

PRISM: A Locality-Aware Near-Memory Processing Framework for Scalable Triangle Counting

Shangdong Zhang*, Xueyan Wang*[†] and Yier Jin[‡]

*School of Integrated Circuit Science and Engineering, Beihang University

[†]Qingdao Research Institute, Beihang University

[‡]University of Science and Technology of China, Hefei, China

Abstract—Triangle Counting (TC) is a fundamental yet expensive graph algorithm. On conventional platforms, its performance is fundamentally limited by the high cost of data movement between processors and memory. Near-Memory Processing (NMP) has emerged to alleviate this issue; however, its efficacy is often compromised by poor data locality, significant set intersection overhead, and prohibitive inter-NMP communication costs when applied to large-scale graphs.

To address these challenges, we propose PRISM, a hardware-software co-design framework based on a connectivity-aware graph partitioning strategy. PRISM provides a unified solution that incorporates three key components: a locality-aware algorithm, a heterogeneous processing engine, and a scalable replication mechanism. Specifically, PRISM (1) improves data locality by employing distinct counting methods for partitioned hub and non-hub regions; (2) reduces set intersection overhead through a hybrid engine combining bitmap and content-addressable memory (CAM); and (3) alleviates communication bottlenecks in large graphs by replicating only a small yet critical hub-subgraph. Evaluations on eight real-world datasets demonstrate that PRISM reduces DRAM access volume by 39.49% and achieves an average speedup of 2.05 \times compared to the state-of-the-art solution.

Index Terms—Triangle Counting, Near-Memory Processing, Hardware-Software Co-design, Graph Partitioning.

I. INTRODUCTION

Triangle Counting (TC) is a fundamental graph analysis tool, which plays a crucial role in fields like social network analysis [1] and serves as a key subroutine for complex graph mining [2]. On conventional processor-centric platforms such as CPUs and GPUs, the performance of TC is hampered by massive data movement, stemming from the need to frequently intersect adjacency lists via irregular memory accesses [3]–[6].

Near-Memory Processing (NMP) has emerged as a promising approach to mitigate data movement overheads [7]–[15]. By placing processing elements (PEs) near memory, NMP reduces data transfer latency. However, its effectiveness for TC is limited by three key challenges, as illustrated in Fig. 1: poor data locality, significant set intersection overhead, and prohibitive inter-NMP communication costs when handling large graphs.

The performance of TC on NMP is fundamentally undermined by poor data locality. The issue stems from intersecting adjacency lists stored in formats such as Compressed Sparse Row (CSR) or its variants. Existing NMP accelerators [7]–[9],

Corresponding authors: Xueyan Wang (wangxueyan@buaa.edu.cn), Yier Jin (jinyier@ustc.edu.cn). This work was supported in part by the National Natural Science Foundation of China (No. 62474016 and No. 92364201).

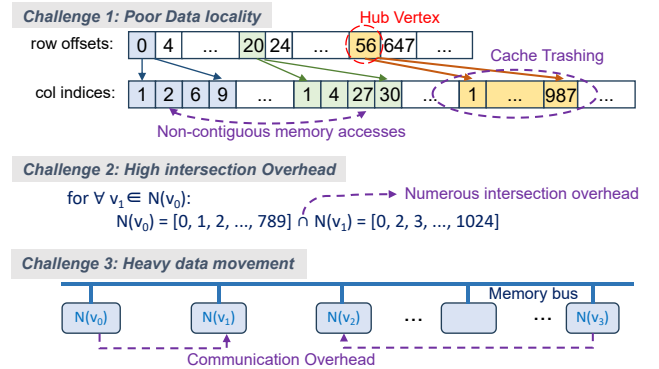


Fig. 1. Challenges in triangle counting applications.

[11], [13]–[17] typically apply a uniform intersection strategy that ignores the distinction between sparse non-hub regions (areas of low-degree vertices) and the dense hub regions (areas of highly connected vertices) of the graph. This “one-size-fits-all” approach leads to two major locality bottlenecks: first, frequently fetching non-contiguous adjacency lists degrades spatial locality, and second, processing the long adjacency lists of hub vertices overwhelms on-chip caches, diminishing temporal locality.

This access inefficiency is compounded by the high cost of set intersection, which accounts for over 90% of total computation time [7]. Early NMP accelerators [7], [11], [16] typically employed a merge-based intersection method, which suffers from limited parallelism and high computational complexity. More recent designs [8], [14], [17] leverage Content-Addressable Memory (CAM) to improve parallelism and reduce complexity to $\mathcal{O}(n)$. However, CAM-based approaches face a critical scalability issue: the limited CAM capacity is easily exceeded by the long adjacency lists of hub vertices, restricting their practicality for real-world graphs.

Furthermore, these locality and computational pressures converge into a system-level communication bottleneck when scaling to large graphs. Prior studies [7], [8], [14] avoid this issue for small graphs through full graph replication. However, this approach becomes unfeasible for large graphs, as their size quickly exceeds the memory capacity of a single NMP module. This limitation forces a resort to graph partitioning—a common strategy exemplified by works like ProMiner [9]—which in turn reintroduces prohibitive inter-NMP data movement as a major bottleneck. This communication overhead is particularly exacerbated by hub vertices, whose numerous cross-partition

edges can saturate the network [15].

