A Combined Content Addressable Memory and In-Memory Processing Approach for k-Clique Counting Acceleration

Xidi Ma*
Beihang University
Beijing, China

Weichen Zhang*
Beihang University
Beijing, China

Xueyan Wang†
Beihang University
Beijing, China

Tianyang Yu
Nanjing University of Aeronautics and Astronautics
Jiangsu, China

Bi Wu
Nanjing University of Aeronautics and Astronautics
Jiangsu, China

Gang Qu
University of Maryland, College Park
Maryland, USA

Weisheng Zhao
Beihang University
Beijing, China

ABSTRACT

k-Clique counting problem plays an important role in graph mining which has seen a growing number of applications. However, current *k*-Clique counting accelerators cannot meet the performance requirement mainly because they struggle with high data transfer issue incurred by the intensive set intersection operations and the inability of load balancing. In this paper, we propose to solve this problem with a hybrid framework of content addressable memory (CAM) and in-memory processing (PIM). Specifically, we first utilize CAM for binary induced subgraph generation in order to reduce the search space, then we use PIM to implement in-place parallel *k*-Clique counting through iterative Boolean logic "AND"-like operation. To take full advantage of this combined CAM and PIM framework, we develop dynamic task scheduling strategies that can achieve near optimal load balancing among the PIM arrays. Experimental results demonstrate that, compared with state-of-the-art CPU and GPU platforms, our approach achieves speedups of 167.5× and 28.8×, respectively. Meanwhile, the energy efficiency is improved by 788.3× over the GPU baseline.

KEYWORDS

k-Clique Counting, Processing-In-Memory, Content Addressable Memory, Hybrid Framework

1 INTRODUCTION

Graph mining seeks to uncover certain subgraph patterns, among which *k*-Clique counting finds extensive applications across diverse domains, such as community detection [1], biological networks [2], and recommendation systems [3]. A *k*-Clique refers to a graph structure where *k* nodes are all directly connected with each other. Given its importance, the acceleration of *k*-Clique counting has received extensive attention from academic and industrial communities.

Plenty of acceleration methods have been designed on traditional platforms like CPU and GPU [4–6]. Mainstream graph mining algorithms adopt set-centric computational paradigm, which constructs search trees through operations among neighbor sets. However, the computational complexity for *k*-Clique counting using this paradigm grows exponentially with search depth. Induced subgraph generation with graph orienting has been proposed to reduce the search space, while this method involves numerous search and comparison operations, along with frequent memory access. Moreover, graph mining necessitates frequent retrieval operations to identify forthcoming expansion nodes. Existing research has demonstrated that in triangle (3-Clique) counting, the volume of data transferred is two orders of magnitude larger than the size of the original graph [7]. Even worse, in traditional Von-Neumann architecture, the frequent data transfer overhead between computing units and memory has become a major bottleneck regarding latency and power consumption. The execution characteristics of *k*-Clique counting have made the problem even more severe.

Fortunately, Content Addressable Memory (CAM) and Processing-In-Memory (PIM) present promising solutions to the aforementioned challenges. On the one hand, CAM enables high-speed parallel data retrieval based on content. This makes CAM well-suited for the frequent search and comparison tasks during induced subgraph generation. On the other hand, PIM processes data directly in memory, significantly reducing the latency and energy costs caused by data transfer. Therefore, PIM is particularly suitable for addressing memory-intensive tasks, such as *k*-Clique counting. Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM) has emerged as one of the ideal mediums for PIM and CAM implementations, for its characteristics of non-volatility, resistance storage, rapid read-write capabilities, high endurance, and integration density. Therefore, this paper adopts STT-MRAM to implement the proposed strategies for evaluation.

Currently, a series of graph processing systems have been constructed based on PIM [8–10], achieving notable improvements in speed and energy efficiency. However, PIM is currently primarily applied to graph algorithms with shallow search depths, such as triangle counting [10, 11], and it still lacks effective PIM strategies

*Both authors contributed equally to this research.
†Corresponding author, wangxueyan@buaa.edu.cn.

BUAA's work was supported in part by the National Natural Science Foundation of China (NO.92364201 and NO.62004011), the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), and Beihang Frontier Cross-Fund Project. NUAA's work was supported in part by the National Natural Science Foundation of China (NO.92364201).

