Exploring Architecture, Dataflow, and Sparsity for GCN Accelerators: A Holistic Framework

Lingxiang Yin
lingxiangyin@knights.ucf.edu
University of Central Florida
Orlando, FL, USA

Jun Wang
jun.wang@ucf.edu
University of Central Florida
Orlando, FL, USA

Hao Zheng
hao.zheng@ucf.edu
University of Central Florida
Orlando, FL, USA

ABSTRACT

Recent years have seen an increasing number of Graph Convolutional Network (GCN) models employed in various real-world applications. However, designing efficient architectures for GCN acceleration remains challenging due to the varied sparsity across graph datasets. Despite significant efforts, very few of the existing works have considered a holistic view of the entire GCN accelerator design, and therefore, the dynamic interactions between architecture, dataflow (i.e., data reuse and parallelization strategies), and compression format are not well studied.

In this paper, we endeavor to develop a holistic framework supporting the rapid exploration of GCN design space - synergizing architecture, dataflow, and sparsity optimizations. Specifically, we implement a variety of compression formats tailored for handling extreme and irregular sparsity. Moreover, we propose a generic GCN architecture capable of supporting various dataflow and compression formats in one combined architecture. Given the exploded GCN design space, a genetic algorithm is implemented to facilitate rapid exploration while suggesting an optimal GCN solution with suitable dataflow and compression formats. Our proposed framework can achieve 12.3×, 2.2×, and 1.37× speedup and 15.3×, 3.7×, and 1.6× energy efficiency on average as compared to HyGCN [1], AWB-GCN [2], and GCNAX [3], respectively.

CCS CONCEPTS

• **Hardware → Electronic design automation**; **Hardware accelerators**.

KEYWORDS

GCN, Accelerator, Dataflow, Sparsity, Design Space Exploration

ACM Reference Format:
Lingxiang Yin, Jun Wang, and Hao Zheng. 2023. Exploring Architecture, Dataflow, and Sparsity for GCN Accelerators: A Holistic Framework. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3583781.3590243

1 INTRODUCTION

Graph Convolutional Networks (GCNs) [4–7] are being deployed in a wide range of application domains [8, 9] and pervading many aspects of our lives. GCNs are considered a variant of traditional convolutional neural networks (CNNs) specialized for graph-structured data, where the data structure is sparse and irregular. Unlike CNN, the irregular sparsity in GCN mainly stems from the connectivity of vertices and edges, and thus it can barely be remedied via existing pruning algorithms [10–13] with even-distributed patterns while respecting the model accuracy. These intrinsic data characteristics, accompanied by the complexity inherited from CNNs, are posing a major challenge to the underlying hardware designs.

Significant research efforts [1–3] have been devoted to addressing the design challenges in the presence of GCNs, facilitating their primitive operations - chain Sparse-Dense Matrix Multiplication (SpMM) [2]. For example, in HyGCN [1], two dedicated compute engines were designed for distinct GCN computation characteristics in Aggregation and Combination phases. AWB-GCN [2] architecture addressed the workload imbalance issue that emerged from the irregular sparsity. GCNAX [3] explored a flexible dataflow with the ability to alter loop order and fusion, thus reducing off-chip memory access and improving overall performance. Moreover, prior work [14] has explored the architecture design to enable dynamic transformation between various compression formats, and it can select suitable compression formats for aggregation and combination phases with varied matrix sparsity. However, relatively few efforts have carefully examined the combined effects of various design choices in GCN optimizations.

Though design space exploration has been studied in conventional neural networks [15, 16], it has limited applicability for GCN accelerators in the presence of extreme sparsity ($\geq$ 99.9%). The sparsity in GCNs relies on compression techniques to reduce unnecessary data movement (i.e., zero elements), whereas CNN accelerators are often optimized with various dataflows to exploit data reuse opportunities [17]. Unfortunately, these techniques may conflict with the desired objective, degrading the overall system performance and energy efficiency. For example, Compressed Sparse Row (CSR) retains the non-zero elements of a sparse matrix while promising fast row access. However, CSR could be a major impediment to efficient outer-product SpMM where the sparse matrix is retrieved from column to column. The inconsistent access patterns of CSR and outer-product SpMM could lead to undesirable performance loss. In addition, by offering various compression options, it is possible to select the most effective compression format for a given GCN dataset. However, each compression format requires distinct processes to accomplish its coordinate reconstruction for SpMM, and its impact on the dataflow and architecture design has not been well studied.

To this end, we endeavor to develop a holistic framework in support of the design space exploration for GCN accelerators, spanning architecture, dataflow, and sparsity. Specifically, we formulate a variety of compression formats and their decompression overhead in the presence of extreme and irregular sparsity. Moreover, we propose a generic GCN architecture capable of supporting various dataflow and compression formats in one combined architecture. Furthermore, we systematically incorporate a set of design choices devised for GCN accelerators, including processing unit number, buffer size, loop optimization, parallelization strategies, compression formats, and decoding latency. To facilitate the rapid exploration of the GCN design space, we formulate it as an optimization problem using a genetic algorithm.