To address these challenges, we propose PRISM, a hardware-software co-design framework. The cornerstone of PRISM is a connectivity-aware partitioning strategy that separates the graph into dense hub regions and sparse non-hub regions. This partitioning enables three tailored optimizations: a locality-aware algorithm that applies region-specific counting methods, a heterogeneous processing engine for efficient set intersection, and a scalable replication mechanism to minimize communication. Our main contributions are as follows:

- A locality-aware TC algorithm that improves data locality by applying hub-pair checking to dense hub regions and separate set intersections to sparse non-hub regions, thereby enhancing cache efficiency and reducing random memory accesses.
- A heterogeneous processing engine that reduces intersection overhead through direct bitmap lookups for dense hub regions and a dual CAM-based engine for sparse non-hub regions. This design also alleviates the capacity limitations of prior CAM-only approaches.
- A scalable replication strategy that reduces inter-rank communication by replicating only a compact hub-subgraph while partitioning the rest of the graph, enabling efficient processing of large graphs where existing methods fail.

The remainder of this paper is organized as follows. Section II reviews background concepts in triangle counting. Section III presents our locality-aware TC algorithm based on connectivity-aware partitioning. Section IV details the co-designed NMP architecture. Section V demonstrates the experimental results, and Section VI concludes.

II. BACKGROUND

A. Terminology and Triangle Counting

The modern TC procedure, illustrated in Fig. 2, finds triangles by computing the set intersection between the neighbor lists $N(v_0)$ and $N(v_1)$ for each edge (v_0, v_1) . To prevent double counting, modern approaches [8], [14], [17] often utilize graph orientation, where edges are first directed from higher- to lower-indexed vertices. Following this convention, we use $N(v)$ to denote the lower-indexed neighbor set, $\{u \in N(v) \mid h_u < h_v\}$, for the remainder of this paper. The size of this set, $|N(v)|$, is denoted as $d_L(v)$. A graph's density is defined as the ratio of its edges to its vertices ($|E|/|V|$).

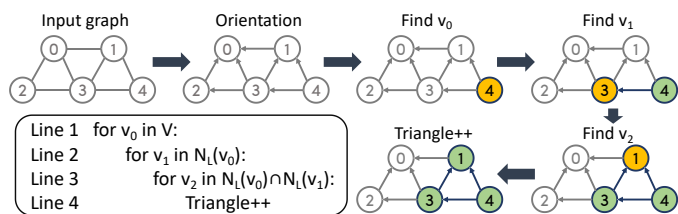


Fig. 2. Triangle counting procedure.

Due to the power-law nature of real-world graphs, partitioning vertices into hubs and non-hubs based on a degree

threshold (referred to as `hub_threshold`) is a common and insightful approach. This classification categorizes triangles into four types based on the number of hub vertices [18]: **HHH** (3 hubs), **HHN** (2 hubs), **HNN** (1 hub), and **NNN** (0 hubs). A hub triangle refers to any triangle including at least one hub vertex (i.e., HHH, HHN, or HNN).

B. Near-Memory Processing

Near-Memory Processing (NMP) mitigates the memory bottleneck by moving logic near data, e.g., near DRAM banks [19]–[21], into 3D-stacked logic dies [16], [22], [23], or on DIMM buffer chips [24]–[27]. Leveraging the vast internal bandwidth significantly reduces energy-intensive data movement. In this work, we focus on a buffer chip-based NMP platform as it offers a practical balance of high parallelism, large capacity support, and wide process compatibility.

C. Content Addressable Memory

Content-Addressable Memory (CAM) is a specialized hardware that inverts the conventional memory paradigm, searching its entire contents in parallel to find a given data word in a single clock cycle [28]. CLAP [8] pioneered the use of CAM to accelerate set intersection in TC. The method stores one set (size m) as CAM entries and sequentially searches for elements of the second set (size n). This parallel search capability reduces the time complexity to $\mathcal{O}(n)$, providing an asymptotic advantage over software algorithms like binary search $\mathcal{O}(n \log m)$ [29] and merge-based intersection $\mathcal{O}(m + n)$ [4].

III. LOCALITY-AWARE TRIANGLE COUNTING

In this section, we present our locality-aware TC algorithm, which provides the algorithmic foundation for the NMP accelerator detailed in Section IV. This algorithm is built upon a connectivity-aware partitioning strategy. We introduce this partitioning strategy first.

A. Connectivity-Aware Partitioning Strategy

Our work employs a connectivity-aware partitioning strategy, inspired by Lotus [18], to create a specialized Connectivity-Aware Storage (CAS) format. This strategy first sorts vertices by their degree and then designates the top `hub_threshold` (default=1%) as hubs. Subsequently, after globally orienting all edges from higher- to lower-index vertices, it segregates them into three specialized structures: (1) a compact Hub-to-Hub (H2H) triangular bitmap for edges connecting hub pairs; (2) a Hub Edges (HE) subgraph in CSR representing the hub neighbors for all vertices; and (3) a Non-Hub Edges (NHE) subgraph in CSR representing the non-hub neighbors for all vertices.