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656513

for tasks with higher search depths, like k-Clique counting. In addition, as the search depth increases, uneven distributions of graph data cause severe load imbalance. A naive approach to alleviating this problem is to directly divide all tasks into more fine-grained partitions. However, due to the power-law distribution nature of graphs, only a few nodes perform a large amount of computations, thus most nodes do not require fine-grained partitioning. Therefore, the above approach achieves limited performance improvement and incurs significant additional scheduling overhead.

To address these issues, in this paper, we propose a novel combined CAM and PIM approach to realize fast and energy-efficient k-Clique counting through hardware-software co-design. To the best of our knowledge, this is the first hardware accelerator for k-Clique counting with combined CAM and PIM framework. Our contributions are summarized as follows:

- We leverage the rapid retrieval mechanism of CAM for efficient binary-induced subgraph generation, which effectively reduces the search space for k-Clique.
- We propose to perform k-Clique counting on the induced subgraphs through iterative basic boolean logic like "AND", which can be efficiently implemented with PIM arrays in parallel, simultaneously reducing data movement overhead.
- A combined CAM and PIM framework for k-Clique counting, with a dynamic task scheduling strategy that achieves nearly ideal load balancing among PIM arrays with negligible overhead.

The rest of this paper is organized as follows: Section 2 introduces the preliminaries. Section 3 illustrates our proposed approach that includes the combined memory framework and dynamic task scheduling method. Section 4 presents our experimental methodology and results, and Section 5 concludes the paper.

2 PRELIMINARIES

2.1 k-Clique Counting

A k-Clique is defined as a graph structure where all k vertices are connected with each other. Consequently, k-Clique counting aims to identify and enumerate the k-Clique patterns within graphs. Current mainstream graph mining algorithms employ the set-centric paradigm to construct a search tree and then enumerate the leaf nodes. For k-Clique counting, there are currently solutions and corresponding optimization technologies on traditional platforms like CPU and GPU [4–6].

Two optimization strategies are proposed for k-Clique counting: graph orientation and graph pivoting. Graph orientation leverages the symmetrical properties of k-Clique. The key idea is transforming an undirected graph into a directed one while preserving its connectivity. The primary objective of this transformation is to reduce both the out-degree of individual vertices and the overall maximum out-degree of the graph, consequently minimizing the size of the neighboring set and memory consumption during computations. Graph orientation effectively counts cliques and precisely locates the target cliques, demonstrating good scalability. It primarily relies on simple set operations, making it well-suited for hardware-level optimization. Comparatively, graph pivoting searches for the largest clique in the graph to determine the number of contained k-Cliques, thereby reducing the overhead of constructing the search tree. It can only calculate the number of target cliques without pinpointing their exact positions, and it involves complex combinatorial calculations, making it hard to optimize at the hardware level. Therefore, this paper adopts the graph orientation way for optimization.

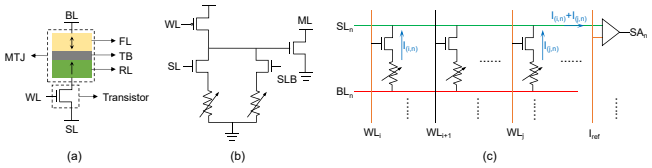

Figure 1: Hybrid memory cell and array designs: (a) STT-MRAM bitcell, (b) CAM bitcell, and (c) PIM array.

Moreover, the above two strategies require a significant amount of time to perform neighbor set operations. A common optimization method for this problem is to use subgraph inducing, which reduces the size of the neighboring set by deleting vertices that will never be reached in each search tree. In this paper, we use the binary adjacency matrix to represent the induced subgraph with reference to [6]. We will provide a detailed description of our approach to accelerate this technique in Section 3.2.

2.2 CAM and PIM based on STT-MRAM

STT-MRAM emerges as a promising non-volatile memory for its low power consumption, fast access speed, and high storage density [12]. Also, it boasts exemplary radiation resistance, robust reliability, and promising scalability. As illustrated in Figure 1(a), a standard STT-MRAM cell is designed to store 1 bit information, consisting of a magnetic tunnel junction (MTJ) paired with a transistor. This setup is controlled by a word line (WL), bit line (BL), and source line (SL). Specifically, the MTJ includes three critical layers: a reference layer (RL), a tunneling barrier, and a free layer (FL). The resistance of the MTJ, which encodes binary data, depends on the alignment of the RL and FL. The WL enables access to specific storage cells, while the combined roles of BL and SL define the voltage difference across the MTJ, enabling read, write, and logical operations.