The major contributions of this paper are as follows:

- **Compression Formats in GCN:** We formulate the compression ratio and decompression latency for four generic compression formats in compliance with GCN computation, namely Compressed Sparse Row/Column (CSR/CSC), Coordinate List (COO), and Zero-Value Compression (ZVC) [18].
- **A holistic Design Space Exploration (DSE):** We integrate a collection of system parameters into a holistic design space exploration framework. The proposed framework uncovers the dynamic interactions between architecture (e.g., PE number and buffer size), dataflow (e.g., loop order and parallelization strategies), and sparsity (e.g., compression ratio and decompression latency).
- **Genetic Algorithm (GA)-based Search:** We formulate GCN DSE as an optimization problem, and the proposed GA-based search could suggest viable GCN accelerator designs that can achieve 12.3×, 2.2× and 1.37× speedup and 15.3×, 3.7×, and 1.6× energy efficiency on average as compared to HyGCN [1], AWB-GCN [2], and GCNAX [3], respectively.

2 BACKGROUND AND MOTIVATION

2.1 Graph Convolutional Networks

Graph Convolutional Network (GCN) is a variant of traditional CNNs but specific to graph-structured data. GCNs aggregate information from graph structure and learn features from neighboring nodes. Similar to CNNs, GCN is composed of multiple layers, where each layer performs an element-wise multiplication. As shown in Figure 1, the computation in a GCN layer can be formulated as:

$$a_v^{(k)} = Aggregate^{(k)}(h_u^{(k)} | u \in \mathcal{N}(v)), \\ h_v^{(k)} = Combine^{(k)}(a_v^{(k)}, h_v^{(k-1)}) \tag{1}$$

where the *Aggregate* function aggregates multiple feature vectors from neighbors to one single feature vector $a_v^{(k)}$, and the *Combine* function transforms the feature vector, $h_v^{(k-1)}$, of each vertex to another feature vector using a multi-layer perceptron neural network. This turns out that the prevalent computation pattern of GCN can be considered as a chain matrix multiplication:

$$X^{(k+1)} = \sigma(\hat{A}X^{(k)}W^{(k)}) \tag{2}$$

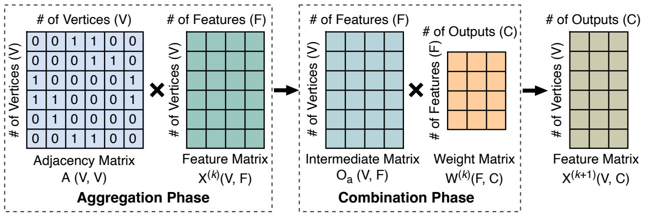


Figure 1: An example of Chain Matrix Multiplication ($(A \times X) \times W$) performed in a Graph Convolutional Network Layer.

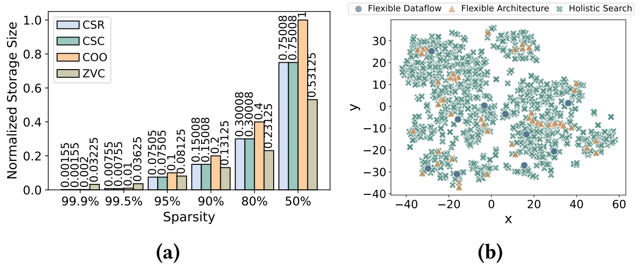


Figure 2: (a) Normalized storage size for different compression formats with varied sparsity, and (b) tSNE of 1000 randomly selected design choices for dataset Reddit.

where $X^{(k)}$ is the matrix of input features in layer k; each column of X represents a feature vector while each row denotes a node. $W^{(k)}$ is the weight matrix of layer k. $\sigma(\cdot)$ denotes the non-linear activation function such as ReLU. $\hat{A}$ is a transformed matrix from the graph adjacency matrix, and the transformation function varies across different GCN models.

The order of the chain matrix multiplication determines the types of matrix multiplication with different sparsity, which varies in the *Aggregation* phase and *Combination* phase. In the *Aggregation* phase, the computation can be generalized as sparse-sparse matrix multiplication (SpGEMM) or sparse-dense matrix multiplication (SpMM) operations, whereas the computations of the *Combination* phase are considered as SpMM or dense matrix multiplication (DenseMM) operations. Despite these differences, the computation order of $A \times (X \times W)$ has been widely considered for GCN computation, as it can unify the computation in both *Aggregation* and *Combination* phases as SpMM. In this paper, we will also consider the unified SpMM for both aggregation and combination phases.

2.2 Motivation

We conducted a preliminary study of various compression formats and their performance in reducing storage size across different sparsity ratios. As shown in Figure 2 (a), CSR, CSC, and COO have better performance in reducing storage size in matrices with higher sparsity ratio, whereas ZVC performs better in those matrices of lower sparsity. Even though different compression formats have shown distinct performance in reducing storage size, we will analyze their interactions with different dataflows in what follows.