This partitioning is motivated by the power-law degree distribution of real-world graphs, where a small number of highly-connected hub vertices dominate the graph's structure. Our analysis across eight datasets quantitatively confirms this structural dichotomy. As shown in Table I, on average, the H2H subgraph is over $1000\times$ denser than the full graph, and hub-involved triangles constitute a commanding 75.9% of the total workload. This evidence strongly validates isolating the graph's dense hub regions from its sparse non-hub regions to enable partition-specific processing.

TABLE I
ANALYSIS OF TRIANGLE DISTRIBUTION AND HUB SUBGRAPH DENSITY IN
VARIOUS DATASETS

Dataset	Hub Triangles (%)				NNN (%)	H2H Density /Full Graph
	HHH	HHN	HNN	Total		
Email-Enron (EE)	14.7	34.3	32.5	81.5	18.5	602.7
Astro (AS)	1.5	9.0	24.6	35.1	64.9	155.4
Mico (MC)	59.4	21.9	10.1	91.4	8.6	1092.9
Youtube (YT)	49.9	38.5	7.6	95.9	4.1	1129.4
LiveJournal (LJ)	46.6	12.1	9.8	68.5	31.5	727.9
Orkut (OK)	15.8	14.2	9.1	39.1	60.9	453.2
Wiki (WK)	25.0	67.2	7.7	99.9	0.1	1994.9
Twitter (TT)	56.3	29.7	9.7	95.7	4.3	2302.8
Average	33.7	28.4	13.9	75.9	24.1	1057.4

B. Locality-Aware Triangle Counting

While this partitioning provides an excellent foundation, its purely software-based algorithm [18] is not directly applicable to the NMP paradigm. The reliance on merge-based set intersection creates a critical performance bottleneck, as its high computational cost and poor data locality result in excessive data movement. To overcome these bottlenecks, our algorithm is structured in three distinct parts as shown in Algorithm 1, each applying a specialized counting technique.

Algorithm 1 Locality-Aware Triangle Counting

```

1: Input: Graph  $G(hub\_threshold, H2H, HE, NHE)$ 
2: Output: Total number of triangles
3:  $\Delta \leftarrow 0$ 
   // Counting HHH and HHN triangles using bitmap
4: for all  $u \in G.V$  do
5:   for all  $u_1 \in HE.N(u)$  do
6:     for  $u_2 \in \{v \in HE.N(u) \mid v < u_1\}$  do
7:        $\Delta += H2H[h_1, h_2]$ 
   // Counting NNN triangles using NH-CAM array
8: for all  $u \in NHE.V$  do
9:   for all  $v \in NHE.N(u)$  do
10:     $\Delta += |NHE.N(v) \cap_{NH-CAM} NHE.N(u)|$ 
   // Counting HNN triangles using H-CAM array
11: for all  $u \in NHE.V$  do
12:   for all  $v \in NHE.N(u)$  do
13:     $\Delta += |HE.N(v) \cap_{H-CAM} HE.N(u)|$ 
14: return  $\Delta$ 

```

HHH and HHN Counting via Hub-Pair Check. Our algorithm begins by enumerating HHH and HHN triangles (Lines 4-7, Alg. 1). It replaces costly set intersections with hub-pair checks in the dense H2H bitmap. For a given root vertex u , the algorithm iterates through pairs of its hub neighbors (u_1, u_2) and verifies the existence of the closing edge (u_1, u_2) by checking for a ‘1’ in the H2H bitmap. With graph orientation ensuring $h_1 > h_2$ (where h_1 and h_2 are the indices of u_1 and u_2), the specific index for this lookup is calculated as $\frac{h_1(h_1-1)}{2} + h_2$. Thus, the search for a triangle, which would otherwise require a costly set intersection, is transformed into a single, direct address calculation and a bitwise check.

This hub-pair check approach provides a significant locality advantage due to the exceptional information density of the H2H bitmap. While the CSR format requires 4-8 bytes to represent an edge, the H2H bitmap uses just a single bit. Con-

sequently, a single 64-byte cache line can hold the connectivity data for 512 potential hub-pairs. This allows hundreds of edge lookups to be resolved with a single DRAM access, drastically reducing memory traffic and improving cache utilization.

NNN and HNN Counting via CAM-based Intersection. The hub-pair check method, while effective for the dense H2H subgraph, is inapplicable to the sparse connections involving at least two non-hub vertices. Our algorithm therefore reverts to set intersection for NNN and HNN triangles (Lines 8-13, Alg. 1). We propose a novel dual CAM-based method inspired by CLAP [8] to accelerate these costly operations. This design features two specialized engines: an NH-CAM for non-hub neighbor sets ($|NHE.N(v) \cap NHE.N(u)|$) and an H-CAM for hub neighbor sets ($|HE.N(v) \cap HE.N(u)|$).

The tight coupling of this dual-CAM design with the segregated HE and NHE data structures yields a two-fold benefit. First, it significantly enhances data locality. By operating only on the relevant NHE or HE neighbor list, each intersection has a smaller working set, which prevents irrelevant data in one structure from evicting useful data in the other. Furthermore, this segregated approach ensures that each intersection processes only hub neighbors or only non-hub neighbors, rather than the mixed neighbor list of a hub vertex, fundamentally mitigating the severe cache thrashing pressure caused by accessing the complete neighbor list of any hub vertex.