Content Addressable Memory (CAM) provides inherent parallel searching capability. This allows users to query using the data content itself, swiftly locating its storage address [13]. This unique retrieval mechanism significantly accelerates the data lookup process, making it particularly suitable for scenarios that demand high-speed search capabilities. Figure 1(b) illustrates the structure of a CAM bitcell built upon STT-MRAM, employing two MTJs for information storage. By activating the SL or source line bar (SLB), the corresponding transistor is turned on, allowing data from the WL to be written into the respective MTJ. During search operations, a search signal is generated on the match line (ML). If there is a successful match, the voltage level on the ML remains unchanged; if there is no match, the ML will discharge rapidly.

Bit-wise logic functions are facilitated by simultaneously activating two word lines, WL_i and WL_j. As shown in Figure 1(c), with a reading voltage applied across BL_n and SL_n, the current flow into the n-th sense amplifier (SA) aggregates currents from MTJ nodes $MTJ_{(i,n)}$ and $MTJ_{(j,n)}$. The versatility of this method allows for diverse logic operations based on varied reference sensing currents.

3 COMBINED HYBRID MEMORY APPROACH

To fully exploit the efficient retrieval capability of CAM and leverage PIM to accelerate set intersection and alleviate the memory bottleneck, we design a combined approach that effectively utilizes the strengths of both technologies. In this section, we will outline the overall workflow of our method and provide a detailed explanation of how CAM and PIM are applied to address the k-Clique counting.

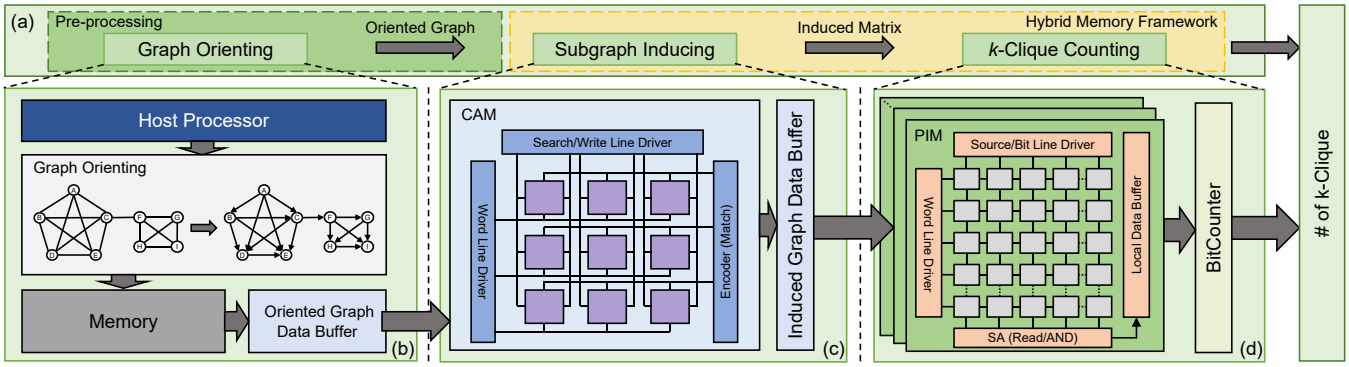

Figure 2: Overview of our approach: (a) overall workflow, (b) graph orienting processing, (c) subgraph inducing with CAM, and (d) k-Clique counting with PIM.

3.1 Overall Workflow

Our proposed combined content addressable memory and in-memory processing approach for k-Clique counting is demonstrated in Figure 2. First, graph orientation is conducted on the undirected graph by the host processor. Then, to reduce the search space, CAM extracts an induced subgraph of a root node and fetches the needed neighbor sets from the oriented graph data buffer. The induced subgraph is then subsequently sent to PIM for k-Clique counting.

3.2 CAM for Binary Subgraph Inducing

Existing CPU and GPU approaches encounter challenges due to the high overhead involved in intensive set intersection when inducing subgraphs. To overcome this issue, we leverage the rapid retrieval mechanism of CAM to perform the induced subgraph generation efficiently. The specific circuit is shown in Figure 2(c).

Given a graph $G(V, E)$, where V denotes the set of vertices and E the set of edges. Suppose the root node is R, its neighbor set is $N(R)$, and the nodes in the neighbor set are represented by v_i, where $v_i \in N(R)$ and $i = 0, 1, 2, ..., n-1$, with n denoting the number of nodes in the neighbor set.