As mentioned, while significant research efforts have been devoted to optimizing GCN performance, they are a few isolated points in the complex GCN design space. We used a tSNE (t-distributed

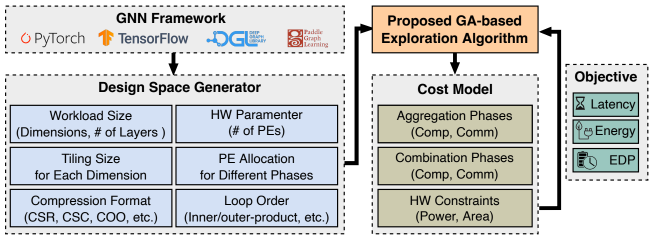

GNN Framework
PyTorch
TensorFlow
DGL
Paddle Graph Learning
Design Space Generator
Workload Size (Dimensions, # of Layers)
HW Paramenter (# of PEs)
Tiling Size for Each Dimension
PE Allocation for Different Phases
Compression Format (CSR, CSC, COO, etc.)
Loop Order (Inner/outer-product, etc.)
Proposed GA-based Exploration Algorithm
Cost Model
Aggregation Phases (Comp, Comm)
Combination Phases (Comp, Comm)
HW Constraints (Power, Area)
Objective
Latency
Energy
EDP

Figure 3: Proposed holistic DSE framework for GCN Designs.

stochastic neighbor embedding) [19] to visualize complex, nonlinear relationships between various GCN designs. We observed that a holistic design exploration could yield a more diversified design space, which could turn into improved performance.

3 PROPOSED FRAMEWORK FOR GCN DSE

The aim of this work is to capture the dynamic interactions between a wide range of optimization techniques being used for GCN designs as shown in Figure 3. The proposed framework is capable of exploring a collection of system parameters related to architecture (e.g., area, buffer, and processing elements), dataflow (e.g., parallelization and data reuse), and sparsity (e.g., compression formats). In addition, we implement a customized genetic algorithm, together with a generic GCN architecture, in support of various optimization objectives such as latency, energy, and energy-delay product (EDP).

3.1 Generic GCN Architecture

A number of GCN accelerators have been proposed to expedite SpMM within both aggregation and combination phases, albeit in different forms. To exploit design space, in this work, we consider a generic GCN architecture capable of supporting common SpMM computation characteristics and compression formats. As illustrated in Figure 4, the generic GCN architecture consists of DRAM, a global buffer, a task distributor, and an array of processing elements (PEs). The global buffer consists of a sparse buffer, decompression logic, and a dense buffer. The decompression logic can support the CSR, CSC, COO, and ZVC compression formats. This allows the accelerator to take full benefits of compression formats at their finest granularity with considerable storage savings. The task distributor delivers paired data to PE arrays as well as balances the workload [2], which can efficiently manage the unbalanced workload caused by the irregular sparsity. The PE array is designed to support SpMM, where PEs can be dynamically partitioned to support aggregation and combination phases. This allows the simultaneous execution of aggregation and combination phases in a pipeline fashion, which reduces unnecessary off-chip data movements. We note that the partition scheme will be managed by the proposed GA-based search framework. Each PE has a 16 MAC units array with 32-bit precision. We note that the buffer size, PE size, and compression format could be dynamically selected during the design space exploration.

3.2 Sparsity Optimization for GCNs

The sparse and unstructured connectivity of graphs causes extreme and irregular sparsity in the adjacency matrices. Significant research

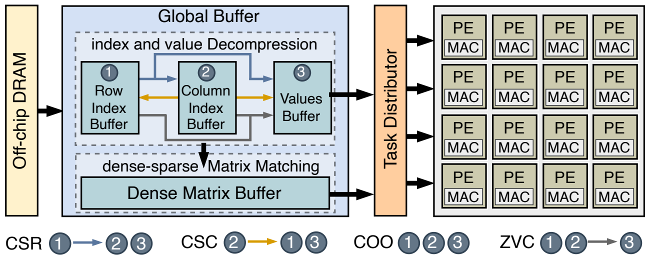


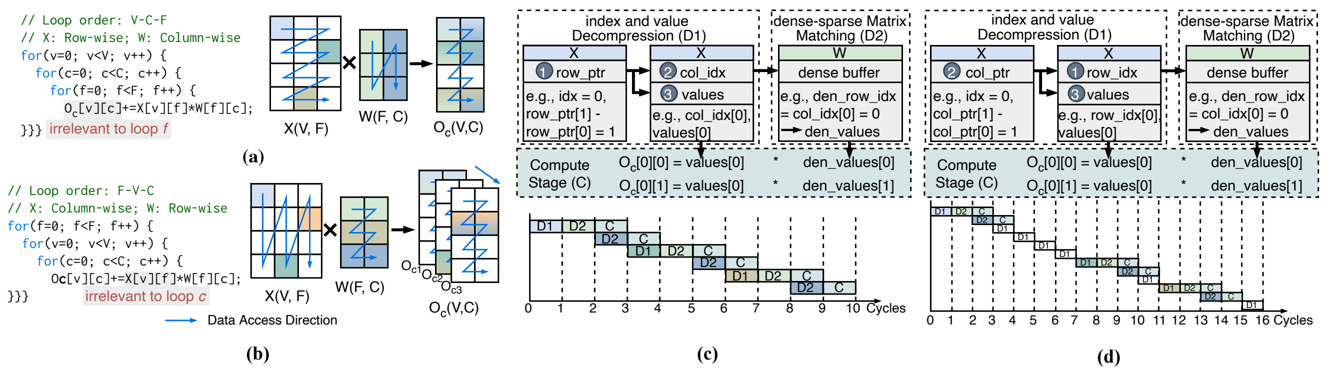
Figure 4: A generic GCN architecture consisting of DRAM, global buffer, task distributor, and processing elements.

