Configurable DSP-Based CAM Architecture for Data-Intensive Applications on FPGAs

***Abstract*—Content-addressable memory (CAM) is a type of fast memory unique in its ability to perform parallel searches of stored data based on content rather than specific memory addresses. They have been used in many domains, such as networking, databases, and graph processing. Field-programmable gate arrays (FPGAs) are an attractive means to implement CAMs because of their low latency, reconfigurable, and energy-efficient nature. However, such implementations also face significant challenges, including high resource utilization, limited scalability, and suboptimal performance due to the extensive use of look-up tables (LUTs) and block RAMs (BRAMs). These issues stem from the inherent limitations of FPGA architectures when handling the parallel operations required by CAMs, often leading to inefficient designs that cannot meet the demands of high-speed, data-intensive applications. To address these challenges, we propose a novel configurable CAM architecture that leverages the digital signal processing (DSP) blocks available in modern FPGAs as the core resource. By utilizing DSP blocks' data storage and logic capabilities, our approach enables configurable CAM architecture with efficient multi-query support while significantly reducing search and update latency for data-intensive applications. The DSP-based CAM architecture offers enhanced scalability, higher operating frequencies, and improved performance compared to traditional LUT and BRAM-based designs. In addition, we demonstrate the effectiveness of our proposed CAM architecture with a triangle counting application on real graphs. This innovative use of DSP blocks also opens up new possibilities for high-performance, data-intensive applications on FPGAs. Our proposed design is open-sourced at: https://anonymous.4open.science/r/CAM-B9D1/.**

***Index Terms*—Content addressable memory, digital signal processor, FPGA, scalability, high-performance**

I. Introduction

Content-addressable memory (CAM) performs fast content matching due to its ability to parallel searches across all stored entries in a single operational cycle [8], [12], [16]. Unlike traditional memory, which requires sequential access or indexing methods that introduce significant latency, CAM's processing pattern significantly accelerates search-intensive operations common in graph processing, such as finding adjacent nodes or matching patterns within the graph [5].

FPGAs are widely adopted as accelerator platforms for data-intensive applications, this drives the requirement for CAM design on it. Due to the reconfigurability advantage, the CAM implementations on FPGA emulate all CAM types, including binary, ternary, and range-matching CAMs. Current CAM implementations on FPGAs are typically based on LUTs, BRAMs, or hybrid designs combining both, but all face significant challenges, especially in scalability and performance, as shown in Figure 1.

LUT-based CAMs require extensive LUT resources, as the size of the CAM grows, the number of used LUTs increases exponentially [7], [18], leading to routing congestion, longer critical paths, and dramatically reduced operating frequencies. While LUTRAM-based designs optimize LUT usage, they impose additional preprocessing overhead for input data, complicating updates and limiting their suitability for dynamic applications [10], [16], [20]. BRAM-based CAMs, on the other hand, leverage the large storage capacity of Block RAMs [3], [10], but these memory blocks are designed for sequential access, requiring additional logic for parallel comparisons, which reduces search speed and increases complexity [4], [17]. Hybrid-resource CAMs [4], [9] attempt to balance the flexibility of

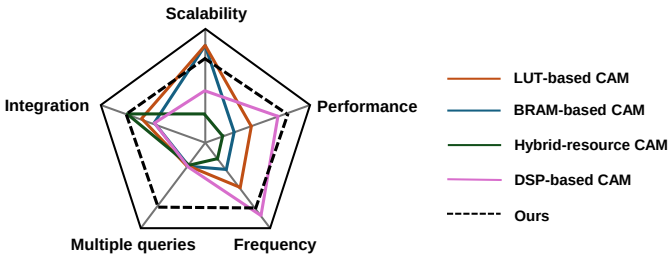

Fig. 1: Current FPGA CAM designs.

LUTs with the storage efficiency of BRAMs, but they often encounter complex update processes due to the overhead of managing multiple resource types. The only existing DSP-based CAM design shows the advantage of higher operating frequency and optimized data pathways while encountering long search time which is not suitable for data-intensive applicatioins [14].

Furthermore, the inherent computational requirements of parallel comparisons in CAMs add to the challenges, making these designs less effective for data-intensive applications demanding scalability, rapid updates, and universal applicability.

This paper presents a novel configurable design of high-performance CAM with DSP slices as the primary resource. The proposed CAM architecture addresses the need for flexible and efficient data updates and searches in data-intensive applications and also provides easy integration to accelerator systems. The main contributions are listed as follows.

- **Configurable DSP-based CAM architecture**: We introduce a fully parameterized CAM design that mainly uses DSP slice and can be customized at different granularities, across cell, block, and unit levels.
- **Multi-query support**: Our CAM architecture supports multiple content searches in parallel with an optimized search logic.
- **Advantageous experimental results**: Our proposed CAM design on FPGA demonstrates superior performance in different scales.
- **Real-world case study**: We demonstrate the effectiveness of our proposed CAM design in the context of triangle counting in graphs, showcasing improved efficiency and scalability.