For a root node R, when inducing the subgraph, CAM first loads neighbor set $N(R)$. Next, based on nodes v_i in $N(R)$, CAM fetches the neighbor set $N(v_i)$ from the oriented graph data buffer, and swiftly retrieves if the nodes in $N(v_i)$ exist in $N(R)$. If there is a match, the corresponding position in the induced matrix is set to '1'; otherwise, it is set to '0'. Eventually, the binary adjacency matrix representation of the induced subgraph is produced. Each element represents whether the root node R has a common neighbor with v_i, given by:

$$I_R[i][j] = \begin{cases} 1, & \text{if } v_j \in N(R) \cap N(v_i) \\ 0, & \text{if } v_j \in N(R) - N(v_i) \end{cases} \quad (1)$$

In the induced subgraph I generated with the root vertex R, $I_R[i][j]$ denotes the element at the i-th row and j-th column. After being generated, the induced subgraph is temporarily stored in the induced graph data buffer for further processing.

Figure 3 provides an example to illustrate the process of generating induced subgraphs using CAM. Figure 3(a) displays the example graph after orientation, highlighting the root node A (colored yellow) and the neighbor nodes B to E (colored green). These neighbor nodes, directly connected to the root node A, form a potential subgraph. The nodes F to H are pruned and not included in the subgraph due to their lack of direct connection to A. Figure 3(b) shows the process of induced subgraph generation using CAM. Initially, the root node A and its neighbor set $N(A)$ are stored in

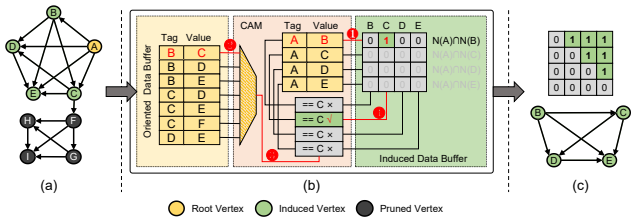

Figure 3: An example of CAM workflow: (a) the example graph after orientation, (b) the process of induced subgraph generation using CAM, and (c) the final induced matrix and corresponding subgraph.

CAM, while the neighbor set $N(v_i)$ for each $v_i \in N(A)$ are kept in the oriented graph data buffer. The induced matrix is initialized to all zero. In step ①, CAM uses the tag label to identify A as the root node of the induced subgraph. The value label field is the neighbor of A and also the column index of the induced binary matrix. For example, the tag A and value B refer to the first line of the matrix. In step ②, the value B currently in CAM is used to locate neighbor set $N(B)$ in the buffer. In step ③, a node instance C from $N(B)$ in the buffer is imported into CAM for a match check. In step ④, upon successfully retrieving node C, CAM activates the corresponding column in the induced matrix, assigning a value to that matrix element. This iterative process results in the final induced matrix and corresponding subgraph shown in Figure 3(c). The methodology effectively exploits the parallel search capability of CAM, significantly enhancing subgraph generation efficiency.

3.3 PIM for k-Clique Counting

In this section, we describe our approach of utilizing binary induced subgraph and the AND operation of PIM to traverse all the search trees and count all the k-Cliques.

Once receiving the induced subgraph matrix generated by CAM from the induced graph data buffer, PIM arrays start to process the matrix based on the user-specified value of k. Since the induced matrix is in binary form, it can be calculated directly using logical operations. Each row of the induced matrix can be interpreted as the intersection result of the neighbor set of root node R and the corresponding node v_i in $N(R)$, denoted as:

$$I_R[i][*] = N(R) \cap N(v_i) \quad (2)$$

Performing row-wise AND operations in PIM has the following implications:

$$AND(I_R[i][*], I_R[j][*]) = N(R) \cap N(v_i) \cap N(v_j) \quad (3)$$

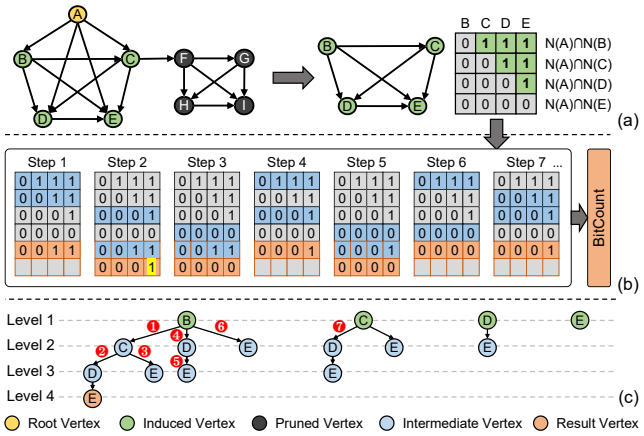

Figure 4: An example of PIM workflow: (a) the induced subgraph for the root vertex A, (b) the computational flow within the PIM array, and (c) the search trees generated based on the induced subgraph.