Second, it mitigates the scalability challenge in prior CAM-based accelerators [8], [14], [17]. In their designs, the required CAM capacity is dictated by the largest adjacency list in the graph. Our segregated approach breaks this dependency. By partitioning a vertex’s neighbors into two smaller, distinct lists (HE and NHE) and processing them in separate, specialized CAMs, we ensure that each CAM only needs to store a subset of any vertex’s neighbors, rather than the entire combined list. This dramatically reduces the peak CAM entry requirement, overcoming a key scalability limitation of the monolithic CAM approach. This enhanced locality, combined with CAM’s inherent parallelism, enables sustained high throughput. The detail workflow will be introduced in Section IV-B.

IV. ARCHITECTURE DESIGN

This section presents PRISM, the NMP architecture designed to execute our locality-aware TC algorithm. We first present the architecture overview and its tailored workload distribution scheme. Next, we describe the PE microarchitecture and its specialized workflow for the bitmap and CAM-based counting stages. Finally, we present our hub-subgraph replication strategy that mitigates inter-rank data movement in large-scale graphs.

A. Architecture Overview

PRISM’s architecture overview is shown in Fig. 3(a) and (b). PRISM adopts a similar multi-rank NMP configuration to previous work [7], [8]. We use M PEs to construct a processing unit (PU) on the buffer chip of each DRAM rank. A memory controller handles data requests. N PUs and DRAM ranks form the system.

As illustrated in Fig. 3(c), the microarchitecture of each PE is a specialized, heterogeneous engine designed to execute

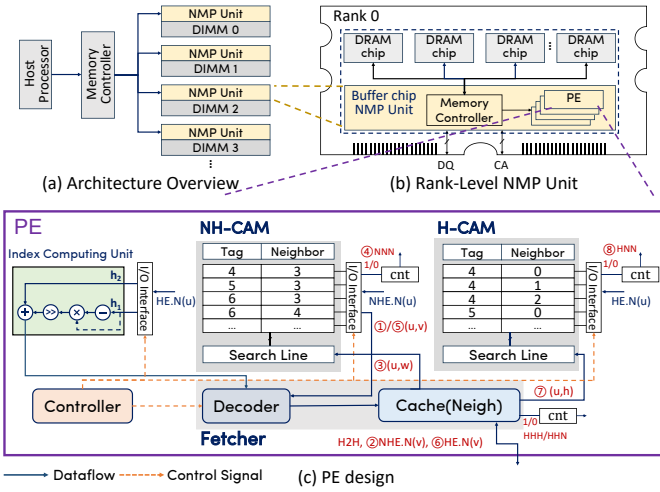


Fig. 3. The PRISM architecture: (a) architecture overview, (b) modified DIMM with NMP unit, (c) the workflow within each PE.

our locality-aware TC algorithm. It comprises three distinct computational units: an Index Computing Unit (ICU) for bitmap address generation, alongside dedicated NH-CAM and H-CAM engines for set intersection. The operation of these units is orchestrated by a central Controller and supplied with data by a Fetcher module, which contains a Decoder and an on-chip neighbor Cache. This partitioned design allows each PE to efficiently execute the different stages of our algorithm, the specifics of which are detailed in Section IV-B.

Workload Distribution. Given our partitioning of vertices into hubs and non-hubs, we apply different distribution strategies to ensure load balance. Specifically, the computations originating from hub vertices are distributed globally across the N PUs using a simple round-robin assignment, where a hub vertex v_h is assigned to the $(h_{v_h} \pmod{N})$ -th PU. In contrast, the workload for non-hub vertices is first grouped into batches to accommodate finite on-chip CAM resources. A batch is formed such that the cumulative size of the neighbor sets for all its constituent vertices does not exceed CAM capacity.

Specifically, a batch is formed when either the sum of all $|NHE.N(u)|$ exceeds the NH-CAM capacity or the sum of all $|HE.N(u)|$ exceeds the H-CAM capacity. These pre-formed batches are then distributed across the N PUs. Since the number of PUs is typically small compared to the vast number of resulting batches, this distribution is empirically sufficient to achieve effective load balance. Crucially, since the workload for each PU is independent, all PUs can execute their assigned tasks in parallel after this distribution.

B. PE Workflow

Once a PU receives its assigned workload, it distributes the tasks among its constituent PEs in a round-robin manner. Each PE then executes a three-stage sequential workflow, as illustrated in Fig. 3(c), which is managed by its Controller to ensure orderly access to the shared on-chip Cache.

Stage 1: HHH/HHN Counting via Bitmap. In the first stage, the PE processes each assigned root vertex u (both hub and non-hub batches) by first fetching its hub neighbor list,

$HE.N(u)$, from local DRAM. This list is streamed to the ICU, which generates addresses for each unique pair of hub neighbors (h_1, h_2) . These addresses, representing requests for specific bits in the H2H bitmap, are serviced by the Decoder. The Decoder then retrieves the corresponding bitmap data from the Cache. A direct lookup on the fetched data then determines if a triangle exists, incrementing the HHH (root u is hub) or HHN (root u is non-hub) counter on a match.