This paper is organized as follows. Section II surveys the current CAM designs on FPGAs and abstracts these designs' inefficiency for data-intensive applications. Section III presents our proposed parallel DSP-CAM architecture. Section IV evaluates the proposed CAM design and demonstrates its advantage over the existing designs. A case study with triangle counting on real-world graphs is presented in Section V which demonstrates the effectiveness of our proposed CAM design. Section VI concludes this paper.

II. Background

Content-addressable memories (CAMs) can be categorized into Binary CAM, Ternary CAM (TCAM), and Range-Matching CAM

TABLE I: A survey of recent CAM designs on FPGAs

Design Name	Category	Platform	Max CAM Size	Frequency (MHz)	Resource Utilization			Latency (cycles) ‖	
					LUT	BRAM	DSP	Update	Search
Scale-TCAM [10]	LUT	XC7V2000T	4096 × 150 bits	139	322648 *	0	0	33	-
DURE [16]	LUT	Xilinx Virtex-6	1024 × 144 bits	175	35807	0	0	65†	1†
BPR-CAM [15]	LUT	XC6VLX760	1024 × 144 bits	111	15260	0	0	-	2
Frac-TCAM [20]	LUT	XC7V2000T	1024 × 160 bits	357	16384	0	0	38	-
HP-TCAM [19]	BRAM	Xilinx Virtex-6	512 × 36 bits	118	5326	56	0	-	5
PUMP-CAM [17]	BRAM	XC6VLX760	1024 × 140 bits	87	7516	80	0	129	-
IO-CAM [13]	BRAM	Intel Arria V 5ASTD5	8192 × 32 bits	135	19017 ‡	2112§	0	-	-
REST-CAM [4]	Hybrid	Xilinx Kintex-7	72 × 28 bits	50	130	1	0	513	5
Preußer, et.al [14]	DSP	XCVU9P	1000 × 24 bits	350	2843	0	1022	-	42
Ours	DSP	U250	9728 × 48 bits	235	72178	4	9728	6	8

‖ The latency is measured for a single end-to-end operation. * The number of LUT is calculated by the number of slices (80662) multiplied by four. † The update and search latencies are measured on a single CAM block with dimensions of 512 x 36. ‡ The number of used ALM in Intel FPGA. § The number of used M10K in Intel FPGA. - mark in the table indicates the value is not reported in the literature.

(RMCAM). Binary CAMs perform exact-match searches using binary data (0s and 1s), making them suitable for applications like cache memory tag matching where precise data retrieval is essential [2], [3], [12], [18]. Ternary CAMs (TCAMs) introduce a third state, the "don't care" (X) condition, allowing for partial or wildcard matching, and is widely used in IP routing or packet redirection. Range-matching CAMs are designed to match input data within specific numerical ranges, which is valuable in applications like database indexing and firewall rule matching. Recent representative designs are collected in Table I. FPGAs can emulate all CAM designs due to their logical representation capability. However, the significant differences in resource types within FPGAs necessitate categorizing CAM designs based on the specific resources primarily utilized: LUT, BRAM, DSP, and combinations thereof.

A. Challenges When Facing Data-Intensive Applications

Data-intensive applications often require frequent data updates and rapid, simple data searches, which raise challenges for the underlying CAM architecture. The challenges observed when adopting these CAM designs for data-intensive applications include:

Limited Scalability. In traditional CAM implementations that adopt LUT and BRAM, as the size of the CAM increases — in terms of both the number of entries and the width of each entry — the number of required LUT or BRAM grows exponentially. Consequently, the implemented timing drops significantly, which become impractical for data-intensive tasks like graph processing.

High Search and Update Latency. Many CAM architectures are optimized for read-intensive operations with infrequent updates, which is not suitable for applications that require frequent data modifications. Frequent updates result in increased latency and create bottlenecks that reduce the system's ability to process data efficiently. The slow update rates and the complexity of update logic hinder the performance of applications that demand immediate reflection of data changes, such as dynamic graph algorithms.

Multiple Concurrent Queries. Multi-query is an essential processing pattern in data-intensive applications. However, CAMs on FPGAs rely heavily on resources like Lookup Tables (LUTs) to perform simultaneous comparisons across all stored entries, which makes them not capable of dealing with multiple search requests from different input data. Even in the most recent design that adopts DSP as the processing resource, multi-query support remains unexplored.

System Integration Complexity. Data-intensive applications on FPGAs generally require domain-specific architectures, in which the CAM implementations must coexist with other system components to achieve the accelerator functionality, which requires the configurability and integrability of the CAM microarchitecture to be considered during the design.

