HyperGRAF: Hyperdimensional Graph-based Reasoning Acceleration on FPGA

Hanning Chen*, Ali Zakeri*, Fei Wen†, Hamza Errahmouni Barkam* and Mohsen Imani*
*University of California, Irvine, Irvine, CA 92697, USA
†Texas A&M University, College Station, TX 77843, USA
Email: *{hanningc, azakerij, herrahmo, m.imani}@uci.edu, †fei8wen@gmail.com

2023 33rd International Conference on Field-Programmable Logic and Applications (FPL) | 979-8-3503-4151-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/FPL60245.2023.00013

***Abstract*—The latest hardware accelerators proposed for graph applications primarily focus on graph neural networks (GNNs) and graph mining. High-level graph reasoning tasks, such as graph memorization and neighborhood reconstruction, have barely been addressed. Compared to low-level learning applications like node classification and clustering, high-level reasoning typically requires a more complex model to mimic human brain functionalities. Brain-inspired Hyper-Dimensional Computing (HDC) has recently introduced a promising lightweight and efficient machine learning solution, particularly for symbolic representation. General-purpose computing platforms (CPU/GPU) have been revealed to be inefficient for HDC applications. Therefore, it becomes essential to design a domain-specific accelerator targeting HDC-based graph reasoning algorithms.**

In this work, we propose the first domain-specific accelerator for HDC-based graph reasoning, HyperGRAF. We first develop a scheduler to balance the sparse matrix computation workloads, before parallelizing the hypervector calculations on two levels for the graph memorization task. Finally, we design a pipeline-style matrix multiplication accelerator for the neighborhood reconstruction task. We evaluate our design under a wide range of generated graphs with different sizes and sparsity. The results show that HyperGRAF achieves over 100× improvement in both speedup and energy efficiency of graph reasoning compared to NVIDIA Jetson Orin.

I. Introduction

Designing domain specific accelerators (DSA) targeting graph-based applications has drawn the attention of researchers in recent years. Most of the previous acceleration works focus on accelerating low-level graph learning applications, such as graph node classification and graph mining [1], [2] and barely consider high-level graph reasoning tasks. Unlike computer programs, the human brain memorizes graph information and retrieves node information at an astonishing speed. A principal advantage of the brain over computer programs is the fact that the former effectively uses billions of neurons to conduct computing, while sample-based deep neural networks (DNNs) and recursive algorithms are naturally much less efficient. Therefore, to conduct high-level graph reasoning tasks more efficiently, using a novel model that closely mimics the brain functionalities is beneficial.

Recently, Hyper-dimensional Computing (HDC) has been introduced as a brain-inspired computational model for high-efficiency and noise-tolerant computation [3]. Unlike DNN, HDC is a model of the cerebellum cortex, which biologically represents the human memory. HDC is motivated by the observation that the cerebellum cortex operates on high-dimensional data representations [4]. Recently, HDC-based machine learning models have shown promising results in various cognitive tasks on various types of data including text, graph, voice, genome sequence, and image [5]–[18]. Graphd [19] is the first proposed framework for human brain's high-level graph information memorization and neighborhood reconstruction.

While Graphd [19] achieves great accuracy on the high-level graph reasoning, it also reveals that performing HDC-based graph reasoning (**HGR**) applications require a significant amount of computation time. To approach the issue, Graphd [19] also designs an accelerator based on digital processing in memory (DPIM). However, much more potential improvements in HGR acceleration on the DSA platform have yet to be exploited from the perspective of computer architecture research. First, Graphd [19] does not consider graph sparsity, specifically for computing workload unbalance during the sparse matrix multiplication (SpMM) process [20]. Second, to apply PIM to graph processing, previous works have shown that pre-processing over the input graph is necessary [21], [22]. However, graph pre-processing generally requires a large amount of computation time, which obstructs the accelerator's online learning capability. Last but not least, compared to other computing platforms, such as ASIC and FPGA, PIM lacks computing flexibility and is hard to be deployed on edge devices.

To the best of our knowledge, we propose the first FPGA-based hyperdimensional graph reasoning acceleration platform, called HyperGRAF. We conducted hardware-software co-design to maximize graph reasoning throughput by considering FPGA's resource utilization. Here are the main contributions of the paper:

- We analyze the previous HGR algorithms and identify their computing bottlenecks by comparison with traditional GNN workloads.
- We propose a software scheduler running on the host CPU to balance the computing workload when processing sparse graphs.
- We parallelize the graph memorization computing at both node level and intra-hypervector level. The computation of hypervector chunks is pipelined.
- We design a pipeline-style decoder IP to accelerate node reconstruction, which achieves a balance between execution speedup and resource utilization.

We also evaluate our approach on a wide range of generated

1946-1488/23/$31.00 ©2023 IEEE
DOI 10.1109/FPL60245.2023.00013

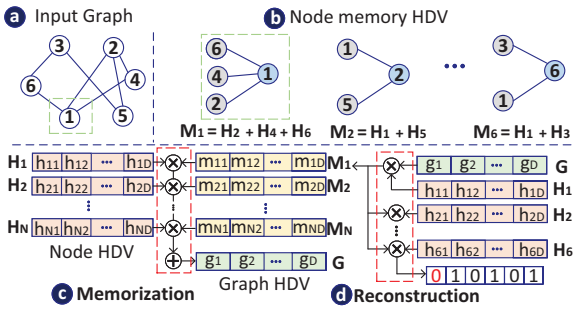

Fig. 1. Hyperdimensional graph reasoning (HGR) example. a. Example input graph. b. Memory node hypervector generation. c. Graph memory hypervector generation (memorization). d. Node memory reconstruction.

graphs. The results show that the FPGA platform provides, on average, over 100× and 10× improvement for both reasoning speedup and energy efficiency compared to the Jetson Orin and previous acceleration methods running on DPIM, respectively.