efforts have been made to exploit various compression/decompression techniques from both algorithm [10–12] and hardware [20, 21] designs. In addition to prior efforts, we target the interaction between compression formats and existing GCN design choices, which is being neglected in the literature. In this work, we formulate the compression ratio and decompression latency of the four most commonly used compression formats, CSC, CSR, COO and ZVC, for generic GCN architectures. These compression formats are optimized for different memory access patterns and sparsity ratios due to their unique metadata formats.

3.2.1 Compression Formats in GCNs. The major idea of different compression formats is to eliminate zero elements from sparse matrices, reducing storage size. Despite much-reduced storage size, the compression formats could potentially jeopardize the data regularity and incur additional latency when reconstructing the original matrices [14]. For example. as shown in Figure 4, compression formats rely on row (①) and index (②) information to locate value (③), called index and value decompression. The value will be sent to PEs. In the meantime, the row and index will be sent to the dense matrix to find its corresponding elements, called dense-sparse matrix matching.

The position information of non-zero elements is used for indexing the dense matrix to extract the corresponding value at runtime. The data structure of position information determines the compression ratio (i.e., $\frac{Compressed\ storage\ size}{Original\ storage\ size}$) and latency for pairing non-zero elements called decompression latency. In what follows, we will analyze the compression ratio and decompression latency for different compression formats specialized for GCN designs. For simplicity, we take an X by Y matrix as an example, where X is the row size, Y is the column size, and d is the matrix density (i.e., $\frac{Number\ of\ Non-zero\ elements(NNZ)}{Total\ number\ of\ elements}$) of the matrix. We also assume the size of each non-zero element is E bits.

CSR/CSC stores non-zero values using column and row information, where three arrays are required. CSR needs to store $d \times X \times Y$ non-zero elements and their corresponding column indices. In addition, $X+1$ of row indices are needed. As a result, the total storage size is $NNZ \times E/8$ (for *values*) + $NNZ \times \lceil \lceil log_2^Y \rceil /8 \rceil$ (for *col_idx*) + $(X+1) \times \lceil \lfloor log_2^{NNZ} + 1 \rfloor /8 \rceil$ (for *row_ptr*) bytes. The decompression of CSR involves two stages - it accesses the row_ptr (①), and then col_idx and values (②③). We formulate the decompression latency of CSR as shown in Listing 1. CSC has a very similar data structure and performance as compared to CSR, but it is optimized for column access.



```
// Loop order: V-C-F
// X: Row-wise; W: Column-wise
for(v=0; v<V; v++) {
  for(c=0; c<C; c++) {
    for(f=0; f<F; f++) {
      Oc[v][c]+=X[v][f]*W[f][c];
}}} irrelevant to loop f
```

```
// Loop order: F-V-C
// X: Column-wise; W: Row-wise
for(f=0; f<F; f++) {
  for(v=0; v<V; v++) {
    for(c=0; c<C; c++) {
      Oc[v][c]+=X[v][f]*W[f][c];
}}}        irrelevant to loop c
```

X(V, F)
W(F, C)
O_c(V,C)
(a)
O_c1
O_c2
O_c3
Data Access Direction
(b)
index and value Decompression (D1)
dense-sparse Matrix Matching (D2)
X
1 row_ptr
2 col_idx
3 values
e.g., idx = 0, row_ptr[1] - row_ptr[0] = 1
e.g., col_idx[0], values[0]
W
dense buffer
e.g., den_row_idx = col_idx[0] = 0 → den_values
Compute Stage (C)
O_c[0][0] = values[0] * den_values[0]
O_c[0][1] = values[0] * den_values[1]
D1 D2 C
0 1 2 3 4 5 6 7 8 9 10 Cycles
(c)
2 col_ptr
1 row_idx
3 values
e.g., idx = 0, col_ptr[1] - col_ptr[0] = 1
e.g., row_idx[0], values[0]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Cycles
(d)

Figure 5: Dataflow examples for combination phase: (a) Inner-product, (b) Outer-product, (c) CSR for Inner-product, (d) CSC for Inner-product.

```
function decompress(cur_row, values, col_idx, row_ptr):
    # cur_row: current row index
    # nze: number of non-zero elements in cur_row
    nze = row_ptr[cur_row+1]-row_ptr[cur_row]
    for i in range(0, nze):
        de_row[cur_row][col_idx[i]] = values[i]
    return de_row
```

Listing 1: CSR decompression scheme pseudo code.