This implies that k-Clique counting can be parallelized on a large scale directly in memory using binary subgraph. The formal description of k-Clique counting in PIM is as follows:

$$CC^{(k)}(G) = \sum_{v \in V} BitCount\left(RowAND^{(k-3)}(I(v))\right) \quad (4)$$

Here, $CC^{(k)}(G)$ denotes the result of k-Clique counting on the graph G. $I(v)$ refers to the induced subgraph of node v. $RowAND^{(k-3)}$ means the nested $k-3$ level loop for row-wise `AND` operation and the final result vectors (generated in the last line of the result array) contains the k-Clique numbers based on v. `BitCount` is used to count the valid values in the result vectors generated after the `RowAND` operation.

Figure 4 gives an example to illustrate the workflow of a 5-Clique counting in PIM. Figure 4(a) displays the induced subgraph matrix generated in the previous section. This subgraph is stored in the computing array of PIM (the first four rows, indicated in gray when not selected, while the last two rows are spared for storing the intermediate and final result value). Figure 4(b) shows the computational flow of the induced matrix within the PIM array. Figure 4(c) presents the final search trees generated based on this induced subgraph during computation.

The computational process is depicted from step 1 to step 7. As shown in figure 4(b), the rows in blue are selected for row-wise `AND` operation and the row in red is for storing the above computing results. In step 1, PIM starts the search tree at the node B, and the corresponding PIM operation is `AND` between row 0 and row 1, while the temporary result is stored in row 5. In step 2, PIM uses this temporary vector to further explore the search tree and find node D in level 3 (as shown in Figure 4(c), the same as below). This time PIM computes row 2 and row 4 and saves the result in row 5. The number of valid bits in row 5 indicates the valid value of 5-Clique during this search, and this line will be later sent to `BitCounter` for counting. In step 3, after reaching the bottom of the search tree, PIM returns to level 3 to explore other branches and perform `AND` operation between row 3 and row 4, still based on the value previously calculated. As the result is all-zero this time, it indicates that this branch is pruned at this level. For the remained steps, PIM follows the rule of depth-first search to explore the whole tree.

3.4 Dynamic Hardware Task Scheduling

Due to the significant difference among the search trees depth and width of real-world graph datasets directly assigning the tasks based on root node to a single PIM array(vertex-centric strategy) leads to severe load imbalance and wastage of hardware resources. Edge-centric strategy is one of the basic solutions to alleviate the load imbalance problem. It breaks the tasks into finer-grained sub-tasks based on edge and allocates them across multiple PIM arrays for more parallelism. However, the imbalance issue of this method tends to resurface as the hardware parallelism increases. Therefore, excessive subdivision is unnecessary, which just provides limited performance improvement but leads to additional scheduling overhead.

To address the challenges above, our work introduces a dynamic hardware task scheduling mechanism. Specifically, after tasks are distributed based on root nodes, the system actively monitors the utilization rate of the PIM arrays during computation. When the utilization falls below a certain threshold, the system fetches the working status stack (further described in Section 3.5) from the running PIM array, divides its task into sub-tasks at the next level based on the current status and dispatches the sub-tasks across available units. We observe that a limited number of nodes contribute to the majority of the running time in node-centric strategies. Therefore, our proposed scheduling approach just needs to partition a small number of large tasks, resulting in significantly improved workload balance and keeping scheduling overhead to a minimum. Moreover, this strategy can dynamically divide sub-tasks at arbitrary levels of the search tree to solve the reoccurred load imbalance of the edge-centric strategy.

3.5 Pseudocodes for Our Combined Approach

Algorithm 1 outlines the hardware workflow of CAM in generating an induced subgraph. `CAM.search` refers to the retrieving function of CAM and then returns two signals *hit* and *addr* (line 6), which can be directly written into the data buffer. The *hit* signal indicates whether the searched content is present in the CAM, while *addr* represents the address of the content if it is found in the CAM.