II. Hyperdimensional Graph Reasoning

This section introduces hyperdimensional graph reasoning (HGR) from an algorithmic perspective. Section II-A summarizes HGR's core operations, initially developed by [19]. Section II-B compares HGR with graph neural networks (GNNs) [23] and shows their difference. In Sections II-C, we analyze the most time-consuming part of HGR and illustrate that developing domain-specific accelerators (DSA) targeting graph reasoning applications is necessary.

A. Hyperdimensional Graph Reasoning

Figure 1 shows the general HGR procedure. HGR supports two high-level reasoning tasks: graph memorization (**memorization**) and graph reconstruction (**reconstruction**). Graph memorization is the process of compressing the information of a graph into a single hypervector. Graph reconstruction aims to rebuild the relations between entities based on the previously done memorization.

Graph Memorization The memorization process (Figure 1(a)) includes two steps: the node memory hypervector generation, and node memory bundling. Suppose we assign a random hypervector $\vec{H}_i$ to each node as its feature vector and use matrix $\mathbf{H}$ to represent the concatenation of all the generated hypervectors for nodes (i.e. the feature matrix). The dimension of matrix $\mathbf{H}$ is $|V| \times D$, where $|V|$ is the number of nodes in the graph, and D is the dimension of hypervectors. The memory hypervector is generated by aggregating each node's neighbors' feature hypervectors: $\mathbf{M} = \mathbf{A_g} \cdot \mathbf{H}$, where $\mathbf{M}$ is the matrix representation of graph node memories, and $\mathbf{A_g}$ is the graph's adjacency matrix. Figure 1(b) provides an example of node memory hypervectors generation based on the graph shown in Figure 1(a). The memory bundling process consists of binding each node's hypervector with its memory hypervector and bundling the results across all nodes, as shown in Figure 1(c):

$$\vec{G} = \sum_{i=1}^{V} \vec{H}_i \circ \vec{M}_i \tag{1}$$

TABLE I
Comparison of HDC-based graph reasoning (HGR) with GNN.

	Task Level	Kernel	Datasets	Sparsity
HGR	high level	Hypervector	Small ∼ Large	High
GNN	low level	Neural Network	Large	High

Where $\vec{G}$ represents the final graph hypervector, $\circ$ is the element-wise multiplication (Hadamard product) which performs binding, and the sum operation represents bundling. Also note that the binding operation associates two hypervectors' information, whereas the bundling operation synthesizes the information of all the hypervectors it is used on [24].

Graph Reconstruction Figure 1(d) gives an example of graph node memory reconstruction. To determine each node's neighbor nodes, first we need to reconstruct each node memory hypervector. Since randomly generated hypervectors are orthogonal [24], the node memory computation can be approximated as:

$$\vec{M}_i = \vec{G} \circ \vec{H}_i \ where \ i \in [1:V] \tag{2}$$

However, as the graph size ($|V|$) increases, the node memory matrix generated by Equation 2 will have high amounts of noise which undermines subsequent neighbor node reconstruction. Graphd [19] proposes an iterative computing method to mitigate the noise:

$$\vec{M}_i^k = \vec{H}_i \cdot (\vec{G} - \Sigma_{j \neq i} \vec{H}_j \cdot \vec{M}_j^{k-1}) \ where \ k > 0 \tag{3}$$

Here k is the iteration cycle. Say we have an auxiliary hollow matrix $\mathbf{I_A}$ of shape $|V| \times |V|$ whose diagonal elements are zero and all others are 1. By introducing $\mathbf{I_A}$, Equation 2 becomes:

$$\mathbf{M}^k = \mathbf{G} \circ \mathbf{H} - \mathbf{I_A} \cdot (\mathbf{M}^{k-1} \circ \mathbf{H}) \circ \mathbf{H} \ where \ k > 0 \tag{4}$$

With the memory hypervector generated from each node, we can determine whether two nodes are connected, by checking the cosine similarity between node A's memory hypervector ($\vec{M}_A$) and node B's feature hypervector ($\vec{H}_B$). Here we denote this cosine similarity by f(A,B), then A and B are determined to be connected if f(A,B) $\approx$ 1, or disconnected if f(A,B) $\approx$ 0.

B. Comparison with Graph Neural Network

Although the graph memorization of HGR resembles the graph neural network (GNN)'s aggregation phase, its nature is different. GNN has been widely adopted for traditional machine learning tasks, such as classification and clustering, over graph datasets [1], [21]. While GNN manifests excellent learning capabilities, it does not support high-level reasoning such as memorization and knowledge extraction. The underlying reason is that neural networks generally excel in learning, but not storing information. By contrast, hyperdimensional learning relies on symbolic hypervectors and therefore has brain-like solid reasoning capabilities [19]. Besides the distinct target tasks, the computing procedure is also different. GNN repeats the nodes aggregation operation over multiple layers, whereas that only happens once during the memorization phase as for HGR. We tabulate the comparisons between HGR and GNN in Table I.

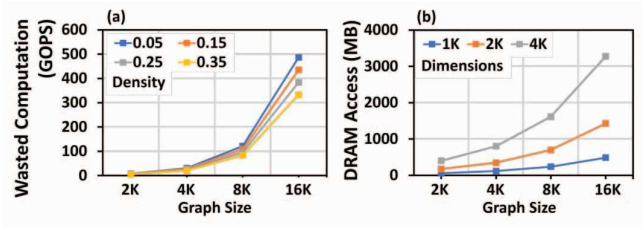

Fig. 2. (a) Amount of wasted computation in GOPS vs. graph size, for different graph densities. (b) Volumes of DRAM accesses in MB vs. graph size, for different dimensions.