Stage 2: NNN Counting via NH-CAM. After the bitmap stage, the PE begins to process the non-hub root vertices in its assigned workload in batches. For a given batch, the PE first populates the NH-CAM. It iterates through each root vertex u and pre-loads the NH-CAM by concatenating u with each neighbor $v \in NHE.N(u)$ to form entries (u, v) .

With the NH-CAM loaded, the PE executes the following micro-architectural pipeline: ① The NH-CAM acts as a regular memory, sequentially reading out an entry (u, v) , which establishes the base edge. ② The neighbor v is passed to the Decoder, which fetches its non-hub adjacency list, $NHE.N(v)$ from the Cache. ③ For each fetched neighbor $w \in NHE.N(v)$, the PE forms (u, w) and sends it to the NH-CAM's search line as the search word for lookup. ④ Each match generates a valid signal, which increments the NNN triangle counter by one.

Stage 3: HNN Counting via H-CAM. After the NNN pass is complete for the current batch, the PE initiates the HNN counting stage. For the same batch of root vertices, the PE first pre-loads the H-CAM by concatenating each root u with its hub neighbors $h \in HE.N(u)$ to form entries (u, h) .

The HNN pipeline is then driven by the data still residing in the NH-CAM: ⑤ The NH-CAM again acts as a regular memory, sequentially reading out an entry (u, v) to re-establish the base edge. ⑥ The neighbor v is passed to the Decoder, which this time fetches its hub adjacency list, $HE.N(v)$ from the Cache. ⑦ For each fetched hub neighbor $h \in HE.N(v)$, the PE forms (u, h) and sends it to the H-CAM's search line as the search word. ⑧ Similarly, each match generates a valid signal, which increments the HNN triangle counter by one.

Once both the NNN and HNN counting stages are complete for the current batch of root vertices, the CAMs are cleared. The entire process then returns to the beginning of Stage 2 to be repeated for the next batch, continuing until the PE's full workload of non-hub vertices has been processed.

C. Mitigating Inter-Rank Data Movement

A primary challenge in multi-rank NMP systems is mitigating costly inter-rank data movement when scaling to large graphs. A simple solution for small graphs is to replicate the entire graph to each PU's local DRAM [7], [8], [14]. While this completely eliminates inter-rank transfers, the approach is fundamentally unscalable and becomes infeasible once the graph's size exceeds a single rank's memory capacity.

To address this scalability challenge for large graphs, our work leverages the CAS format to enable a uniquely memory-efficient hub-subgraph replication scheme. This approach builds on the graph's segregation into a compact H2H bitmap and the much larger HE/NHE subgraphs. It allows us

to replicate only the small yet vital H2H bitmap to every PU while partitioning the remainder of the graph across the ranks.

This scheme’s effectiveness stems from the fact that this globally replicated bitmap contains all connectivity for HHH and HHN triangles, which constitute the majority of the workload (62.1% on average, as shown in Table I). By replicating only this critical data structure, we mitigate a massive source of inter-rank communication without the prohibitive memory cost of full graph replication. This enables our work to scale efficiently to large graphs where prior methods are impractical.

V. EXPERIMENTAL RESULTS

A. Experiment Setup

1) Datasets. We evaluate the performance of our proposed method on different real-world datasets: Email-Enron (EE), Astro (AS), Youtube (YT), LiveJournal (LJ), Orkut (OK) from the Stanford Network Analysis Project (SNAP) [30], Mico (MC) from the Microsoft co-authorship network [31], Wiki (WK) from the English Wikipedia link graph [32], and Twitter (TT) from the Twitter graph [33]. All graphs are pre-processed with degree orientation. For the `hub_threshold`, we designate the top 1% of vertices as hubs for the first six datasets. For the larger Wiki and Twitter graphs, this threshold is reduced to 0.1%, as this is sufficient to ensure over 90% of triangles are hub-involved while also saving storage space for the H2H subgraph and maintaining a high density. The dataset statistics are shown in Table II.

TABLE II
DATASET INFORMATION AND MEMORY SPACE COMPARISON

Graph	#Vertices	#Edges	#Triangles	CSR (half)	CAS (This work)
EE [30]	36.7K	183.8K	727.0K	861.4KB	1.0MB
AS [30]	18.8K	198.1K	1.4M	847.0KB	922.4KB
MC [31]	100.0K	1.1M	12.5M	4.5MB	4.9MB
YT [30]	1.1M	3.0M	3.1M	15.7MB	27.7MB
LJ [30]	4.0M	34.7M	177.8M	147.6MB	258.1MB
OK [30]	3.1M	117.2M	627.6M	458.8MB	526.7MB
WK [32]	42.6M	255.7M	881.4M	1.1GB	1.4GB
TT [33]	61.6M	1.2G	34.8G	4.7GB	5.2GB

2) PRISM setup: The hardware configuration is detailed in Table III. Our design uses 15 small PUs, each containing 8 PEs equipped with a 512-entry H-CAM and a 256-entry NH-CAM, which are sufficient for the majority of vertices. For the few vertices ($< 0.1\%$) with $d_L > 512$, a dedicated large PU is employed, containing 4 PEs equipped with larger CAMs of 4096 (H-CAM) and 1024 (NH-CAM) entries. We simulate the end-to-end performance with: (1) an in-house behavioral level TC simulator to emulate PRISM computation and generate the memory access trace; (2) the cache simulator [7] for cache behavior and CACTI [34] in 32nm technology for area and energy; (3) Ramulator [35] for DRAM latency; and (4) the 28nm CAM at 400MHz [8] for CAM metrics.