B. Design Motivations

The challenges outlined above motivate a reevaluation of CAM design on FPGA for data-intensive applications.

Leveraging DSP Blocks for better scalability. DSP blocks are specialized units optimized for high-speed arithmetic and logic operations. Repurposing the DSP slice for CAMs, we may reduce reliance on LUTs and BRAMs, alleviating resource constraints and enhancing performance.

Balancing update and search latency. Data-intensive applications require more frequent data updates which necessitates a more balanced update and search logic path design.

Multi-query support. Supporting multiple concurrent queries at the microarchitecture level is necessary.

Configurable architecture. The variation of the accelerator architectures necessitates a flexible CAM architecture that can accommodate these differing needs without requiring a complete redesign for each application.

In summary, utilizing DSP blocks as the core resource in our CAM design addresses the critical challenges of resource utilization and scalability inherent in traditional FPGA-based CAMs. Multi-query support equips the system with the flexibility to dynamic data and application requirements, enhancing efficiency and responsiveness. Parameterized architectural design is also necessary to support the adaptation of it to wider acceptance of applications.

III. Adaptive Parallel DSP CAM

With the motivations above, our adaptive parallel CAM design on FPGA adopts DSP slice as the primary resource and is designed hierarchically so it can be fully parameterized. The CAM cell, configured using a DSP slice, provides basic data storage and data comparison functionalities. Multiple CAM cells are grouped to form a CAM block, with integrated data update and search logic, making it the fundamental unit to perform essential CAM operations. The proposed adaptive parallel CAM unit is constructed by combining multiple CAM blocks with additional logic for data updates and parallel searches. This CAM unit can operate as multiple independent CAMs, with each CAM sized at the granularity of a CAM block, providing exceptional flexibility. The CAM unit also supports multiple search queries across multiple grouped CAM blocks, significantly enhancing processing throughput and parallelism.

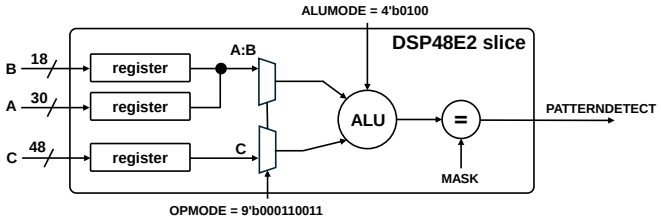

Fig. 2: DSP-based CAM cell.

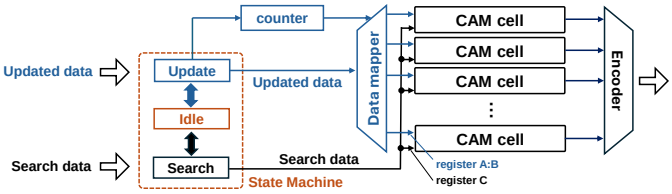

Fig. 3: Architecture of DSP-based CAM block.

A. Configurable CAM cell with DSP

The DSP48E2 [6] slice is designed to provide arithmetic processing capacity, which contains an arithmetic unit, logic and comparison unit, shift and preprocessing unit, registers, control, and configurable logic, cascading and interconnection. Turning the DSP slice into a CAM cell requires the configuration of the DSP block into logic processing mode. Specifically, the operational mode of the DSP slice is determined by configuring the ALUMODE and OPMODE registers, as shown in Figure 2, enabling the slice to perform XOR operations between the two 48-bit registers:

$$O = (A : B) \oplus C \tag{1}$$

With this configuration, in each DSP48E2 slice, registers A and B are concatenated to provide a 48-bit storage space for input data, while register C is used to hold the search key during search operations. For Ternary CAM (TCAM) and Range-Matching CAM (RMCAM) functionality, the additional MASK function is employed to handle partial matches and range checks. By setting specific bits in the mask to ”don’t care” (X) status, the DSP performs post-processing after the XOR operation to identify matches based on the masked conditions. However, in Range-Matching CAM, the representation is limited to ranges where the start and end values are powers of 2. This limitation arises from the mask control’s bit-level granularity, restricting the range specifications’ flexibility. Table II summarizes the MASK definitions and their effects for binary, ternary, and range-matching CAMs. The mask is also used for the data bit width control, where the unused data bits are masked out to simplify the data path.

TABLE II: MASK value for CAM type configuration

Type	MASK value	Behavior
BCAM	All bits are zero	All bits are compared
TCAM	Active bits = 0, ignored bits = 1	Bits with MASK = 1 are ”don’t care” bits
RMCAM	Relevant bits = 0, others = 1	Bits with MASK = 0 are selected range

B. CAM block microarchitecture

A DSP-based CAM cell can not perform the full functional update and search operations as it only provides the data storage and comparison logic. The design of a CAM block involves organizing multiple DSP slices along with additional control and data management components to support search and update operations. Specifically, a CAM block in our design includes a configurable number of CAM cells, a State Machine, a Counter, a Data Mapper, and an Output Encoder, as shown in Figure 3. The Data Mapper ensures that incoming data is formatted to match the CAM cells’ internal data structure. The Counter calculates precise storage locations for incoming data by tracking the number of input data packets and outputs these locations to the Data Mapper, allowing it to route the data to the corresponding CAM block. The Update and Search Controller incorporates a free-running state machine to manage operational states such as idle, update, and search, ensuring state transitions without external intervention.

