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

Yao Chen*, Feng Yu*, Di Wu†, Weng-Fai Wong*, Bingsheng He*
*National University of Singapore, Singapore, †University of Science and Technology of China, Hefei, China
{yaochen, wongf, dcsheb}@nus.edu.sg, yuf@u.nus.edu, ddiwu@mail.ustc.edu.cn

***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 platform for implementing CAMs because of their low latency, reconfigurability, 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 frequency, 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://github.com/Xtra-Computing/DSP_CAM/.**

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

I. Introduction

Content-addressable memory (CAM) performs fast content matching 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 data-intensive applications such as graph analytics, network processing, and database query acceleration [5].

Field-programmable gate arrays (FPGAs) are widely adopted as accelerator platforms for data-intensive applications, driving the demand for CAM designs 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 Lookup Tables (LUTs), Block RAMs (BRAMs), or hybrid designs combining both, but all face significant challenges, especially in scalability and performance, as shown in Figure 1. The main challenge of LUT-based CAMs is on how to achieve very good search performance while efficiently utilizing the LUT resources [7], [18]. 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 (BRAM) [3], [10], but BRAM blocks are designed for sequential access, requiring additional logic for parallel comparisons, which reduces search performance and achievable clock frequency [4], [17]. Hybrid-resource CAMs attempt to balance the flexibility of LUTs with the storage efficiency of BRAMs but encounter complex update processes due to the overhead of managing multiple resource types [4], [9]. The Digital Signal Processing (DSP) slices in FPGAs have been explored as the primary resource for CAM design due to their superior achievable clock frequency compared to LUT and ability to be configured as a logical operation unit. Repurposing the DSP slice for CAM may also help us to reduce the reliance on LUTs and BRAMs, alleviating resource constraints with enhanced performance. The existing DSP-based CAM design has demonstrated benefits, such as higher operating frequency and optimized data pathways. However, it remains challenged by prolonged search latency, rendering it less suitable for data-intensive applications [14]. Moreover, the intrinsic computational demands associated with executing parallel comparisons within CAMs exacerbate existing challenges, rendering these designs suboptimal for data-intensive applications that necessitate scalability, frequent updates, and multiple concurrent query searches.

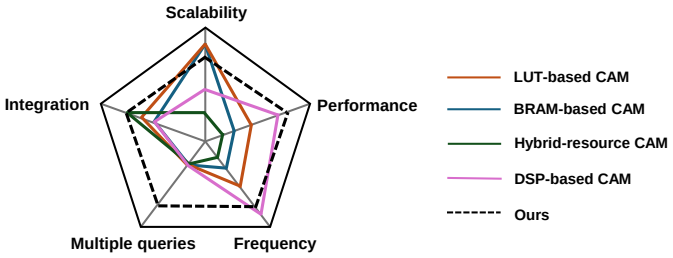

Fig. 1: The Characteristics of current FPGA-based CAM designs. Scalability denotes the achieved CAM size. Performance represents the normalized search and update latency. Frequency is the maximum achievable clock frequency. Integration demonstrates the level of easy integration to an application. Multiple queries stand for the concurrent support to multiple input queries.

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 design on FPGA demonstrates superior performance in different scales.

†This work was done when Di Wu was a visiting student at NUS.

TABLE I: A survey of recent CAM designs on FPGA

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. - indicates the value is not reported in the literature.

- **System case study**: We demonstrate the effectiveness and integrability of our proposed CAM design in the context of triangle counting acceleration system in graphs, showcasing improved efficiency and scalability.

The remainder of this paper is organized as follows. Section II identifies the inefficiencies of current CAM on FPGAs for data-intensive applications and motivates our proposed design. Section III presents our proposed configurable DSP-based CAM architecture. Section IV evaluates our design and demonstrates its advantage over the existing ones. A case study with triangle counting on real-world graphs is presented in Section V and Section VI concludes this paper.

II. Background

CAMs can be categorized into Binary CAM, Ternary CAM (TCAM), and Range-Matching CAM (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 compiled in Table I. FPGAs possess a unique capability to emulate all forms of CAM designs due to their inherent logical representation functionality. However, the pronounced variations in resource types available within FPGAs necessitate categorizing CAM designs based on the primary resources utilized: LUTs, BRAMs, DSPs, and combinations thereof.