COO also stores non-zeros by utilizing the column and row information but in a different manner. The first array stores the non-zero elements, whereas the second and third arrays store the row and column indices for related values. The total storage size is $NNZ \times E/8$ (for *values*) + $NNZ \times \lceil\lceil log_2^X\rceil/8\rceil$ (for *row_idx*) + $NNZ \times \lceil\lceil log_2^Y\rceil/8\rceil$ (for *col_idx*) bytes. COO can decode the row indices, column indices, and values (①②③) at the same time, and we formulate its decompression latency as depicted in Listing 2.

```
function decompress(cur_row, values, row_idx, col_idx):
    # cur_row: current row index
    for i in range(0, len(row_idx)):
        if row_idx[i] == cur_row:
            de_row[cur_row][col_idx[i]] = values[i]
    return de_row
```

Listing 2: COO decompression scheme pseudo code.

ZVC utilizes a bitmap to represent the position information of non-zero elements. This leads to the total storage size of $NNZ\times E/8$ (for *values*) + $\lceil X \times Y/8\rceil$ (for *bitmap*) bytes. To decode a non-zero element, it has to traverse the bitmap (①②) to find the data (③) as shown in Listing 3.

```
function decompress(cur_row, values, bitmap):
    # cur_row: current row index
    for i in range(0, len(bitmap)):
        # COL: numbers of column
        # idx: index for values
        if floor(i/COL) == cur_row
            de_row[cur_row][i%COL] = values[idx]
            idx += 1
    return de_row
```

Listing 3: ZVC decompression scheme pseudo code.

The latency overheads can be overlapped in a pipeline manner, but its performance requires a careful selection of dataflows which will be discussed in the next section.

3.3 The interaction between dataflow and compression format

Common dataflow strategies, like loop order and tiling optimizations, can be effectively applied to facilitate the primitive GCN operation [3, 22–24]. However, given the unique characteristics of GCNs, the selection of dataflow, especially the loop order, should consider the deployment of compression techniques.

Loop interchange is the process of altering the order of a nested loop as shown in Figure 5 (a) and (b), where three loops, V, C, and F, could be interchanged. The interchanged loop order can impact the data movement and the type of matrix multiplication [25, 26]. For example, in Figure 5 (a), loop F is the innermost loop, where F iterates more frequently than the other two loops (i.e., V and C). Since matrix O_c is irrelevant to iterator F, the data of matrix O_c appears to be reused for multiple multiplications. Similarly, the matrix X is likely to be mostly reused in Figure 5 (b). However, this raises two potential issues for optimizing GCN accelerations.

First, the loop interchange decides the type of matrix multiplication, such as inner product and outer product. In the outer product as shown in Figure 5 (b), each column of the sparse matrix X will be multiplied by each row of the weight matrix W. The inner product performs in an opposite way as compared to the outer product, where each row of the sparse matrix X will be multiplied by each column of the weight matrix W. The difference in access patterns caused by loop interchange could conflict with the mentioned compression formats. This not only increases the latency overhead but also affects the storage efficiency. For example, three steps are required in GCN to perform one computation, namely index and value decompression (D1), dense sparse matrix matching (D2), and computation (C). As shown in Figure 5 (c), in an inner product SpMM, the sparse matrix (X(V,F)) can be decoded in a row-wise manner when CSR format is selected, and therefore the decoding latency can be overlapped. However, each row of the sparse matrix has to be decoded multiple times when CSC is applied, as its metadata only records the coordinate information of each column. As such, the decompression latency is hard to be overlapped as shown in Figure 5 (d).

Second, the interchange will also affect the data locality of sparse matrices and later influences the decompression latency overheads. The root cause of decompression latency is sparse matrices, in

Table 1: Row-column access patterns and data reuse of various loop orders for matrix X and W in combination phase.

Loop Order	X(V, F)		W(F, C)		Oc
	Row/Column Access Patterns	Data Reuse	Row/Column Access Patterns	Data Reuse	Data Reuse
V-F-C	Row	High	Row	Low	High
V-C-F	Row	High	Column	Low	High
F-V-C	Column	Low	Row	High	Low
F-C-V	Column	High	Row	Low	Low
C-V-F	Row	Low	Column	High	High
C-F-V	Column	Low	Column	High	Low

which non-zero elements have to pair with their associated elements for SpMM. Frequently iterating sparse matrices could affect the data locality of sparse matrices. This could lead to redundant decompression and data paring in SpMM. As such, it could be beneficial to retain the sparse matrices at PE while iterating the dense matrices. However, this may conflict with the objective of optimizing data movement. Consequently, the competing objectives of various design choices should be carefully evaluated. Table 1 summarizes the row-column access patterns and data reuse rate for various loop orders for both sparse (X(V,F)) and dense (W(F,C)) matrices in the GCN combination phase.

4 PROPOSED GENETIC ALGORITHM-BASED SEARCH MODEL

Given the large design space, we formulate the GCN design space search as an optimization problem in the form of a Genetic Algorithm (GA) [27]. GA is a search algorithm inspired by the phenomena of evolution, where the fittest individuals are selected to produce offspring from generation to generation. A variety of design space choices are formulated as a gene. Each genome will evaluate its fitness (i.e., performance or energy) through the cost model. The GA will select those genomes with the highest fitness to produce their offspring in the next iteration.