In the update state, the CAM block accepts multiple data by taking advantage of the register-based storage within the DSP slices, enabling simultaneous updates across multiple CAM cells in a single cycle. When the CAM block transitions to the search state, the operation becomes inherently sequential. In this state, only one input search key is broadcast to all CAM cells within the block in a single cycle. This ensures each CAM cell processes the search key simultaneously. To manage the search results, the Output Encoder module consolidates outputs from all DSP slices, encoding the address of the slice where a match is found to generate the final search result. The encoding scheme is left configurable to support various address management solutions.

C. CAM unit microarchitecture

To efficiently utilize multiple CAM blocks and support multiple concurrent search queries, we design the CAM unit to contain multiple CAM blocks. An overview of our architecture is shown in Figure 4, which consists of the CAM unit update module and the search module. A CAM unit is wrapped with the input and output interfaces to the user kernels.

1) Update Operation: The update operation in the CAM unit is executed through the update datapath. The user kernel specifies the update mode, address, and the data to be updated to the CAM. The Block Selector decodes the address to the exact location and routes the input data to the appropriate CAM block and CAM cell. There are slight differences between the single data update and the whole unit update.

a) Single data update: To update a single data in the CAM unit, the user kernel only needs to send the address and the data to the Update module, where the Block Selector decodes the location of the CAM block and the CAM cell, enabling the update to the data storage register.

b) Whole unit update: When all the data in the CAM needs to be updated, the Block Selector executes in whole unit update mode, in which the data is updated with the maximum input bus width. The Block Selector routes all the data from the input based on the ID of the CAM block and the offset of the data in the block. The Block Selector operates based on the address calculation function, which calculates the addresses and offsets based on the input bus width and number of CAM blocks and CAM cell configurations. The Routing Table is updated based on the address calculation function - the blocks occupied by a continuous sequence of data are marked to be in the same group. Then the group information is updated to the Routing Table.

2) Search operation: The search operation is facilitated by three key components: the Routing Table (RT), the Router Compute (RC), and the Crossbar.

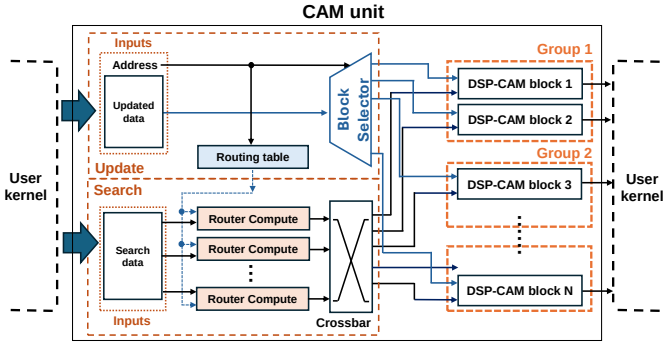

Fig. 4: Architecture of our proposed CAM unit.

a) Single data search: To perform a search for a single data, the data is broadcast to all CAM blocks and the results from all CAM blocks are sent to the user kernel.

b) Multi-query search: For a parallel multi-query search, which involves multiple groups within the CAM unit, the broadcasting of the search key is based on the grouping information from the routing table. Each group operates independently for different inputs, while the blocks in each group execute in parallel and return the results. This architecture allows the CAM unit to dynamically adapt to varying search workloads when the utilization of the block is low while maintaining high efficiency and high throughput.

Specifically, for our proposed architecture, the user kernel first performs updates while informing the number and list to CAM. The CAM then automatically assigns a unique ID to each list and stores the updates. Once the CAM is fully populated, it can notify the kernel to send multiple search keys simultaneously. These keys are packaged together and processed in parallel for comparison, ensuring efficient utilization of resources and achieving a higher throughput.

D. Parameterized architectural components

To enable seamless integration into different applications, our CAM unit is fully parameterized with different hierarchies of configurations. The configurations include cell-level, block-level, and unit-level. As shown in Table III, parameters such as cell type, data width, block size, unit size, and bus width are configured at the design stage, ensuring optimal resource utilization and hardware compatibility. We design the source file in templates where all the parameters can be defined before the CAM unit is generated.