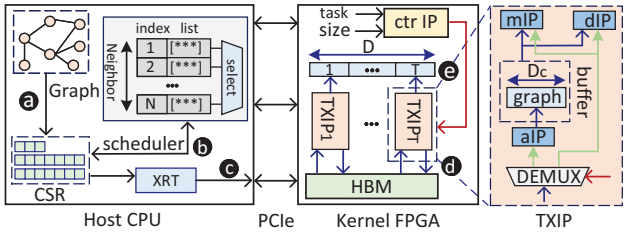

Fig. 3. Top Architecture. (a) Generate CSR representation. (b) Load CSR matrix into out of order scheduler. (c) Offload hyperdimensional computing activities into the FPGA kernel. (d) Transaction IP (TXIP) accesses a node's feature vector from high bandwidth memory (HBM) channel. (e) Concatenate each TXIP's chunk vector and generate the final result hypervector.

C. HGR Computing Bottleneck Analysis

While Graphd [19] presents a novel hyperdimensional graph reasoning method, we identified certain inefficiency and performance bottlenecks in the two processes: graph information memorization and reconstruction. For the memorization process, Graphd [19] does not consider graph sparsity, and hence incurs a notable amount of inconsequential computation as shown in Figure 2(a). The reconstruction process, represented by Equation 4, is composed of an inner product and Hadamard products which are not commutative. Therefore each iteration requires three consecutive matrix multiplication operations. Assuming that we use GPU to accelerate Equation 4, an analysis of DRAM accesses in Figure 2(b) suggests that excessive time and energy is consumed by data transfer between the host CPU and kernel GPU. In Section III, we design a domain-specific accelerator (DSA) to address this challenge.

III. FPGA Acceleration of HyperGRAF

In this section, we present HyperGRAF's architecture design. Based on bottleneck analysis in Section II-C, we conduct a hardware-software co-design for both the memorization and reconstructions phase. In Section III-A, we give an overview of the CPU-FPGA heterogeneous platform. Section III-B gives the algorithm of the out-of-order (OoO) scheduler, which helps to reduce the sparsity and balance the computing workload. We show actual computing unit architecture for memorization and reconstruction in Sections III-C and III-D.

A. Top Architecture

Figure 3 is an overview of the CPU-FPGA heterogeneous platform. We first convert the graph representation from adjacency matrix format into compressed sparse row (CSR) format. Although the CSR format successfully diminishes the matrix's sparsity, it also incurs the computing workload imbalance problem as discussed by previous works [20], [25]. To fix this issue, we design an out-of-order (OoO) style, density-aware scheduler running on the CPU. The scheduler will offload actual hyperdimensional computing (HDC) activities on kernel FPGA. Here suppose the hypervector dimension is $\mathbf{D}$. To parallelize the matrix multiplication (MM), we split each graph node's hypervector into $\mathbf{T}$ chunks. Each chunk's dimension $\mathbf{D_c} = \frac{\mathbf{D}}{\mathbf{T}}$. As shown in Figure 3, those T vectors will be first loaded into T high bandwidth memory (HBM)'s channels and then accessed by T independent transaction IP (TXIP). Inside each TXIP, there is one aggregator IP (aIP) and one decoder IP (dIP). The aggregator IP will conduct memorization computing, and the decoder IP will conduct reconstruction computing. After each TXIP finishes its computing activities, we concatenate each channel's chunk and generate the final result hypervector.

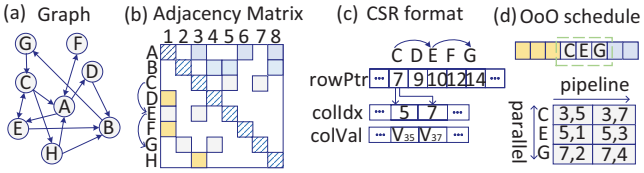

Fig. 4. Sparse Matrix to Dense Matrix Multiplication Optimization. (a) Example graph (b) Adjacency matrix representation of graph (c) CSR format of a graph (d) Out-of-order (OoO) schedule of CSR-based graph processing

B. Density-aware Scheduler

As discussed in Section II-C, effective sparsity reduction is critical for graph memorization computing. Figure 4 illustrates an example of the sparsity problem. Figure 4(a) is the target graph, and Figure 4(b) is the adjacency matrix representation. The blank region of the matrix are the elements of zero value, i.e., the source of sparsity. For such a sparse matrix, each vertex's neighbor aggregation process, as represented by Equation 1, is a sparse matrix multiplication problem (SpMM). A common approach to reduce matrix sparsity is to represent the graph in the CSR format, as is shown in Figure 4(c). However, CSR is a vertex-centric graph format that potentially incurs poor computing workload balancing, due to the discrepancy in vertices' neighbor sizes. Specifically, if we solely parallelize the computing by matrix's row pointer (rowPtr), each threads' computing complexity will be different [26]. Previous works [21], [27], proposed graph pre-processing, such as graph clustering, to group vertices with the common neighbor. However, the pre-processing is time-consuming and graph-specific. Alternatively we propose an out-or-order (OoO) style, density-aware scheduler to balance the computing workload, as described in algorithm 1. This idea was inspired by the famous Tomasulo's algorithm [28], in the sense that it dynamically offloads vertices with the same neighbor count to the kernel FPGA. In addition to exploiting the parallelism among different vertices, we also pipeline each vertex's neighbor feature vector aggregation process. The scheduler will scan the CSR

Algorithm 1: Density-aware Scheduler

Input: rowPtr, colIdx, colVal, Nc

```
/* i is the vertex's index                                   */
i = 0;
H = map<int, list>;
while i < V do
    /* count is vertex's neighbor size, H[count]
       is a C++ STL list storing indices                     */
    count = rowPtr[i + 1] - rowPtr[i];
    H[count].add(i);
    if H[count].size == Nc then
        /* kernel is the FPGA accelerator                    */
        call kernel(H[count], rowPtr, colIdx, colVal);
        H[count].clear;
    end
    i++;
end
/* Finish the remaining computation                          */
count = 0;
while count < H.size do
    if H[count].size > 0 then
        call kernel(H[count], rowPtr, colIdx, colVal);
        H[count].clear;
    end
    count++;
end
```