A. Challenges When Facing Data-Intensive Applications

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

Limited Scalability. In traditional CAM implementations that adopt LUT and BRAM [10], [13], [15]–[17], as the size of the CAM increases — 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 [4], [10], [16], [17], [20] 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 data processing efficiency. The slow update rates and the complexity of update logic hinder the performance of applications that need immediate reflection of data changes, such as dynamic graph algorithms.

Multiple Concurrent Queries. Multi-queries are 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 incapable 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 commonly necessitate domain-specific architectures. In such contexts, CAM implementations must coexist with other system components to achieve overall accelerator functionality. Consequently, the configurability and integrability of the CAM microarchitecture should be explicitly considered during the design process.

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. By 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. Runtime dynamic data path control in the CAM unit is necessary to address the multiple search requests for data-intensive applications.

Configurable architecture. The variation of the applications necessitates a flexible CAM architecture that can accommodate these differing needs without completely redesigning the entire architecture.

In summary, using DSP blocks as the primary resource in our CAM design effectively addresses resource utilization and scalability issues found in traditional FPGA-based CAMs. Multi-query support adds flexibility to accommodate dynamic data and application demands, enhancing both efficiency and responsiveness. Furthermore, a parameterized architectural design facilitates adaptation for broader application acceptance.

III. Configurable DSP-based CAM Architecture

With the motivations above, our configurable CAM design on FPGA adopts the DSP slice as the primary resource and is designed

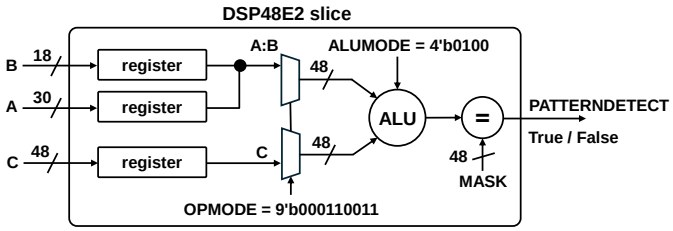

Fig. 2: DSP-based CAM cell.

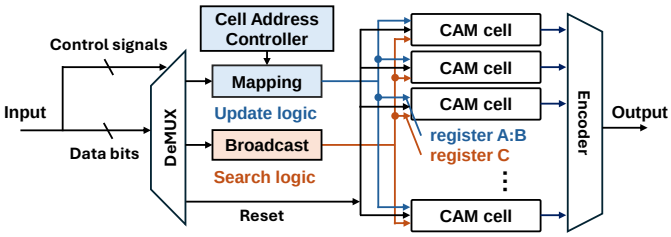

Fig. 3: Architecture of DSP-based CAM block.

hierarchically in a fully parameterized manner. 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, which integrates data update and search logic, serving as the core component for executing essential CAM operations. Our proposed configurable 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.

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, registers A and B in each DSP48E2 slice are concatenated to provide a maximum 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 bit-level granularity of the mask control, 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 update and search operations as it only provides the data storage and comparison logic. The design of a CAM block involves organizing multiple CAM cells along with additional control logic to support search and update operations. Specifically, a CAM block in our design includes a configurable number of CAM cells, a DeMUX, an update and search logic, and an output Encoder, as shown in Figure 3.

The input bus for the CAM block comprises both data bits and control signals that include update, search, and reset. When a new input data packet arrives, the DeMUX routes the data bits to the update logic or the search logic based on the control signals. For update operations, the update logic uses a mapping function based on the Cell Address Controller to direct each data word (with storage data width) in the input data bus to the appropriate CAM cell in the block. Since the input data bits may consist of multiple storage-size data words due to a larger customized input data width than the storage data width in CAM cell, this approach enables the parallel updating of multiple CAM cells in a single operation. For search operations, the redundant bits in the input data bits are masked to ensure that only one data word serves as the search key within the CAM block. The search logic then broadcasts this key to all CAM cells for parallel comparisons. The Encoder collects the match results and generates the final output, supported by a configurable encoding scheme suited to various addressing and management strategies. When the reset signal is asserted, all stored contents in each CAM cell are cleared.

C. CAM unit microarchitecture

The CAM unit consists of multiple CAM blocks, supplemented by additional control and routing logic to facilitate multi-query operations. As illustrated in Figure 4, the overarching microarchitecture includes a Routing Compute module and a Post-Router module, alongside multiple CAM blocks. These components are encapsulated by input and output interfaces that communicate with the user kernel.