TABLE III: Configurable Parameters for CAM Unit

Granularity	Parameter	Description
CAM Cell	Cell type	The type of CAM: Binary, Ternary, Range-matching.
	Storage Data Width	Width of the stored data ($\leq$ 48 bits).
CAM Block	Block Size	Number of cells per CAM block.
	Input Bus Width	The width of the data path in CAM block.
	Result Encoding	Encoding scheme for consolidating search results from multiple CAM blocks.
CAM Unit	Unit Size	Number of CAM blocks in the CAM unit.
	Input Bus Width	The width of the data path in CAM unit.

IV. CAM Evaluations

To comprehensively evaluate the effectiveness of our proposed parallel CAM design, metrics are analyzed at three different granularities: CAM cell, CAM block, and CAM unit. We assess performance, efficiency, and scalability at each granularity to provide a detailed understanding of the architecture's characteristics. Across all granularities of evaluation, the key metrics include:

- **Latency**: Measured in clock cycles, indicates the time required for a single end-to-end operation.
- **Throughput**: The number of operations (updates or searches) performed per second.
- **Resource Utilization**: Quantifies the FPGA resources that are consumed, including LUTs, FFs, DSP, and BRAMs.
- **Scalability**: Reflects the ability to adapt the design to a larger size without significant performance degradation.

The experiments were conducted on an AMD Alveo U250 accelerator card, a high-performance FPGA platform based on the UltraScale+ architecture. The U250 features four DDR4 memory channels and a PCIe Gen3 x16 interface, providing ample bandwidth for data-intensive applications. The resources of the chip is collected in Table IV. The FPGA design was implemented using Xilinx Vivado Design Suite, with the CAM architecture synthesized and deployed on the U250. Timing constraints were validated, and resource utilization metrics, including LUTs, BRAMs, and DSPs, were collected using the Vivado implementation reports. The evaluation included testing various CAM configurations, demonstrating the scalability and performance of the proposed design under different workloads.

TABLE IV: Resource capacity of AMD Alveo U250

Resource	LUTs	Registers	BRAM	URAM	DSP
Quantity	1,728K	3,456K	2,688	1,280	12,288

A. CAM Cell Evaluation

Taking advantage of the reconfigurability of the FPGA DSP slice, our CAM cell is compact and flexible for different CAM functionalities. The configuration of the OPMODE and ALUMODE does not change the resource utilization of the memory cell. As shown in Table V, the resource consumption, update, and search latency stay the same for the configuration of Binary CAM cell, Ternary CAM cell, and Range-matching CAM cells. This provides the basis for a scalable and high-performance CAM unit design.

TABLE V: CAM Cell Evaluation

Metric	Value
Storage Capacity	1 entry $\leq$ 48 bits
Update Latency	1 cycle
Search Latency	2 cycles
Resource Utilization	1 DSP, 0 LUT, 0 BRAM

B. CAM Block Evaluation

Due to the stable characteristic of our DSP CAM cell design, at the granularity of the CAM block, we mainly focus on evaluating the block configuration with scaled numbers of CAM cells for storage of data width no more than 48 bits. The results are shown in Table VI. We set the size of the block to be power-of-two values to maintain a hardware-friendly architecture, which are also the commonly used values for the size of a cache line. Notably, the Update Throughput and Search Throughput are measured in data

update or search operations per second, which is different from the normally used processed packet per second. Our CAM design maintains high frequency and stable update and search latency with different size configurations. Specifically, when the size of the block reaches 256, we added an additional buffer at the Encoder output to optimize the implementation timing. It leads to an increased search latency of 4 but does not harm the search throughput. Our LUT utilization remains very low when compared to designs in Table I, this leaves more resources for the other parts of the system.

TABLE VI: CAM Block Evaluation with different size

CAM size	32	64	128	256	512
Update Latency (cycle)	1	1	1	1	1
Search Latency (cycle)	3	3	3	4	4
Update Throughput (op/s)	4800	4800	4800	4800	4800
Search Throughput (op/s)	300	300	300	300	300
# of LUTs	694	745	808	1225	1371
Utilization(%)	0.05	0.05	0.05	0.07	0.08
# of DSP	32	64	128	256	512
Utilization(%)	0.26	0.52	1.04	2.08	4.17
BRAM Utilization	0	0	0	0	0
Frequency(MHz)	300	300	300	300	300

C. CAM Unit Evaluation

a) Scalability evaluation: To evaluate the effectiveness of our proposed microarchitecture at managing the scaled number of CAM blocks, except for the basic metrics, the evaluations are extended to the performance of randomly updating and searching a single value in the CAM unit. Specifically, the size of the CAM block is set to 256, and the Input Bus Width is 512, which is to be compatible with the interface width of the external DDR memory port in our evaluation platform. The number of adopted blocks increases when the CAM size increases. We maintain the data to be 48 bits so all the internal data paths are active.