Algorithm 1: CAM-based Subgraph Inducing.

Input: Root vertex R.
Output: Binary induced subgraph adjacency matrix I_R.

```
1 CAM.init(DataBuffer.read_neigh(R));
2 for each vertex v_i in CAM do
3     Buffer.init();
4     N(v_i) = DataBuffer.read_neigh(v_i);
5     for each vertex v_j in N(v_i) do
6         hit, addr = CAM.search(v_j);
7         Buffer[addr] = hit;
8     Controller.send(Buffer);
```

When the binary matrix is induced, it is sent to the available PIM, which works as Algorithm 2 shows. $offset$ (line 2) indicates that the last $k-3$ lines of memory space are reserved in PIM for storing temporary values, which can be adjusted based on the scale of k. In order to enable each PIM array to work independently and handle clique counting of arbitrary k, we use a stack to illustrate

the working status of PIM. This stack is managed and updated in the `PIM.Status.update` function. In addition, `PIM.cal_write` performs the row-wise AND operation between two selected lines and writes the result to the corresponding PIM line (lines 14, 20).

Algorithm 2: PIM-based k-Clique Counting.

Input: Induced Subgraph I_R, k.
Output: Numbers of k-Clique counting.

```
1  CC = 0;
2  offset = array_lines − (k − 3);
3  level = 1;
4  PIM.init(I_R);
5  stack.init();
6  while level > 0 do
7      if level == 1 then
8          continue;
9      else if level == 2 then
10         line1 = stack[level − 1];
11         line2 = stack[level];
12         if arrays[line1][line2] == 0 then
13             continue;
14         PIM.cal_write(line1, line2, offset);
15     else if level < k − 3 then
16         line1 = stack[level];
17         line2 = offset + level − 3;
18         if arrays[line2][line1] == 0 then
19             continue;
20         PIM.cal_write(line1, line2, line2 + 1);
21     if level == k − 3 then
22         Stores the last line of PIM arrays as result_vectors;
23     PIM.Status.update();
24 CC+ = BitCount(result_vectors);
25 return CC ;
26 ——————————————————————
27 Function PIM.Status.update()
28     if stack[level] ≥ size then
29         level--;
30     else
31         if level == k − 3 then
32             stack[level]++;
33         else
34             stack[++level] = 0;
```

4 EVALUATION

4.1 Experimental Setup

To rigorously substantiate the efficacy of our proposed methodologies, we embarked on a comprehensive suite of evaluations, employing a diversified set of simulation tools tailored for varying assessment requirements. At the device level, the behavior of the MTJ is delineated based on the Brinkman model and the Landau-Lifshitz-Gilbert (LLG) equation [14]. Transitioning to the circuit layer, we harnessed the Cadence suite for meticulous circuit-level simulations of PIM and CAM, ensuring obtain accurate circuit characteristic data. To gain a deeper understanding of the operational intricacies of the CAM and PIM arrays, we developed a dedicated behavioral-level simulator, crafted in C++. To ensure the universality and authenticity of the evaluation, we selected undirected graph datasets (as shown in Table 2) from SNAP [15] to serve as our benchmark. The datasets are preprocessed using graph orienting by the open source code from [4]. Lastly, we evaluated our method in detail against contemporary state-of-the-art CPU [4] and GPU [6] solutions. The configuration of previous work and our approach is shown in Table 1.

Table 1: Configuration of previous work and our approach

	CPU[4]	GPU[6]	Our Work
Computing Unit	80 Intel Xeon E7 cores	5120 NVIDIA Volta V100 cores	16 CAM PEs & 128 PIM PEs
Memory	3844 GB DRAM	32 GB HBM2	64 KB CAM & 16 MB PIM

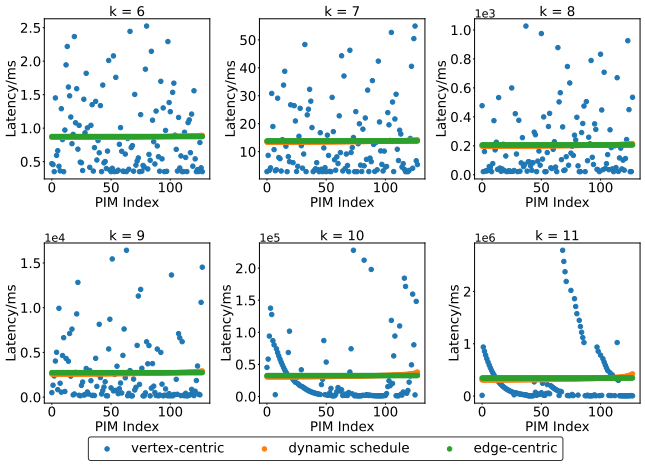

Figure 5: Comparison with vertex-centric, dynamic schedule, and edge-centric on the com-dblp dataset.