1) CAM Group: We define a CAM group as a logical abstraction that forms the foundation for executing search and update operations within the CAM unit. A CAM group consists of multiple CAM blocks and is not tied to the physical layout, enabling flexible configuration and dynamic reassignment of resources. Let N_{total} denote the total number of CAM blocks in the CAM unit and M the number of CAM groups. Each group then contains $N = \frac{N_{\text{total}}}{M}$ blocks; M needs to be divisible by N. The number of CAM groups M is configured at runtime by the user kernel, thereby enabling the CAM unit to dynamically adapt to varying workload demands, specifically catering to multi-query parallelism requirements.

2) Update Process: During the update process, the Routing Table, implemented as an array within the Routing Compute module, stores the mapping relationships between Block IDs and Group IDs. This

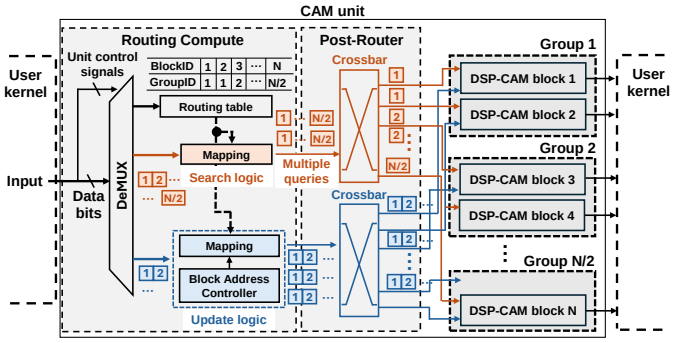

Fig. 4: Architecture of DSP-based CAM unit.

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.
	Block Bus Width	Data path width in CAM block.
	Result Encoding	Encoding scheme for search results from multiple CAM cells.
CAM Unit	Unit Size	Number of CAM blocks per CAM unit.
	Unit Bus Width	Data path width in CAM unit.

array shares the same data path as the input update data, where a control signal is used to update the mapping configuration. Based on the mapping relationship defined in the Routing Table, the input data is replicated and directed to all M CAM groups, ensuring that the update is applied globally. Subsequently, the update crossbar in the Post-Router module routes the replicated data to the designated block in each group. Within each CAM group, the Block Address Controller sequentially assigns data to each block using a round-robin policy; once the current block is full, the controller points to the next block in the group.

3) Search Process: In the search process, each incoming search key is first processed by the Routing Compute module, which employs a mapping function based on the Routing Table to allocate the key to the appropriate CAM group. Once a group is selected, the search key is replicated N times (where N is the number of blocks within that group) and broadcast to every block in the group, thereby enabling parallel comparison across all blocks. The Post-Router module then delivers the duplicated keys to the corresponding blocks, initiating parallel search operations. With each search key assigned to a distinct CAM group, the overall CAM unit is capable of supporting up to M search queries per cycle.

4) Example: Consider the configuration shown in Figure 4, where each CAM group consists of two blocks. During the update process, each incoming data packet is duplicated across all $M = N_{\text{total}}/2$ groups and written sequentially: data is first stored in the first block of each group, and upon reaching capacity, subsequent entries are directed to the second block following a round-robin schedule. In the search phase, the CAM unit supports up to M concurrent search keys, each routed to a distinct CAM group. Within each group, the key is broadcast to both blocks, enabling parallel comparison and efficient multi-query processing.

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.

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 an 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, an FPGA platform based on the UltraScale+ architecture. The U250 features four DDR4 memory channels and a PCIe Gen3 x16 interface, providing ample external memory bandwidth for data-intensive applications. The resources of the chip are collected in Table IV. The results are collected through implementation using Xilinx Vivado Design Suite v2021.2, and the CAM architecture was implemented and deployed on the U250. Resource utilization metrics include LUTs, BRAMs, and DSPs.

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-based 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 DSP 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.

b) Comparison to the state-of-the-arts: We incorporate our design with its maximum configuration into Table I, which includes state-of-the-art designs at their respective maximum configurations. It is worth noting that 4 BRAMs are utilized by the bus interfaces for FIFOs, which we add to facilitate complete synthesis and implementation. Our CAM design exhibits superior scalability, occupying 79.25% of the on-chip DSP resources while utilizing significantly fewer LUTs. Additionally, when compared with existing DSP-based CAM designs, the update and search latency of our proposed CAM is well-balanced and more conducive to data-intensive applications.