The results are collected in Table VII. The required number of LUT increases linearly when the size of the CAM unit increases, this is due to the logic required by the data update and search logic. With the given 11,508 DSPs on our platform, we can easily achieve a CAM size that reaches $9K \times 48$ bits, where the 79.25% of the DSPs are adopted for this CAM unit but only 2.92% of the LUT resource is used. As a result, we achieve a 254MHz clock frequency. The update and search performance are collected in Table VIII, where the data width is configured as 32 bits for wider adoption. The update latency does not change when the CAM unit size changes, which is mainly because of the simpler datapath for updates. The search latency increases by one clock cycle when the CAM size is larger than 2K, which is mainly due to the buffering in the encoder of the CAM block to optimize the timing during implementation. The update and search throughput only relate to the clock frequency since the processes are pipelined with an initial interval of 1.

TABLE VII: CAM Unit Configuration and Resource Utilization

CAM size	LUT Utilization	DSP Utilization	Freq. (MHz)
512 × 48 bits	2491	512	300
1024 × 48 bits	5072	1024	300
2048 × 48 bits	10167	2048	300
4096 × 48 bits	20330	4096	265
6144 × 48 bits	29385	6144	252
8192 × 48 bits	38191	8192	240
9728 × 48 bits	45244	9728	235

TABLE VIII: CAM Performance for 32-bit data with different sizes

CAM size	128	512	2048	4096	8192
Update Latency (cycle)	6	6	6	6	6
Search Latency (cycle)	7	7	8	8	8
Update Throughput (op/s)	4800	4800	4800	4064	3840
Search Throughput (op/s)	300	300	300	254	240

b) Comparison to the state-of-the-arts: We insert our design with the maximum size into Table I which contains the state-of-the-art designs at their maximum sizes. Our CAM design shows better scalability in terms of occupying 79.25% of the on-chip DSP resources with significantly less LUT utilization. Also, when compared to the existing CAM designed based on DSP, the update and search latency of our proposed CAM is balanced and more suitable for data-intensive applications.

V. Case Study: Triangle Counting with CAM

To demonstrate the effectiveness of our proposed CAM design, we implement an accelerator that primarily relies on the CAM to perform triangle counting. Furthermore, we conduct this evaluation to highlight both the seamless integration of our CAM design into the accelerator and the enhanced functionality enabled by the CAM.

A. Preliminary of triangle counting

Triangle counting is a fundamental operation in graph analytics, widely used for calculating clustering coefficients and understanding transitivity in networks across domains such as social sciences, biology, and computer science. A typical edge-centric triangle counting algorithm involves retrieving the adjacency lists of two connected vertices and performing a set intersection to identify common neighbors, as shown in Figure 5. Due to the compact storage format of graph data, which are generally in the Compressed Sparse Row (CSR) format, this operation relies heavily on set intersection on two random length lists, a typical data-intensive task that poses significant challenges to common address-based memory systems, particularly for large-scale and irregular graph data. Two primary challenges arise from this operation. First, the inherent sequential nature of the set-intersection operation makes it difficult to perform efficient parallel processing between two input sets. Second, due to the random lengths of adjacency lists, traditional memory systems, are poorly suited to the unpredictable and frequently small sizes of adjacency lists, resulting in under-utilization of on-chip memory resources. CAM is one of the effective solutions to address the above challenges by enabling highly parallel set intersection operations directly in memory. Consider two adjacency lists with length n and m ($n \leq m$) that require a set intersection. There are two commonly used methods for this operation: the merge-based method and the binary search-based method. The merge-based method processes one comparison per cycle and updates the list indices based on the comparison results, achieving a time complexity of $O(m + n)$. In contrast, the binary search method operates by repeatedly dividing the longer list into half, resulting in a time complexity of $O(n \cdot \log m)$. When CAM is applied to this operation, it significantly reduces processing complexity by naturally enabling parallel comparisons. For instance, if we store the longer list in CAM, the total set intersection complexity becomes $O(n)$.

B. Integration of configurable CAM in triangle counting

We build a triangle counting accelerator mainly based on our proposed CAM design, as shown in Figure 6. Specifically, the input data width is set to 32 bits, the CAM cell is configured to be binary,

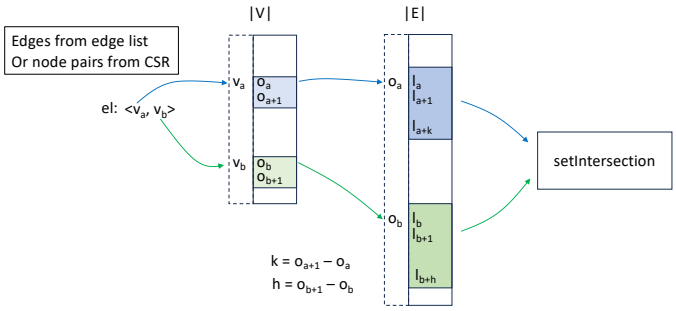

Fig. 5: Simplified process of triangle counting on graphs.