4.2 Performance Results

Table 2 presents our performance results, we conducted k-Clique counting on various graph datasets with k ranging from 4 to 11, where the missing data points in the com-LiveJournal dataset mean running times above 5 hours on all existing methods. Compared to the configurations of current state-of-the-art CPU and GPU solutions as shown in Table 1, our system utilizes only 64KB CAM and 16MB PIM. Despite significantly lower usage of computing units and memory than the existing best solutions, our approach, benefiting from the employed combined memory architecture and dynamic task scheduling method, still achieves substantial performance improvements. As shown in Table 2, our research results indicate that compared to the state-of-the-art CPU and GPU solutions, our approach attains acceleration ratios of 167.5× and 28.8×, respectively.

4.3 Benefit of Dynamic Task Scheduling

Figure 5 demonstrates a comparison among three different scheduling strategies: vertex-centric, dynamic scheduling, and edge-centric. The scatter plots illustrate the actual running time of each PIM in the high k numbers (more than 6). The cases with k less than 6 are not included because of the space limitation, while they exhibit the same trend.

Table 2: Result of runtime (Second)

Graph Datasets	#Vertices	#Edges	Baseline & our work	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	$k=11$
as-skitter	1696415	11095298	CPU[4]	0.60	0.67	1.24	5.73	28.38	158.45	854.87	4158.53
			GPU[6]	0.03	0.07	0.25	1.43	9.53	68.22	474.79	2997.39
			Our Work	**4.89E-03**	**5.39E-03**	**9.69E-03**	**4.62E-02**	**0.34**	**2.44**	**16.00**	**91.27**
com-dblp	317080	1049866	CPU[4]	0.10	0.13	0.30	2.05	24.06	281.39	2981.74	>5 hours
			GPU[6]	8.00E-03	1.60E-02	0.04	0.55	9.03	139.05	2262.99	>5 hours
			Our Work	**1.81E-04**	**2.13E-04**	**9.93E-04**	**1.42E-02**	**0.22**	**2.92**	**38.51**	**424.06**
com-orkut	3072441	117185083	CPU[4]	3.10	4.94	12.57	42.09	150.87	584.39	2315.89	8843.51
			GPU[6]	0.43	1.01	3.51	11.72	45.32	212.91	1002.17	4421.60
			Our Work	**0.17**	**0.19**	**0.26**	**0.63**	**2.34**	**10.39**	**47.56**	**211.36**
com-LiveJournal	3997962	34681189	CPU[4]	1.77	7.52	258.46	10733.21	>5 hours	-	-	-
			GPU[6]	0.10	0.94	23.79	1077.66	>5 hours	-	-	-
			Our Work	**2.11E-02**	**3.50E-02**	**0.67**	**30.22**	**1250.00**	-	-	-

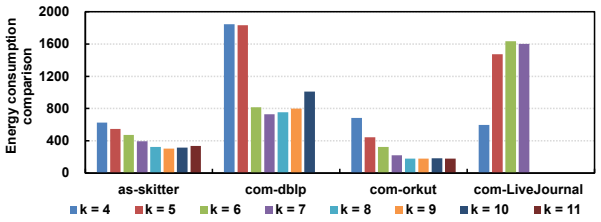

Figure 6: Energy consumption for our work compared to the GPU baseline.

In Figure 5, we can observe that the vertex-centric strategy shows the most serious load imbalance phenomenon especially when the number of k increases. In the later stage of computing, a substantial amount of computational resources are idle, leaving only a small number of PIM arrays performing calculations. The edge-centric method efficiently alleviates this problem, but it needs to divide all the search trees of the graph which significantly increases the overhead of scheduling. Our proposed dynamic task scheduling method achieves practically the extremely close effect as the edge-centric strategy, while only partitioning 0.02% of tasks, thus greatly reducing scheduling overhead.