The first step of GA is to initialize a set of genomes. In our model, we initialize 20 genomes for each generation, where each genome is composed of one random combination of design choices as shown in Table 2. After this, all the genomes will be ranked according to their fitness, where only elite genomes will be selected as parents. The rest unselected genomes will be discarded. In our model, we use *Half Fittest Selection* scheme, which will select half of the unselected genomes for the next generation. We use *Fittest* method by pairing parent candidates from the selected genomes. Upon the generation of parent candidates, we choose *Single Point* mechanism to generate offspring from the above parent genomes. A point on both parents' chromosomes is selected randomly, which is called the "crossover point". Furthermore, we use a method called *Reset* to perform mutation to generate better offspring, which is the final evolution operation. The new generation of offspring would be further evaluated by the cost model for fitness calculation. The GA will be terminated when it satisfies a certain condition. For example, if latency is selected as the major objective, the genome with the lowest latency will have higher fitness. Our proposed model considers latency, energy consumption, or energy-delay product. In our model, the GA search will be terminated after 3000 iterations based on our empirical study or when the fitness converges.

Table 2: Design Space for the GCN Accelerator

Description	Parameter	Set of Values
Tiling factors of different matrix dimensions for combination phase (i, j, k) and aggregation phase (u, v, w)	T_i, T_u, T_v	$min(2^1, 2^2, \ldots, 2^{12}, V)$
	T_j	$min(2^1, 2^2, \ldots, 2^{12}, F)$
	T_k, T_w	$min(2^1, 2^2, \ldots, 2^8, C)$
Number of PEs for combination phase	PE_c	1, 4, 16, 36, 64, 100
Number of PEs for aggregation phase	PE_a	1, 4, 16, 36, 64, 100
Compression format	c	CSR, CSC, COO, ZVC
Loop order for combination phase	l_c	Permutation(i, j, k)
Loop order for aggregation phase	l_a	Permutation(u, v, w)

Table 3: Dimension and Density of the Graph Datasets

Datasets	Dimension		Density (1-Sparsity)				
	Vertex	Feature	A	X1	X2	W1	W2
CR	2708	1433	0.18%	1.27%	78.0%	100%	100%
CS	3327	3703	0.11%	0.85%	89.1%	100%	100%
NL	65755	61278	0.0073%	0.011%	86.4%	100%	100%
PB	19717	500	0.023%	10.0%	77.6%	100%	100%
RD	232965	602	0.21%	51.6%	60.0%	100%	100%

5 EVALUATION

ASIC Synthesis. To evaluate the area and power consumption, we model all the PE logic, including the MAC Array, buffers, and DRAM through synthesis. We use the Synopsys Design Compiler with the TSMC 28*nm* library for the synthesis and estimate the power using Synopsys PrimeTime PX. All the power and area parameters will be integrated into our cost model for design space exploration.

Hardware Simulator. We built a cost model, upon MAESTRO [15], to abstract the behavior of the hardware, dataflow, and compression technique with an area constraint as $10mm^2$. The simulator models the microarchitectural behaviors of all mentioned architectural designs, dataflows, and compression formats. The simulator counts the number of DRAM reads and writes, which is used to estimate the DRAM access energy consumption according to [15]. We validate our model against a modified scale-sim [28] for SpMM, which is an open-source cycle-accurate simulator. The accuracy of our model is within 1-3% as compared to scale-sim.

Benchmark Graph Datasets. We evaluate the GCN performance using Cora (CR), Citeseer (CS), Nell (NL), Pubmed (PB), and Reddit (RD) datasets. Details of dimensions and matrix density are shown in Table 3.

5.1 Benefits of Holistic DSE

5.1.1 Performance. The performance benefit of our holistic framework is evidenced by Figure 6(a), which shows that the holistic framework can achieve 12.3×, 2.2× and 1.37× speedup on average as compared to HyGCN, AWB-GCN, and GCNAX, respectively. For a fair comparison, we set the number of processing elements and buffer size consistent for all the designs. The proposed design achieves 29.6× improvement on NL when compared to HyGCN as this dataset has higher sparsity on both adjacency and feature matrices, and only 4.5× improvement on RD since it has lower sparsity. This is because the decompression latency overheads become more significant when having more zero elements (higher sparsity).

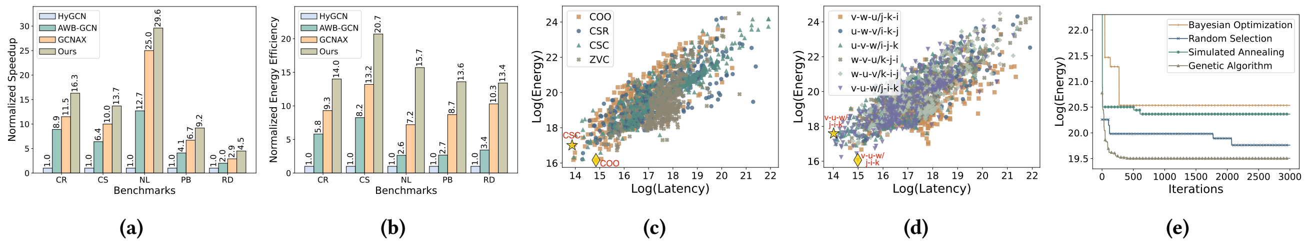


Figure 6: (a) Normalized speedup and (b) normalized energy efficiency between different GCN accelerators, (c) latency and energy consumption analysis with various compression formats of CR, (d) dataflow analysis of CR, and (e) convergence analysis of GCN design space exploration with different algorithms.

