural network via cross-layer computing 2024," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 91–107.

[13] D. Gurevin, M. Shan, S. Huang, M. A. Hasan, C. Ding, and O. Khan, "Prunegnn: Algorithm-architecture pruning framework for graph neural network acceleration," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 108–123.

[14] K. Shivdikar, N. B. Agostini, M. Jayaweera, G. Jonatan, J. L. Abellán, A. Joshi, J. Kim, and D. Kaeli, "Neurachip: Accelerating gnn computations with a hash-based decoupled spatial accelerator," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 946–960.

[15] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 15–29.

[16] J. Li, A. Louri, A. Karanth, and R. Bunescu, "Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 775–788.

[17] H. You, T. Geng, Y. Zhang, A. Li, and Y. Lin, "Gcod: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 460–474.

[18] K. Kiningham, C. Re, and P. Levis, "Grip: A graph neural network accelerator architecture," *arXiv preprint arXiv:2007.13828*, 2020.

[19] Z. Tao, C. Wu, Y. Liang, and L. He, "Lw-gcn: A lightweight fpga-based graph convolutional network accelerator," *arXiv preprint arXiv:2111.03184*, 2021.

[20] F. Shi, A. Y. Jin, and S.-C. Zhu, "Versagnn: a versatile accelerator for graph neural networks," *arXiv preprint arXiv:2105.01280*, 2021.

[21] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2496–2508, 2022.

[22] S. Jain, L. Lin, and M. Alioto, "Automated design of reconfigurable microarchitectures for accelerators under wide-voltage scaling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 3, pp. 777–790, 2020.

[23] S. Song, X. Zheng, A. Gerstlauer, and L. K. John, "Fine-grained power analysis of emerging graph processing workloads for cloud operations management," in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 2121–2126.

[24] S. Liu and A. Karanth, "Dynamic voltage and frequency scaling to improve energy-efficiency of hardware accelerators," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2021, pp. 232–241.

[25] X. Hou, P. Tang, T. Xu, C. Xu, C. Li, and M. Guo, "Cpm: A cross-layer power management facility to enable qos-aware aiot systems," in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*, 2024, pp. 1–10.

[26] C. Tan, M. Jiang, D. Patil, Y. Ou, Z. Li, L. Ju, T. Mitra, H. Park, A. Tumeo, and J. Zhang, "Iced: An integrated cgra framework enabling dvfs-aware acceleration," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1338–1352.

[27] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 261–274.

[28] G. Huang, G. Dai, Y. Wang, and H. Yang, "Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–12.

[29] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[30] G. Santoro, M. R. Casu, V. Peluso, A. Calimera, and M. Alioto, "Design-space exploration of pareto-optimal architectures for deep learning with dvfs," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.
[31] W. Jiang, H. Yu, J. Zhang, J. Wu, S. Luo, and Y. Ha, "Optimizing energy efficiency of cnn-based object detection with dynamic voltage and frequency scaling," *Journal of Semiconductors*, vol. 41, no. 2, p. 022406, 2020.
[32] Q. Wang and X. Chu, "Gpgpu performance estimation with core and memory frequency scaling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2865–2881, 2020.
[33] Z. Yu, P. Machado, A. Zahid, A. M. Abdulghani, K. Dashtipour, H. Heidari, M. A. Imran, and Q. H. Abbasi, "Energy and performance trade-off optimization in heterogeneous computing via reinforcement learning," *Electronics*, vol. 9, no. 11, p. 1812, 2020.
[34] M. Clark, Y. Chen, A. Karanth, B. Ma, and A. Louri, "Dozznoc: Reducing static and dynamic energy in nocs with low-latency voltage regulators using machine learning," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 1–11.
[35] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
[36] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
[37] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 2012.
[38] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to understand large caches," *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, vol. 147, 2009.
[39] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.
[40] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.
[41] 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.
[42] 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.
[43] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.
[44] Y. Yang, Z. Li, Y. Deng, Z. Liu, S. Yin, S. Wei, and L. Liu, "Graphabcd: Scaling out graph analytics with asynchronous block coordinate descent," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 419–432.
[45] H. Yu, Y. Zhang, L. He, Y. Zhao, X. Li, R. Xin, J. Zhao, X. Liao, H. Liu, B. He *et al.*, "Rahp: A redundancy-aware accelerator for high-performance hypergraph neural network," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1264–1277.
[46] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.
[47] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.
[48] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.
[49] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.
[50] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 45, no. 2, pp. 27–40, 2017.
[51] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 151–165.
[52] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

Siqin Liu (S'19) received the B.S. degree in optical information science and technology from the Wuhan University, Wuhan, China in 2011. After graduation, he joined Lingcom Electronics Ltd., a digital signal processing platform provider, and was employed as an electronic hardware engineer for 8 years. He is currently a Ph.D. student in the Electrical Engineering and Computer Science department at Ohio University, Athens, OH, USA. His current research interests include computer architecture, Deep Neural Networks hardware accelerators, and network-on-chips (NoCs).

Prakash Kuve received his B. Tech degree in Electronics and Communication Engineering from Manipal Institute of Technology, Manipal, India and a Master's Equivalent degree in Micro-Electronics from KarMic Training Center, Manipal in 1999 and 2000 respectively. He is currently working with the Timing and Communication Group (TCG) at Microchip Technology Corporation. His work interests include Precision Data Converters, low-jitter PLL circuits, and Power management circuits.

Avinash Karanth (SM'12) received the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Arizona, Tucson, AZ, USA, in 2003 and 2006, respectively. He is currently the Chair of the School of Electrical Engineering and Computer Science in Ohio University, Athens, OH, USA. His current research interests include computer architecture, optical interconnects, chip multiprocessors (CMPs), and network-on-chips (NoCs). He was a recipient of the National Science Foundation CAREER Award in 2011, the Best Paper Award at the ICCD 2013 conference and his papers have been nominated for best paper at the HiPC-2021, DATE-2019, NoCs-2010 and ASP-DAC-2009. He is the Editor-in-Chief for IEEE Transactions on Computers and Associate Editor for IEEE Transactions on Cloud Computing journals. He is a senior member of IEEE and member of ACM.