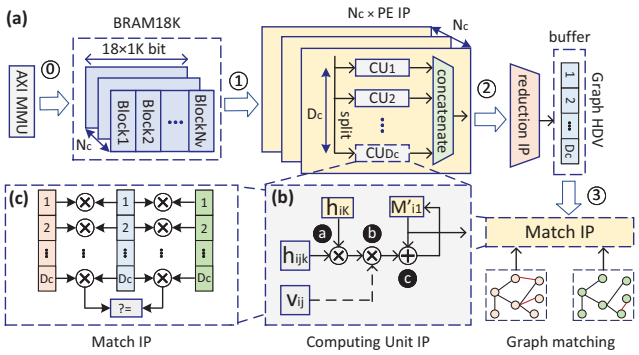

Fig. 5. Aggregator IP architecture. (a) General architecture. (b) Computing Unit (CU) microarchitecture. (c) Match IP architecture.

matrix, and store each vertex's id into a hashmap. The key of the hashmap is neighbor size and the value is a list of the related vertices' IDs. Assume that the kernel FPGA supports a maximum of $\mathbf{N_c}$ concurrent vertex aggregation computing, then whenever the size of the list grows to N_c, we offload the corresponding vertex hypervetor computation into the kernel FPGA. After the kernel computing finishes, we clear this list and continue the scanning. Figure 4(d) shows an example where vertex C, E, and G have the same neighbor size, therefore the scheduler will pack their computation tasks onto the kernel FPGA.

C. Encoder Architecture

Figure 5 presents the architecture design of the encoder IP, which is responsible for the hyperdimensional graph memorization. Let N_c denote the on-chip node parallelism, i.e., the number of nodes that HyperGRAF can simultaneously process. Here, N_c is directly determined by the amount of FPGA on-chip resources, such as the lookup tables (LUT). The encoder IP first loads vertex hypervector chunks from the corresponding HBM channels. We store each vertex's neighbor feature vector to BRAM for later pipeline processing and forward the vertex's feature vector to the on-chip buffer. In Figure 5, there are N_c independent processing element (PE) IP and each PE IP has D_c computing unit (CU) IP. Figure 5 also provides the microarchitecture of the CU IP. Let $\mathbf{h_{ik}}$ be the k^{th} feature vector element of vertex i, and let $\mathbf{r_{ijk}}$ represent vertex i's neighbor feature vector element, with **j** as the pipeline stage index. V_{ij} stands for the edge value, which is always 1 for unweighted graphs. The computation executed by CU IP can be written as:

$$M^{'}_{ik} = M^{'}_{ik} + h_{ik} \times h_{ijk} \times V_{ij} \ \ where \ k \in [1 : D_c] \qquad (5)$$

The last step inside CU IP is to accumulate the previous multiplication product and to generate each vertex's memory vector $\vec{M}'$. Please note that the memory hypervector $\vec{M}'$ in Figure 5 is already the Hadmard product of the memory hypervector and vertex feature vector: $\vec{M}^{'}_{i} = \vec{M}_i \circ \vec{H}_i$. At the end of the pipeline, we aggregate all N_c PE IP's results to generate the partial graph memorization hypervector.

D. Decoder Architecture

The graph reconstruction is the most time-consuming part of HGR due to two facts. First, we need to iteratively remove noise from the memory node hypervector. Our solution is to increase the hypervector dimension, and hence increase the hypervector orthogonality. Second, Equation 4 takes a long time to complete if without any hardware optimization. In that respect, we design a domain-specific accelerator, called decoder IP. If we take a closer look at Equation 4, the main computing overhead is attributed to the second term, since the first term stays constant during iteration. Here we rewrite the second term as $\mathbf{M}^{'}$:

$$\mathbf{M}' = \mathbf{I_A} \cdot (\mathbf{M}^{k-1} \circ \mathbf{H}) \circ \mathbf{H} \ \ where \ k \ > \ 0 \qquad (6)$$

Equation 6 comprises one matrix inner product and two consecutive matrix Hadmard products. To accelerate those matrix operations on FPGA, the most straightforward method is deploying two systolic arrays, one for the inner product and the other for the Hadmard Product. Although the optimistic time complexity is $\mathcal{O}(N_c + D_c + 1)$; however, due to the hardware resource limit, it is impractical to fully parallelize Hadmard product and match the time complexity of $\mathcal{O}(1)$. To make our design more generic and resource-efficient, we present the pipeline decoder IP (dIP) in Figure 6. The dimension of this decoder IP is $N_c \times D_c$. For each row of the noise memory matrix $\mathbf{M}^{'}_{\mathbf{i}}$, we have:

$$\mathbf{M}^{'}_{\mathbf{i}} = (\sum_{j \neq i} \vec{M}_j \circ \vec{H}_j) \circ \vec{H}_i \ \ where \ i, j \in [1, Nc] \qquad (7)$$

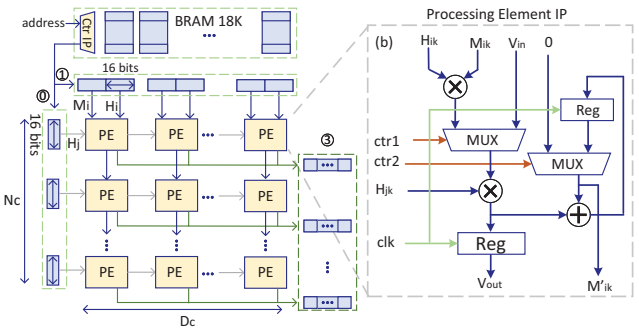

Fig. 6. Pipeline Style Decoder Architecture.