Table 4: Converged GCN designs with the best performance for genetic algorithm and random search

Benchmark	Search Algorithm	Objective (Latency, cycles)	Combination					Aggregation					Compression Format
			Ti	Tj	Tk	PE	Dataflow	Tu	Tv	Tw	PE	Dataflow	
CR	GA	9.86E+05	256	256	8	100	j-i-k	2048	256	4	36	v-u-w	CSC
	RS	1.13E+06	64	128	8	36	j-i-k	256	8	8	36	v-u-w	CSC
CS	GA	2.24E+06	256	256	8	64	i-j-k	128	2048	8	16	u-v-w	CSR
	RS	2.38E+06	512	2048	8	100	i-j-k	512	8	2	36	u-v-w	CSR
NL	GA	1.2E+07	1024	1024	32	100	k-j-i	512	1024	4	64	v-u-w	CSC
	RS	1.61E+07	1024	512	2	36	j-i-k	4096	512	2	100	v-u-w	COO
PB	GA	2.58E+06	4096	128	8	64	i-j-k	512	512	8	100	u-v-w	CSR
	RS	2.99E+06	64	128	8	100	j-i-k	512	512	8	16	v-u-w	CSC
RD	GA	2.47E+09	64	64	16	16	k-j-i	2048	4096	16	100	v-u-w	CSC
	RS	2.6E+09	256	128	16	64	i-j-k	2048	4096	4	64	u-v-w	CSR

5.1.2 Energy. Figure 6(b) shows the energy consumption of our holistic exploration, architecture exploration, and dataflow exploration. The result implies that the holistic work can achieve 15.3×, 3.7×, and 1.6× energy efficiency on average as compared to HyGCN, AWB-GCN, and GCNAX. We achieve 20.7× energy efficiency on CS, as with well-selected dataflow and compression format, this dataset reduces the highest percentage of off-chip memory accesses (up to 89.7%). The proposed design achieves less energy efficiency improvement (13.4×) on RD also because of the lower sparsity.

5.2 Compression Format Analysis

Figure 6 (c) shows the latency and energy results of selecting different compression formats in various GCN accelerator designs. Given the limited space, we only present CR benchmark. In CR, the most energy-efficient GCN design selects COO (up to 67.4% energy reduction as compared to other compression formats) as the compression format, whereas CSC could deliver the best performance (73.7% performance improvement as compared to others).

COO is the most energy-efficient compression format, as it is compatible with both column-wise and row-wise access patterns. To achieve a better performance, the compression format has to be tailored to the specific matrix access pattern. ZVC is never selected in the first layer (X1) of the combination phase, as it performs well only in those applications with a larger density (> 5%).

5.3 Dataflow Analysis

To study the impact of loop interchange in GCN accelerators, we iterated all possible loop combinations via an exhaustive search. Due to a large number of combinations (i.e., $A_3^3 \times A_3^3$), we only present those commonly used loop combinations in Figure 6 (d). It should be noted the loop order is only considered for the off-chip memory access. Through the simulation, we concluded several interesting observations. The loop order is mostly altered for the purpose of reducing off-chip memory access in CNNs. However, the selection in loop interchange appears to be different when facing GCN applications. For example, in the CR benchmark, the loop order of the aggregation phase, v-u-w, is selected for achieving optimal latency. The order indicates two important trends in optimizing GCN performance. First, the v-u order matches the column access required by the selection of CSC. This indicates that the access patterns of sparse matrices and compression formats should be consistent. Second, loop w is the innermost loop which is irrelevant to the sparse matrix. This indicates that loop order could be used to overlap decompression latency. The loop order of the combination phase also indicates a similar trend. To further validate this observation, we applied the conventional CNN loop selection approach [26] to GCN applications without considering decompression latency, in which 7.6-26.4 × latency degradation was observed across different applications if incompatible compression format is selected.

5.4 GA-based Search Model Analysis

5.4.1 Analysis of Optimized Design Space. Our proposed model has three objectives to guide performance optimization: latency, energy, and energy-delay product. Though four optimization algorithms, GA-based (GA), Random Selection (RS), Simulated Annealing (SA), and Bayesian Optimization (BO), are deployed, only the performance of GA and RS is presented in Table 4. The deployment of GA can deliver design choices with better performance (up to 25.6% latency reduction) and energy efficiency (up to 27.2% energy reduction) as compared to RS.

5.4.2 Convergence Time Study. Convergence is a critical measure to estimate how long the search will take to find the ultimate result. In this study, we compare the convergence time of the genetic algorithm against the other three baseline optimization techniques. As shown in Figure 6 (e), the genetic algorithm outperforms other baseline algorithms with an average 41.2% of improvement in convergence time as compared to RS. It can also be noted that the genetic algorithm can also suggest the design with the best performance, energy, and energy-delay product among all algorithms.