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

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. The most commonly used method is the merge-based method, which processes one comparison per cycle and updates the list indices based on the comparison results, achieving a time complexity of $O(m+n)$. 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 $O(n)$.

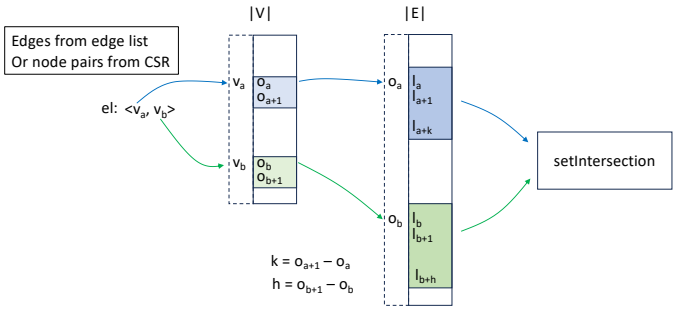

Fig. 5: Simplified process of triangle counting on graphs.

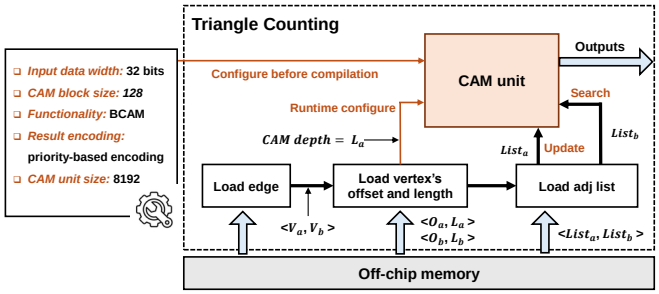

Fig. 6: System architecture of CAM-based triangle counting.

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 storage data width is set to 32 bits, the CAM cell is configured to be binary, the block size is set to 128, the system input bus width is 512, and 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 adjacent lists 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 $(List_a)$ is determined and loaded into the generated CAM unit, and the vertices from the shorter one $(List_b)$ are used as the search keys. The number of groups is decided by the length of the longer list. 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

Since our CAM design provides the grouping mechanism, the adjacent list is duplicated in all groups, enabling a parallel search of the M nodes from the other list. To note here, a list with a length less than 128 occupies the entire CAM block for easy implementation. 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 implemented inside a single SLR.

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

Dataset	Triangles	Ours	Baseline [1]	Speedup
facebook_combined	1,612,010	5.054	18.7	**3.70x**
amazon0302	717,719	23.086	89.5	**3.88x**
amazon0601	3,986,507	71.210	230.3	**3.23x**
as20000102	6,584	0.422	7.4	**17.54x**
cit-Patents	7,515,023	415.808	800.0	**1.92x**
ca-cit-HepPh	195,758,685	1,526.05	5,361.1	**3.51x**
roadNet-CA	120,676	62.058	108.8	**1.75x**
roadNet-PA	67,150	34.559	88.7	**2.57x**
roadNet-TX	82,869	42.323	96.8	**2.29x**
soc-Slashdot0811	551,724	29.402	259.7	**8.83x**

D. Result Analysis

The performance evaluation of our triangle counting accelerator across various graph datasets [11] is summarized in Table IX. The baseline design [1], representing the current state-of-the-art FPGA-based approach for triangle counting, utilizes a fine-grained pipeline optimized to minimize pipeline bubbles. Nevertheless, its overall performance remains limited due to the merge-based set intersection method, which inherently enforces sequential processing. In contrast, our CAM-based accelerator significantly reduces this sequential bottleneck by enabling parallel execution of set intersection operations, achieving an average speedup of $4.92\times$ compared to the baseline. These results clearly demonstrate the efficacy of our parallel CAM-based approach in overcoming the limitations imposed by sequential set intersection methods.

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 demonstrate the superior performance and scalability of the architecture, with a triangle counting case study showcasing 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://github.com/Xtra-Computing/DSP_CAM/.

Acknowledgment

This work is supported in part by the Ministry of Education AcRF Tier 2 grant, Singapore (MOE-T2EP20121-0016), and a Google South & Southeast Asia Research Award. This work is also supported in part by AMD under the Heterogeneous Accelerated Compute Clusters (HACC) program.

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