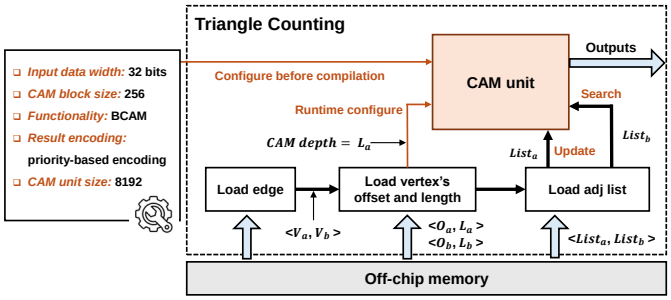

Fig. 6: System architecture of CAM-based triangle counting.

the block size is set to 256 and the system input bus width is 512, the Encoder in the CAM block is configured to a priority-based encoding scheme. There are additional modules as user kernels to perform the other related functionality for triangle counting, including Load edge, Load vertex's offset and length, and load the adjacent lists function modules.

All these graph data are stored in the off-chip memory. The input graph is represented in the CSR format, where each vertex is associated with an offset and length pointing to its neighbors in a column list. The user kernels in the accelerator processes each edge (V_a, V_b) in the graph by first loading the offset index and the length of the lists associated with each vertex, then loading the adjacency lists $<List_a, List_b>$. Before performing the set intersection, the longer list is determined and loaded into the generated CAM unit and the vertices from the shorter one are used as the search keys. The retrieving process performs as follows: a search operation is performed for each vertex in the shorter list. If there is a match, the number of the triangle is increased by one.

C. Performance evaluation

Due to the adaptable parallelism design of our CAM unit, if a single list does not occupy all the CAM blocks[1], more lists will be loaded to fill the blocks, which enables us to perform multiple search queries concurrently. To evaluate the impact of the multi-query support, we also disable it and change the CAM unit to perform single query search only. This design is marked as NoMQ in the following.

We conduct a performance comparison against the official implementation of the triangle counting accelerator provided in the AMD Vitis library [1]. This implementation, which utilizes an optimized merge-based set intersection approach, serves as the baseline for our evaluation. To ensure a fair and consistent comparison, we replicate the baseline [1] implementation on the same platform, AMD Alveo U250, under identical operating conditions. Our CAM-based triangle counting accelerator is configured to match the constraints of the baseline design, which is limited to a single DDR channel. Accordingly, our CAM unit is set with 2K entries to remain within a single super logic region (SLR) since the baseline design is also located inside a single SLR.

[1]List with a length less than 256 occupies the entire CAM block.

TABLE IX: Execution time (ms) of Traditional TC and CAM-Based Approaches

Dataset	Baseline [1]	Ours-NoMQ	Ours	Speedup
facebook_combined	18.7	19.3	11.9	**1.57x**
amazon0302	89.5	30.1	25.3	**3.56x**
amazon0601	230.3	166.8	136.3	**1.69x**
as20000102	7.4	3.5	3.4	**2.18x**
cit-Patents	800	668.1	501.5	**1.60x**
cit-HepPh	5361.1	4256.0	3462.0	**1.55x**
roadNet-CA	108.8	50.3	47.1	**2.31x**
roadNet-PA	88.7	28.2	26.4	**3.36x**
roadNet-TX	96.8	34.8	32.7	**2.96x**
soc-Slashdot0811	259.7	125.4	91.4	**2.84x**

D. Result Analysis

The performance results of our triangle counting accelerator across various graph datasets [11] are summarized in Table IX. The performance of the baseline is known to be constrained by the performance of the merge-based set intersection operations, and it has an optimized utilization of the memory bandwidth that satisfies the processing requirement of the merge-based set intersection hardware kernel. Our design with disabled multi-query support outperforms the baseline in the processing of most of the graph datasets. Due to the significant skewness of facebook_combined dataset, our design without multi-query support is slower than the baseline. The results from our design with multi-query support demonstrate significant performance improvements over the baseline, showcasing the effectiveness of our design. Among the datasets evaluated, the highest speedup is achieved on the amazon0302 dataset, where our design outperforms the baseline by a factor of $3.56\times$. The comparison between "Ours" and "Ours-NoMQ" results also highlights the substantial benefits provided by the multi-query capability in improving the overall performance.

In summary, the parameterized design of our proposed CAM architecture enables easy integration of it into the triangle counting accelerator. The performance results of the triangle counting accelerator based on our CAM design demonstrate significant improvement over the existing optimized baseline design under the same memory and platform condition.