3) Compared Triangle Counting Systems. For our evaluation, we compare against three state-of-the-art NMP-based TC architectures. We reproduced the work of CLAP [8] and CRISP [14], ensuring our experimental configurations were comparable to their original papers. The performance data for

DIMMinning is directly cited from their publication [7]. The hardware configurations are as follows: DIMMinning utilizes 32 NMP modules with 16 DRAM ranks; CLAP employs 16 NMP modules, each with 8 PEs across 16 DRAM ranks; and CRISP integrates 128 PEs within a 32-channel HBM module.

TABLE III
PRISM HARDWARE CONFIGURATION

Parameter	Configuration	
DRAM capacity / #Ranks	64GB / 16	
DDR4 configuration	4Gb x8 2400R	
Frequency (DRAM / PU)	1200MHz / 400MHz	
PU Type	Small	Large
#PUs / #PEs per PU	15 / 8	1 / 4
CAM Entries (H / NH)	512 / 256	4096 / 1024
Tag / Neighbor bits	9 / 32	10 / 32
Cache size per PE	8KB	16KB

B. Overall Performance

We compare our work with the above NMP TC accelerators, DIMMinning, CLAP, and CRISP. As shown in Table IV, our work achieves average end-to-end speedups of $2.21\times$, $2.05\times$, and $1.56\times$ over these designs, respectively.

TABLE IV
COMPARISON OF END-TO-END RUNTIME (SECONDS)

	DIMMinning [7]	CLAP [8]	CRISP [14]	PRISM This work
EE	N/A	4.20E-04	2.42E-04	1.23E-04
AS	2.60E-04	1.81E-04	1.41E-04	1.72E-04
MC	1.60E-03	1.52E-03	1.44E-03	1.02E-03
YT	5.20E-03	2.70E-03	1.53E-03	1.27E-03
LJ	6.90E-02	4.92E-02	4.64E-02	2.96E-02
OK	N/A	4.83E-01	4.67E-01	3.28E-01
WK	N/A	1.73E+00	1.65E+00	5.22E-01
TT	N/A	1.65E+01	OoM	4.73E+00
Speedup	2.21	2.05	1.56	1.00

N/A: Data not available in the original paper. OoM: Out of Memory.

Our evaluation is conducted in two scenarios based on the dataset size. For the first six datasets small enough to fit within a single DRAM rank (EE to OK), we employ the full graph replication strategy for all accelerators, eliminating inter-rank communication. The on-chip execution time breakdown for all datasets is shown in Fig. 4, with a detailed memory traffic analysis in Section V-C.

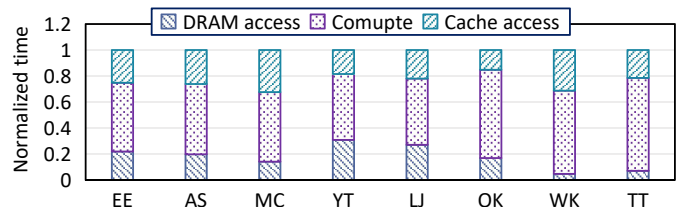


Fig. 4. Breakdown of on-chip execution time.

For the two large-scale datasets (WK and TT) that exceed single-rank capacity, graph partitioning is necessary. We configured CLAP and CRISP with a baseline equal partitioning [7], noting that TT surpassed the memory capacity of

CRISP (OoM error). In contrast, PRISM utilizes the hub-subgraph replication strategy. The results underscore the superiority of our holistic design on large graphs, with PRISM achieving speedups over CLAP of $3.31\times$ on WK and $3.49\times$ on TT. A detailed analysis of how this replication strategy contributes to performance by reducing data movement is presented in Section V-D.

C. Memory Traffic Reduction

To quantify the improvements from reduced memory access, we compared our work against CLAP and CRISP. As detailed in Fig. 5, our work reduces DRAM access volume on average by 39.49% and 32.10% compared to CLAP and CRISP, respectively. This advantage is particularly pronounced on large-scale graphs, such as the WK and TT datasets, where reductions reached 64.65% and 68.00% respectively, compared with CLAP.

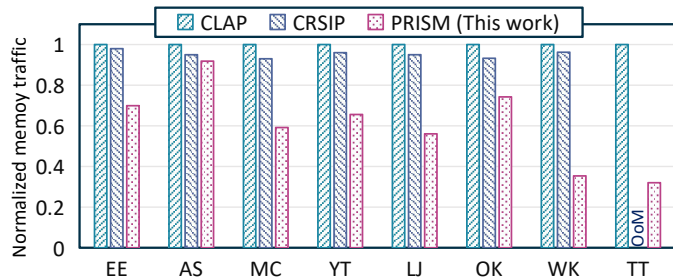


Fig. 5. Overall memory traffic comparison.