4.4 Energy Consumption Analysis

In terms of energy consumption, Figure 6 presents the energy consumption ratio of the GPU baseline compared to our work in different k across four distinct graph datasets. By effectively leveraging the non-volatile nature of STT-MRAM and its in-memory-computing capability, our method significantly enhances energy efficiency. Compared to the GPU baseline, our approach on average achieves a reduction in energy consumption by 788.3×.

5 CONCLUSION

In this paper, we propose a novel combined memory approach integrating CAM and PIM for efficient k-Clique counting. Our method employs CAM to generate binary-induced subgraphs, effectively reducing the search space, which is crucial for reducing the computational complexity in large graphs. Additionally, the integration of PIM facilitates parallel k-Clique counting within the binary induced matrixes using basic boolean logic, greatly enhancing processing efficiency. Compared with the state-of-the-art CPU and GPU solutions, experimental results show that our approach demonstrates substantial improvements of 167.5× and 28.8× speedups, respectively, while reducing energy consumption by 788.3× compared to the GPU baseline.

REFERENCES

[1] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks VS Lakshmanan, and Xuemin Lin. Efficient algorithms for densest subgraph discovery. *Proceedings of the VLDB Endowment*, 12(11):1719–1732, 2019.

[2] Balázs Adamcsek, Gergely Palla, Illés J Farkas, Imre Derényi, and Tamás Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006.

[3] Dr Samuel Manoharan and Ammayappan Sathesh. Patient diet recommendation system using k clique and deep learning classifiers. *Journal of Artificial Intelligence and Capsule Networks*, 2(2):121–130, 2020.

[4] Jessica Shi, Laxman Dhulipala, and Julian Shun. Parallel clique counting and peeling algorithms. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 135–146. SIAM, 2021.

[5] Shweta Jain and C Seshadhri. The power of pivoting for exact clique counting. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 268–276, 2020.

[6] Mohammad Almasri, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. Parallel k-clique counting on gpus. In *Proceedings of the 36th ACM International Conference on Supercomputing*, pages 1–14, 2022.

[7] Guohao Dai, Zhenhua Zhu, Tianyu Fu, Chiyue Wei, Bangyan Wang, Xiangyu Li, Yuan Xie, Huazhong Yang, and Yu Wang. Dimmining: pruning-efficient and parallel graph mining on near-memory-computing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 130–145, 2022.

[8] Yuntao Wei, Xueyan Wang, Shangtong Zhang, Jianlei Yang, Xiaotao Jia, Zhaohao Wang, Gang Qu, and Weisheng Zhao. Imga: Efficient in-memory graph convolution network aggregation with data flow optimizations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[9] Xuhang Chen, Xueyan Wang, Xiaotao Jia, Jianlei Yang, Gang Qu, and Weisheng Zhao. Accelerating graph-connected component computation with emerging processing-in-memory architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(12):5333–5342, 2022.

[10] Xueyan Wang, Jianlei Yang, Yinglin Zhao, Yingjie Qi, Meichen Liu, Xingzhou Cheng, Xiaotao Jia, Xiaoming Chen, Gang Qu, and Weisheng Zhao. Tcim: triangle counting acceleration with processing-in-mram architecture. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[11] Tianyu Fu, Chiyue Wei, Zhenhua Zhu, Shang Yang, Zhongming Yu, Guohao Dai, Huazhong Yang, and Yu Wang. Clap: Locality aware and parallel triangle counting with content addressable memory. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[12] Yinglin Zhao, Jianlei Yang, Bing Li, Xingzhou Cheng, Xucheng Ye, Xueyan Wang, Xiaotao Jia, Zhaohao Wang, Youguang Zhang, and Weisheng Zhao. Nand-spin-based processing-in-mram architecture for convolutional neural network acceleration. *Science China Information Sciences*, 66(4):142401, 2023.

[13] Didi Zhang, Bi Wu, Haonan Zhu, and Weiqiang Liu. Capacity-oriented high-performance nv-tcam leveraging hybrid mram scheme. In *Proceedings of the 17th ACM International Symposium on Nanoscale Architectures*, pages 1–6, 2022.

[14] Hao Cai, You Wang, Lirida Alves De Barros Naviner, and Weisheng Zhao. Robust ultra-low power non-volatile logic-in-memory circuits in fd-soi technology. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(4):847–857, 2016.

[15] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.