Since i is independent of summation, we can fuse two Hadmard products into one stage:

$$\mathbf{M}_{\mathbf{i}}^{'} = \sum_{j \neq i} \vec{M}_j \circ \vec{H}_j \circ \vec{H}_i \ \ where \ i, j \in [1, Nc] \qquad (8)$$

In Figure 6, we unroll the summation of Equation 8 over dIP's N_c row and distribute each element of feature vector over dIP's D_c column. As a result, we achieve a timing complexity of $\mathcal{O}(N_c+N_c)$ for a single iteration under the limited resource. As shown in Figure 6, each PE stores its own row index and it only accumulates those product results with a different row index. We believe this pipeline-style dIP successfully strikes a balance between performance and resource utilization.

IV. Evaluation

A. Experiment Setup

We synthesize and implement HyperGRAF on a CPU-FPGA heterogeneous computing platform, with the FPGA serving kernel acceleration. Specifically the CPU is Intel Core i9-12900 running at 3.2 GHz, the FPGA is Xilinx Alveo U50, and the connection between them is PCIe generated by the Xilinx Vitis framework [29]. The scheduler is developed with the Vitis Unified Software Platform which provides APIs, kernel drivers, board utilities, and firmware, to facilitate communication with the FPGA accelerator. We choose the generated graphs from work [30] as the workloads, and evaluate our design in terms of accuracy and speedup. We also analyze the impact of graph attributes, such as sparsity, on our accelerator's performance.

B. Graph Reasoning Accuracy

Figure 8(a) shows the effect of hypervector dimensionality and the number of edges on the quality of information retrieval. The number of nodes is set to 150 for all experiments. The results suggest that larger graphs require higher dimensionality to ensure full graph reconstruction. For example, graphs with 600 and 1000 edges can only be accurately stored and reconstructed using hypervectors with at least D = 2k and D = 3k dimensionality, respectively.

Figure 8(b) inspects the quality of graph matching using hypervectors with different dimensions. For all tests, the graph size is assumed to be fixed (50 nodes and 200 edges). Ideally there should be no edge mismatch, but that is untenable except for an oracle HDC model with zero noise and strictly orthogonal vectors. The x-axis in the plot shows the total number of edge differences, while the y-axis shows the error (i.e. number of mismatched edges) from our predictions. Each data point is the average of over 50 trials. The results show that decreasing the number of dimensions will give rise to the amount of noise in our estimation.

TABLE II
FPGA Resource Utilization on Xilinx Alveo U50 where the frequency is 200MHz and the power consumption is 29.8W.

	LUT	FF	BRAM*	UltraRAM*	DSP
aIP	244.4K	101.7K	128	0	0
dIP	268.3K	122.8K	0	64	2048
HBM	4320	3496	16	0	0
Other	72.1K	80.6K	94	0	0
Total	589.1K (83.6%)	308.7K (21.8%)	238 (21.3%)	64 (11.7%)	2048 (41.6%)

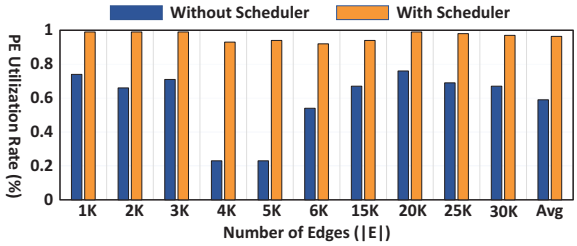

Fig. 7. The processing element (PE)'s utilization rate. The graph's vertex size ($|V|$) is constantly set to 1K. With the edge size ($|E|$) increase, the density of the graph also increase.

C. Resource Utilization

Table II presents the FPGA resource utilization when $D = 2K$, $D_c = 128$, $T = 16$, and $N_c = 4$. The precision of each hypervector's element is 8 bits. The frequency and power consumption of HyperGRAF are 200 MHz and 29.8 W. As is shown in Figure 3, we implement both aggregator IP and decoder IP on the same FPGA board. However, as discussed in Sections III-C and III-D, the graph memorization and reconstruction processes are independent, allowing us to implement the two IPs on separate FPGA boards. Another important detail is that the on-chip storage resources (BRAM and UltraRAM) in table II may vary with the graph changing. Specifically, the usage of BRAM is related to the graph's sparsity, as it stores each node's neighbor feature hypervector, and the usage of UltraRAM depends on the graph node size, as it is used to store each node's feature hyperevector during graph reconstruction. We will discuss more about resources and performance trade-off in Section IV-E.

D. Effect of Scheduler on HyperGRAF

Figure 7 illustrates the impact of a scheduler on increasing the utilization rate of processing elements (PEs) as the graph sparsity changes. To be specific, a PE is considered utilized/active when it has a computation load, rather than waiting for other PEs to finish their tasks. A higher utilization rate indicates a more balanced distribution of computation tasks across each PE. Figure 7 shows that, without a scheduler,

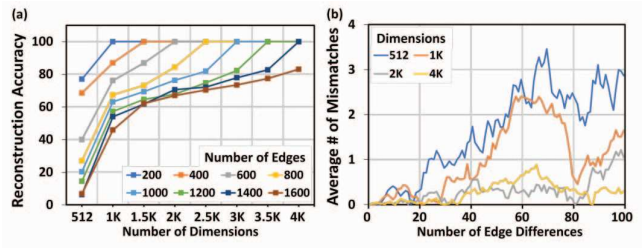

Fig. 8. (a) Graph reconstruction accuracy vs. dimensionality, for different numbers of edges (b) Graph matching error vs. number of edge differences between the two graphs

approximately 40% of PEs remain idle during the graph memorization computation. In contrast, the average utilization rate of HyperGRAF exceeds 95% with the scheduler. The substantial difference demonstrates the scheduler's ability to optimize task allocation and improve load balancing across PEs.