VI. Conclusion

This paper introduces a novel CAM architecture leveraging DSP slices in FPGAs to address the limitations of traditional LUT and BRAM-based designs when facing data-intensive applications. By repurposing DSP blocks, the proposed architecture achieves high scalability, low update and search latency, efficient resource utilization, and easy system integration. The design supports configurable granularity at cell, block, and unit levels, enabling adaptability to diverse data-intensive applications. Experimental results validate the superior performance and scalability of the architecture, with a triangle counting case study demonstrating its practical effectiveness. This work establishes a foundation for integrating DSP-based CAMs into high-performance FPGA systems, opening new avenues for optimizing data-intensive tasks. Our implementation is open-sourced at: https://anonymous.4open.science/r/CAM-B9D1/.

REFERENCES

[1] "Triangle count - vitis graph library documentation," 2022. [Online]. Available: https://github.com/Xilinx/Vitis_Libraries/tree/main/graph/L2/benchmarks/triangle_count.

[2] A. M. S. Abdelhadi and G. G. F. Lemieux, "Deep and narrow binary content-addressable memories using fpga-based brams," in *2014 International Conference on Field-Programmable Technology (FPT)*, 2014, pp. 318–321. [Online]. Available: https://ieeexplore.ieee.org/document/7082808

[3] A. M. Abdelhadi and G. G. Lemieux, "Modular sram-based binary content-addressable memories," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 207–214. [Online]. Available: https://ieeexplore.ieee.org/document/7160073

[4] A. Ahmed, K. Park, and S. Baeg, "Resource-efficient sram-based ternary content addressable memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1583–1587, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7797247

[5] 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 and Test in Europe Conference and Exhibition (DATE)*, 2023, pp. 1–6.

[6] X. Inc., *UltraScale Architecture DSP Slice User Guide*, 2019, document UG579 (v1.9.1), Last updated: October 2, 2019. [Online]. Available: https://0x04.net/~mwk/xidocs/ug/ug579-ultrascale-dsp.pdf

[7] M. Irfan and Z. Ullah, "G-aetcam: Gate-based area-efficient ternary content-addressable memory on fpga," *IEEE Access*, vol. 5, pp. 20 785–20 790, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8049445

[8] M. Irfan, Z. Ullah, and R. C. Cheung, "D-tcam: A high-performance distributed ram based tcam architecture on fpgas," *IEEE Access*, vol. 7, pp. 96 060–96 069, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8755972

[9] M. Irfan, H. E. Yantır, Z. Ullah, and R. C. Cheung, "Comp-tcam: An adaptable composite ternary content-addressable memory on fpgas," *IEEE Embedded Systems Letters*, vol. 14, no. 2, pp. 63–66, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9613821

[10] W. Jiang, "Scalable ternary content addressable memory implementation using fpgas," in *Architectures for Networking and Communications Systems*, 2013, pp. 71–82. [Online]. Available: https://ieeexplore.ieee.org/document/6665177

[11] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection," 2014. [Online]. Available: http://snap.stanford.edu/data

[12] Z. U. Muhammad Irfan and R. C. C. Cheung, "Zi-cam: A power and resource efficient binary content-addressable memory on fpgas," *Electronics*, vol. 8, no. 5, p. 584, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/5/584

[13] X.-T. Nguyen, T.-T. Hoang, H.-T. Nguyen, K. Inoue, and C.-K. Pham, "An efficient i/o architecture for ram-based content-addressable memory on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 3, pp. 472–476, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8395019

[14] T. B. Preußer, M. Chiosa, A. Weiss, and G. Alonso, "Using dsp slices as content-addressable update queues," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 121–126. [Online]. Available: https://ieeexplore.ieee.org/document/9221593

[15] A. Ullah, A. Zahir, N. A. Khan, W. Ahmad, A. Ramos, and P. Reviriego, "Bpr-tcam—block and partial reconfiguration based tcam on xilinx fpgas," *Electronics*, vol. 9, no. 2, p. 353, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/2/353

[16] I. Ullah, Z. Ullah, U. Afzaal, and J.-A. Lee, "Dure: An energy- and resource-efficient tcam architecture for fpgas with dynamic updates," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1298–1307, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8674772

[17] I. Ullah, Z. Ullah, and J.-A. Lee, "Efficient tcam design based on multipumping-enabled multiported sram on fpga," *IEEE Access*, vol. 6, pp. 19 940–19 947, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8329957

[18] Z. Ullah, "Lh-cam: Logic-based higher performance binary cam architecture on fpga," *IEEE Embedded Systems Letters*, vol. 9, no. 2, pp. 29–32, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7842534

[19] Z. Ullah, K. Ilgon, and S. Baeg, "Hybrid partitioned sram-based ternary content addressable memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 12, pp. 2969–2979, 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6270666/citations#citations

[20] A. Zahir, S. K. Khattak, A. Ullah, P. Reviriego, F. B. Muslim, and W. Ahmad, "Fractcam: fracturable lutram-based tcam emulation on xilinx fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2726–2730, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9217507