6 CONCLUSION

In this paper, we endeavor to develop a holistic framework in support of the rapid exploration of GCN design space - synergizing architecture, dataflow, and sparsity optimizations. To accomplish this, we study and formulate a variety of compression formats tailored for handling extreme and irregular sparsity in GCN architectures. Leveraging this study, we develop a comprehensive cost model with the aim of unraveling the dynamic interactions and trade-offs among various design choices in architecture, dataflow, and sparsity. In addition, we apply a genetic algorithm to facilitate the rapid exploration of GCN design space. Lastly, through extensive exploration and observation, we summarize takeaways for designing and utilizing modern and future GCN accelerators.

ACKNOWLEDGMENTS

This project was supported in part by the US National Science Foundation Grant CCF-1717388, CCF-1907765, and PPoSS-2028481.

REFERENCES

[1] 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 *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29, 2020.

[2] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936, 2020.

[3] J. Li, A. Louri, A. Karanth, and R. Bunescu. GCNAX: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 775–788, 2021.

[4] A. Abdolrashidi, D. Tripathy, M. Belviranli, L. Bhuyan, and D. Wong. WIREFRAME: Supporting data-dependent parallelism through dependency graph execution in gpus. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 600–611, 2017.

[5] X. Ma, D. Zhang, and D. Chiou. FPGA-accelerated transactional execution of graph workloads. In *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, page 227–236, 2017.

[6] F. Sadi, J. Sweeney, S. McMillan, T. Low, J. Hoe, L. Pileggi, and F. Franchetti. PageRank acceleration for large graphs with scalable hardware and two-step spmv. In *Proceedings of IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–7, 2018.

[7] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian. GraphQ: Scalable pim-based graph processing. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture(MICRO)*, page 712–725, 2019.

[8] H. Yang. Aligraph: A comprehensive graph neural network platform. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, page 3165–3166, 2019.

[9] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich. PyTorch-BigGraph: A large-scale graph embedding system. In *Proceedings of Machine Learning and Systems (MLSys)*, pages 120–131, 2019.

[10] A. Ren, T. Zhang, Y. Wang, S. Lin, P. Dong, Y. Chen, Y. Xie, and Y. Wang. DARB: A density-adaptive regular-block pruning for deep neural networks. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, pages 5495–5502, 2020.

[11] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *Proceedings of ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 548–560, 2017.

[12] S. Kwon, D. Lee, B. Kim, P. Kapoor, B. Park, and G. Wei. Structured compression by weight encryption for unstructured pruning and quantization. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1906–1915, 2020.

[13] Rumi, A. Masuma, X. Ma, Y. Wang, and P. Jiang. Accelerating sparse CNN inference on GPUs with performance-aware weight pruning. In *Proceedings of ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 267–278, 2020.

[14] E. Qin, G. Jeong, W. Won, S. Kao, H. Kwon, S. Srinivasan, D. Das, G. E. Moon, S. Rajamanickam, and T. Krishna. Extending sparse tensor accelerators to support multiple compression formats. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1014–1024, 2021.

[15] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar. MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. In *IEEE Micro*, pages 20–29, 2020.

[16] A. Parashar, P. Raina, Y. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer. Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 304–315, 2019.

[17] Y. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[18] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler. Compressing DMA engine: Leveraging activation sparsity for training deep neural networks. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 78–91, 2018.

[19] N. Pezzotti, B. F. Lelieveldt, L. Maaten, T. Hollt, E. Eisemann, and A. Vilanova. Approximated and user steerable tSNE for progressive visual analytics. In *IEEE Transactions on Visualization & Computer Graphics*, pages 1739–1752, 2015.

[20] B. Asgari, R. Ramyad, and H. Kim. ASCELLA: Accelerating sparse computation by enabling stream accesses to memory. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 318–321, 2020.

[21] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70, 2020.

[22] J. Li, H. Zheng, K. Wang, and A. Louri. SGCNAX: A scalable graph convolutional neural network accelerator with workload balancing. In *IEEE Transactions on Parallel & Distributed Systems (TPDS)*, pages 2834–2845, 2021.

[23] J. Yang, H. Zheng, and A. Louri. Adapt-Flow: A flexible DNN accelerator architecture for heterogeneous dataflow implementation. In *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, page 287–292, 2022.

[24] J. Yang, H. Zheng, and A. Louri. Venus: A versatile deep neural network accelerator architecture design for multiple applications. In *Proceedings of Design Automation Conference (DAC)*, 2023.

[25] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of ACM Field Programmable Gate Arrays (FPGA)*, pages 161–170, 2015.

[26] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 343–348, 2018.

[27] S. Gandham, L. Yin, H. Zheng, and M. Lin. OCMGen: Extended design space exploration with efficient FPGA memory inference. In *Proceedings of IEEE International Symposium On Field-Programmable Custom Computing Machines (FCCM)*, 2023.

[28] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna. SCALE-Sim: Systolic CNN accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.