E. Cross Platform Comparison

Starting from this section, we discuss HyperGRAF's hardware acceleration performance from different perspectives. Figure 9 presents the graph reasoning accelerations performance on different hardware platforms. We implemented HDR algorithm proposed in [19] using PyTorch [31] and tested the program on NVIDIA Jetson Orin, NVIDIA GTX 1660, NVIDIA RTX 3090, and Intel i9-12900, while the digital processing in memory (DPIM) acceleration performance is based on [19]. A generated graph from [30] with the node size N_V=1000 and the average degree d_{avg}=2 is used in the experiments. Since graph memorization takes most of the execution time of the graph matching task, in Figure 9, we choose to use this task to benchmark the memorization performance. As shown in Figure 9, HyperGRAF achieves on average over 100× improvement over embedded system-on-chip (Jetson Orin) and CPU for both speedup and energy-delay product (EDP). With the increase of dimensionality, the advantage of HyperGRAF becomes more evident. When compared with DPIM, HyperGRAF shows on average over 10× speedup improvement. Even though both FPGA and DPIM have strong parallelism, DPIM's computing capability is restricted when it comes to sparse matrices. For HyperGRAF, however, a scheduler is running on the host CPU, that offloads HDC operations to kernel FPGA, balancing the computing workload and resulting in a significant improvement in the accelerator's performance. When it comes to energy efficiency, FPGA-based HyperGRAF shows an advantage over DPIM, although it is not as strong as the speedup difference as a result of FPGA's power consumption being much higher compared to PIM. For our design with 32 HBM channels usage, the power consumption is around 30W. Since Graphd [19] does not disclose the exact power consumption of their DPIM design, we are not able to do an in-depth power comparison.

F. Throughput vs Resource Utilization

Figures 10 and 11 present the trade-off between HyperGRAF's throughput and FPGA on-chip resource utilization. Here the hypervector dimension (D) is 4K, and the precision of each hypervector's element is 8 bits. Figure 10(a) gives the HyperGRAF's memorization throughput change and corresponding LUT usage when increasing the on-chip node parallelism (N_c). Here we use million embedded nodes per second (**MENPS**) to measure the node memorization throughput of HyperGRAF. MENPS is also used by previous FPGA-based graph processing accelerator designs [32], [33]. With the node processing parallelism (N_c) increasing, both the throughput and LUT utilization will increase. However for Xilinx Alveo U50, the maximum N_c is 16. If we keep increasing N_c, HyperGRAF's memorization throughput (MENPS) will not increase. In such case, to keep increasing throughput, we need to use larger FPGA board such as Alveo U280. Figure 10.(b) presents the influence of graph sparsity over HyperGRAF's memorization throughput. With the graph edge size increase, to maintain the constant reasoning speed (MENPS), we need to also increase on-chip node parallelism (N_c).

Figure 11(a) presents the effect of changing hypervector dimensionality (D) on HyperGRAF's performance. Figure 11(a) shows the performance comparison between systolic array style reconstruction IP (**dIP**) and pipeline style dIP. When dimensionality is low, systolic array style dIP offers high throughput at its high parallelism. However, with dimensionality increasing, pipeline-style dIP shows a better balance between resource utilization and reasoning throughput. Figure 11(b) shows that the on-chip storage (UltraRAM) will increase with dimensionality, as more space is needed to store the feature hypervectors.

TABLE III
COMPARISON WITH PREVIOUS GRAPH LEARNING FPGA ACCELERATOR

	Work [33]	HP-GNN [34]	HyperGRAF
Device	Alveo U200	Alveo U250	Alveo U50
Task	GNN Inference	GNN Training	Graph Reasoning
Dimension	300∼600	200∼600	512∼4K
Model	GCN	GCN and GraphSAGE	HGR
Precision	float32	float32	uint8
Frequency	250 MHz	300 MHz	200 MHz
Power	∼	∼	29.8 W
Throughput	2.64 MENPS	10.77 MENPS	0.9∼18.42 MENPS

G. Comparison with Previous FPGA Acceleration Works

While HyperGRAF is the first work that utilizes FPGA acceleration for the HDC graph reasoning model, we compared it with previous works on FPGA acceleration for graph learning in Table III. At the model level, HyperGRAF differs from traditional GNN models in that it stores graph information in a single HDV and reconstructs each vertex's neighbors. At the hardware level, due to HDC's holographic representations [35], the bit precision of each HDV's element is relatively low, but the embedding dimension (HDV dimension) is quite large. By parallelizing the graph memorization computation at both the node and intra-hypervector level, HyperGRAF achieves high computation throughput on a relatively small FPGA platform. Table III shows that HyperGRAF's throughput is influenced by factors such as graph sparsity, HDV dimension, and FPGA resources. To further extend our findings, future work could

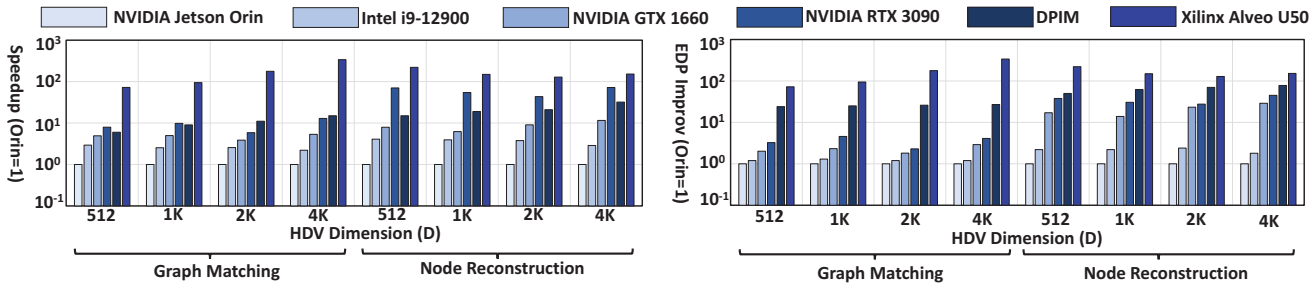

Fig. 9. Hyperdimensional graph reasoning (HGR) acceleration performance on different platforms. Here EDP represents energy delay product. We report the digital processing in memory (DPIM) acceleration result based on Graphd [19].