This marked improvement is driven by two key optimizations. First, the compact H2H bitmap for HHH and HHN counting enables cache-friendly lookups, drastically reducing memory traffic. Second, partitioning the neighbor sets into separate hub and non-hub sets shrinks the working set for the subsequent intersection, improving memory efficiency by preventing irrelevant data from polluting the cache.

The on-chip cache provides efficient access to frequently-used intermediate data, including the H2H bitmap and NHE/HE subgraphs. As Fig. 6 shows, increasing the cache size boosts the hit rate, reducing DRAM access volume. However, this trend is countered by longer access latency and higher area cost, creating a performance trade-off. Therefore, to balance performance and area, we select a 16KB cache for large PEs and an 8KB cache for small PEs.

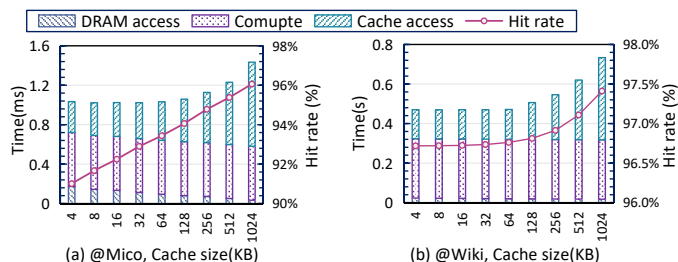


Fig. 6. Latency breakdown and cache size design space exploration.

D. Data Movement Reduction

We evaluate the hub-subgraph replication against two baselines—full replication and equal splitting—on four large

graphs across 16 ranks (Fig. 7). Baselines prove unscalable: full replication hits memory limits (OoM on WK/TT), while equal partitioning incurs high communication overhead, with runtime penalty growing from $1.20\times$ to $1.50\times$.

In contrast, our strategy avoids OoM errors with minimal communication overhead, increasing only from $1.08\times$ to $1.13\times$. This effectively mitigates the bottleneck, extending NMP-based TC acceleration to large graphs.

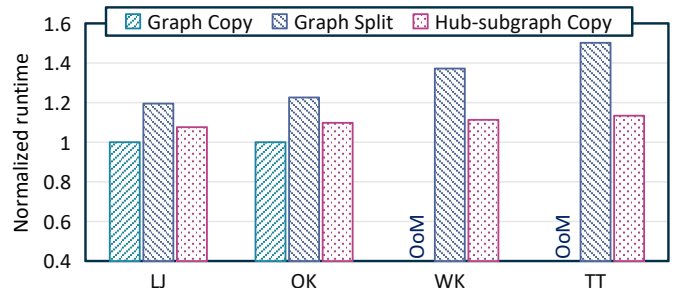


Fig. 7. Comparison of three distribution strategies on 16 ranks: full replication (Graph Copy), equal partitioning (Graph Split), and hub-subgraph replication (Hub-subgraph Copy).

E. Area and Power Analysis

The area and power overhead of our design are detailed in Table V. A key advantage of our scheme is the significant enhancement of data locality. This improved locality enables the use of a much smaller cache within each PE, which in turn substantially reduces both area and power overhead while maintaining a low number of DRAM accesses. Compared to CLAP, our design achieves a 19.28% reduction in area and a 13.49% reduction in power, showcasing the benefits of our algorithm-hardware co-design.

TABLE V
AREA AND POWER ANALYSIS OF PRISM

	Compute Units		Fetcher	Controller	Total
	CAM	ICU			
Area (μm^2)	701,980	72,534	5,260,932	667,255	6,702,701
Power (mW)	757.25	2.65	1730.90	367.10	2857.90

VI. CONCLUSION

In this paper, we introduced PRISM, a hardware-software co-design framework to address key challenges in NMP-based triangle counting. PRISM employs a connectivity-aware partitioning strategy that divides the graph into hub and non-hub regions, enabling a unified approach to enhance performance. This strategy improves data locality through region-specific counting methods, accelerates set intersection via a heterogeneous engine combining bitmap and CAM-based techniques, and minimizes inter-rank communication by replicating only a compact hub-subgraph.

Evaluation across eight real-world datasets demonstrates that PRISM achieves an average speedup of $2.05\times$ and reduces DRAM accesses by 39.49% compared to the state-of-the-art NMP-based TC accelerator. By holistically addressing the core bottlenecks of locality, computation, and communication, PRISM offers a scalable yet efficient solution for accelerating triangle counting on large real-world graphs.

REFERENCES

- [1] C. Seshadhri, A. Pinar, and T. G. Kolda, "Wedge sampling for computing clustering coefficients and triangle counts on large graphs," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 4, pp. 294–307, 2014.
- [2] H. Yin, K. Wang, W. Zhang, Y. He, Y. Zhang, and X. Lin, "Motif counting in complex networks: A comprehensive survey," *arXiv preprint arXiv:2503.19573*, 2025.
- [3] L. Hu, N. Guan, and L. Zou, "Triangle counting on gpu using fine-grained task distribution," in *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2019, pp. 225–232.
- [4] S. Pandey, Z. Wang, S. Zhong, C. Tian, B. Zheng, X. Li, L. Li, A. Hoisie, C. Ding, D. Li *et al.*, "Trust: Triangle counting reloaded on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2646–2660, 2021.
- [5] L. Hu, L. Zou, and Y. Liu, "Accelerating triangle counting on gpu," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 736–748.
- [6] D. A. Bader, F. Li, A. Ganeshan, A. Gundogdu, J. Lew, O. A. Rodriguez, and Z. Du, "Triangle counting through cover-edges," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2023, pp. 1–7.
- [7] G. Dai, Z. Zhu, T. Fu, C. Wei, B. Wang, X. Li, Y. Xie, H. Yang, and Y. Wang, "Dimining: pruning-efficient and parallel graph mining on near-memory-computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 130–145.
- [8] T. Fu, C. Wei, Z. Zhu, S. Yang, Z. Yu, G. Dai, H. Yang, and Y. Wang, "Clap: Locality aware and parallel triangle counting with content addressable memory," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.
- [9] L. Yan, X. Lu, S. Xu, X. Chen, X. Zou, Y. Han, and X.-H. Sun, "Prominer: Enhancing locality, parallelism, and offloading for graph mining on processing-in-memory systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.
- [10] Y. Wu, J. Zhu, W. Wei, L. Chen, L. Wang, S. Wei, and L. Liu, "Shogun: A task scheduling framework for graph mining accelerators," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.
- [11] H. Qi, Y. Zhang, L. He, K. Luo, J. Huang, H. Lu, J. Zhao, and H. Jin, "Psmminer: A pattern-aware accelerator for high-performance streaming graph pattern mining," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.
- [12] Z. Li, X. Chen, and Y. Han, "Tminer: A vertex-based task scheduling architecture for graph pattern mining," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1295–1308.
- [13] N. Talati, H. Ye, Y. Yang, L. Belayneh, K.-Y. Chen, D. Blaauw, T. Mudge, and R. Dreslinski, "Ndminer: accelerating graph pattern mining using near data processing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 146–159.
- [14] S. Zhang, X. Wang, W. Zhao, and Y. Jin, "Crisp: Triangle counting acceleration via content addressable memory-integrated 3d-stacked memory," in *2024 IEEE International Test Conference in Asia (ITC-Asia)*. IEEE, 2024, pp. 1–6.
- [15] J. Su, "Pimminer: A high-performance pim architecture-aware graph mining framework," Master's thesis, Illinois Institute of Technology, 2022.
- [16] M. Besta, R. Kanakagiri, G. Kwasniewski, R. Ausavarungnirun, J. Beránek, K. Kanellopoulos, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, I. Stefan *et al.*, "Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 282–297.
- [17] X. Ma, W. Zhang, X. Wang, T. Yu, B. Wu, G. Qu, and W. Zhao, "A combined content addressable memory and in-memory processing approach for k-clique counting acceleration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [18] M. K. Esfahani, P. Kilpatrick, and H. Vandierendonck, "Lotus: locality optimizing triangle counting," in *PPoPP*, 2022, pp. 219–233.
- [19] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim *et al.*, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 350–352.
- [20] Y. Yang, W. Yang, Q. Wang, N. Jing, J. Jiang, Z. Mao, and W. Sheng, "An efficient near-bank processing architecture for personalized recommendation system," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 122–127.
- [21] B. Tian, Y. Li, L. Jiang, S. Cai, and M. Gao, "Ndpbridge: Enabling cross-bank coordination in near-dram-bank processing architectures," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 628–643.
- [22] B. Tian, Q. Chen, and M. Gao, "Abndp: Co-optimizing data access and load balance in near-data processing," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 3–17.
- [23] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "Spacea: Sparse matrix vector multiplication on processing-in-memory accelerator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 570–583.
- [24] H. Li, D. Chen, and T. Mitra, "Sadimm: Accelerating sparse attention using dimm-based near-memory processing," *IEEE Transactions on Computers*, 2024.
- [25] L. Chen, D. Lyu, J. Jiang, Q. Wang, Z. Mao, and N. Jing, "Asyncdimm: Achieving asynchronous execution in dimm-based near-memory processing," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 518–532.
- [26] M. Lee, S.-S. Lee, K. Kim, E. Lee, and S.-J. Jang, "Hail-dimm: Host access interleaved with near-data processing on dimm-based memory system," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [27] Y. Li, Y. Jin, B. Tian, H. Zhang, and M. Gao, "Ansmet: Approximate nearest neighbor search with near-memory processing and hybrid early termination," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1093–1107.
- [28] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging trends in design and applications of memory-based computing and content-addressable memories," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1311–1330, 2015.
- [29] Y. Hu, H. Liu, and H. H. Huang, "Tricore: Parallel triangle counting on gpus," in *SCI8: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 171–182.
- [30] L. Jure, "Snap datasets: Stanford large network dataset collection," Retrieved December 2021 from <http://snap.stanford.edu/data>, 2014.
- [31] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 517–528, 2014.
- [32] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd international conference on world wide web*, 2013, pp. 1343–1350.
- [33] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 591–600.
- [34] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.
- [35] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.