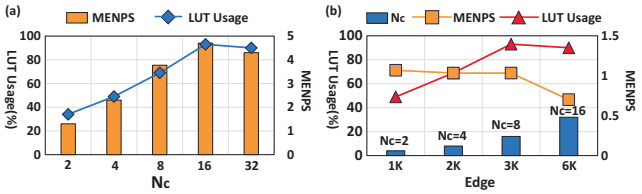

Fig. 10. (a) Vertex memorization throughput and LUT usage with different on-chip parallelism. (b) Vertex memorization throughput and LUT usage with different graph sparsity. We set the graph edge size as 1K in (a), and the graph node size V as 1K in both (a) and (b).

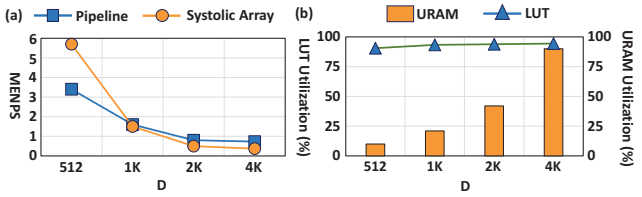

Fig. 11. (a) Neighborhood reconstruction with different hypervector dimension on two different architectures. (b) Pipeline-style reconstruction accelerator's Lookup table (LUT) and UltraRAM (URAM) utilization with different dimensionalities. We set the number of nodes $|V|$ as 1K and the average node degree d_{avg} as 6, in both (a) and (b).

involve deploying HyperGRAF on larger FPGA boards to evaluate its performance and scalability in more complex scenarios.

V. Related Works

Accelerating graph-related algorithms, such as GNNs on DSA, including FPGA [1], [27], [36], [37], ASIC [38], and PIM [21], has recently amassed considerable attention. Previous works focus on accelerating graph learning applications, such as classification [39] and graph mining [40], but rarely address high-level graph reasoning applications like graph memorization and reconstruction. [41], [42] inspire the possibility of interconnecting multiple FPGAs or heterogeneous systems for larger scale tasks, while [43]–[47] address the ever-increasing memory/storage requirement. Hyperdimensional computing (HDC) has recently shown much more potential in graph applications compared to traditional machine learning methods such as CNN or GNN [6], [19], [48], [49]. The work in [6] encodes a graph into an hypervector and performs graph classification. The study conducted by [48] presents a novel approach utilizing a ferroelectric field-effect transistor (FeFET)-based processing in-memory (PIM) hardware accelerator specifically designed for enhancing the performance of HDC-based graph models. Graphd [19] proposes the first HDC-based graph reasoning algorithm, which has been revealed to be infeasible for traditional computing platforms, namely CPU and GPU. It also attempts to accelerate its application with NVMe-based PIM but it fails to consider graph sparsity and workload balance. Moreover, PIM is hard to deploy and lacks flexibility compared to FPGA. The application of HDC models on the FPGA platform has demonstrated exceptional performance capabilities [50]–[55]. In particular, the study conducted by [53] aims to enhance the efficiency of HDC model learning on FPGAs, with a focus on classification tasks. Another research endeavor, conducted by [50], endeavors to expedite HDC regression model computations on FPGAs. Additionally, [51] explores the utilization of FPGA acceleration for HDC reinforcement learning models. Based on all these observation, we propose HyperGRAF, the first FPGA-based hardware acceleration of the HDC-based graph reasoning algorithms.

VI. Conclusion

In this paper we conduct a software-hardware co-design to propose the first FPGA-based hyperdimensional graph reasoning acceleration platform. We design a scheduler to balance the FPGA kernel computing workload from the software perspective, while exploring the trade-off between reasoning throughput, on-chip resource utilization and graph parameters on the hardware side. We evaluate our design under a wide range of generated graphs. The results show that HyperGRAF platform provides over 100× and 10× improvement for both speedup and energy efficiency, compared to the Jetson Orin and prior DPIM-based accelerator, respectively.

VII. Acknowledgement

This work was supported in part by DARPA, National Science Foundation #2127780 and #2312517, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, and generous gifts from Xilinx and Cisco.

REFERENCES

[1] T. Geng *et al.*, "I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization," in *IEEE/ACM MICRO-54*, pp. 1051–1063, 2021.

[2] X. Chen *et al.*, "FlexMiner: A pattern-aware accelerator for graph pattern mining," in *ACM/IEEE ISCA 2021*, pp. 581–594, IEEE, 2021.

[3] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.

[4] Z. Zou *et al.*, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *ACM SC*, pp. 1–15, 2021.

[5] F. Liu *et al.*, "L3E-HD: A framework enabling efficient ensemble in high-dimensional space for language tasks," in *ACM SIGIR*, pp. 1844–1848, 2022.

[6] I. Nunes *et al.*, "GraphHD: Efficient graph classification using hyperdimensional computing," in *DATE 2022*, pp. 1485–1490, IEEE, 2022.

[7] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *IEEE ICRC 2017*, pp. 1–8, IEEE, 2017.

[8] M. Imani, A. Zakeri, H. Chen, *et al.*, "Neural computation for robust and holographic face detection," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 31–36, 2022.

[9] M. Issa *et al.*, "Hyperdimensional hybrid learning on end-edge-cloud networks," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pp. 652–655, IEEE, 2022.

[10] Y. Ni *et al.*, "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1141–1146, 2022.

[11] Y. Ni *et al.*, "Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[12] Y. Ni *et al.*, "Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 292–297, IEEE, 2022.

[13] Z. Zou *et al.*, "Biohd: an efficient genome sequence search platform using hyperdimensional memorization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 656–669, 2022.

[14] Z. Zou *et al.*, "Eventhd: Robust and efficient hyperdimensional learning with neuromorphic sensor," *Frontiers in Neuroscience*, vol. 16, 2022.

[15] C.-K. Liu *et al.*, "Cosime: Fefet based associative memory for in-memory cosine similarity search," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[16] H. E. Barkam *et al.*, "Hdgim: Hyperdimensional genome sequence matching on unreliable highly scaled fefet," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2023.

[17] Z. Zou *et al.*, "Memory-inspired spiking hyperdimensional network for robust online learning," *Scientific Reports*, vol. 12, no. 1, p. 7641, 2022.

[18] M. Imani *et al.*, "Hierarchical, distributed and brain-inspired learning for internet of things systems," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2023.

[19] P. Poduval *et al.*, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, p. 5, 2022.

[20] L. Song *et al.*, "Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication," in *ACM/SIGDA FPGA 2022*, pp. 65–77, 2022.

[21] Y. Zhu *et al.*, "Exploiting parallelism with vertex-clustering in processing-in-memory-based GCN accelerators," in *DATE 2022*, pp. 652–657, IEEE, 2022.

[22] Y. Huang *et al.*, "Accelerating graph convolutional networks using crossbar-based processing-in-memory architectures," in *IEEE HPCA 2022*, pp. 1029–1042, IEEE, 2022.

[23] T. N. Kipf *et al.*, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[24] L. Ge *et al.*, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[25] C. Hong *et al.*, "Efficient sparse-matrix multi-vector product on GPUs," in *HPDC 2018*, pp. 66–79, 2018.

[26] N. Srivastava *et al.*, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *IEEE/ACM MICRO-53*, pp. 766–780, IEEE, 2020.

[27] H. Zeng *et al.*, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," in *ACM/SIGDA FPGA 2020*, pp. 255–265, 2020.

[28] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM Journal of research and Development*, vol. 11, no. 1, pp. 25–33, 1967.

[29] V. Kathail, "Xilinx vitis unified software platform," in *ACM/SIGDA FPGA 2020*, pp. 173–174, 2020.

[30] R. A. Rossi *et al.*, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[32] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on FPGA," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 103–110, IEEE, 2016.

[33] B. Zhang, H. Zeng, and V. Prasanna, "Hardware acceleration of large scale GCN inference," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 61–68, IEEE, 2020.

[34] Y.-C. Lin, B. Zhang, and V. Prasanna, "Hp-gnn: generating high throughput gnn training implementation on cpu-fpga heterogeneous platform," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 123–133, 2022.

[35] A. Rahimi and etc, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 international symposium on low power electronics and design*, pp. 64–69, 2016.

[36] R. Sarkar *et al.*, "Flowgnn: A dataflow architecture for real-time workload-agnostic graph neural network inference," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1099–1112, IEEE, 2023.

[37] T. Geng *et al.*, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 922–936, IEEE, 2020.

[38] M. Yan *et al.*, "HyGCN: A GCN accelerator with hybrid architecture," in *IEEE HPCA 2020*, pp. 15–29, IEEE, 2020.

[39] S. Mondal *et al.*, "GNNIE: GNN inference engine with load-balancing and graph-specific caching," in *DAC*, pp. 565–570, 2022.

[40] P. Yao *et al.*, "A locality-aware energy-efficient accelerator for graph mining applications," in *IEEE/ACM MICRO 2020*, pp. 895–907, IEEE, 2020.

[41] J. Yen *et al.*, "Meeting slos in cross-platform nfv," CoNEXT, 2020.

[42] J. Wang *et al.*, "Quadrant: A cloud-deployable nf virtualization platform," in *13th SoCC*, ACM, 2022.

[43] G. Zhang *et al.*, "Cocktail: Mixing data with different characteristics to reduce read reclaims for nand flash memory," *IEEE-TCAD*, 2022.

[44] F. Wen *et al.*, "An fpga-based hybrid memory emulation system," in *31st FPL*, IEEE, 2021.

[45] F. Wen *et al.*, "Hardware memory management for future mobile hybrid memory systems," *IEEE-TCAD*, vol. 39, no. 11, 2020.

[46] F. Zhang *et al.*, "Max-pim: Fast and efficient max/min searching in dram," in *58th ACM/IEEE Design Automation Conference*, 2021.

[47] C. Yu *et al.*, "A 65-nm 8t sram compute-in-memory macro with column adcs for processing neural networks," *IEEE-JSSC*, 2022.

[48] J. Kang *et al.*, "Relhd: A graph-based learning on fefet with hyperdimensional computing," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pp. 553–560, IEEE, 2022.

[49] N. McDonald, "Modularizing and assembling cognitive map learners via hyperdimensional computing," *arXiv preprint arXiv:2304.04734*, 2023.

[50] H. Chen *et al.*, "Full stack parallel online hyperdimensional regression on fpga," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pp. 517–524, IEEE, 2022.

[51] H. Chen *et al.*, "Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[52] H. Chen and M. Imani, "Density-aware parallel hyperdimensional genome sequence matching," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 1–4, IEEE, 2022.

[53] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 221–234, IEEE, 2021.

[54] S. Duan *et al.*, "A brain-inspired low-dimensional computing classifier for inference on tiny devices," *arXiv preprint arXiv:2203.04894*, 2022.

[55] S. Duan *et al.*, "Lehdc: Learning-based hyperdimensional computing classifier," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1111–1116, 2022.