SCIENCE CHINA
Information Sciences

• **Supplementary File** •

CATCAM: A 28 nm Constant-time Alteration TCAM Enabling Less than 50 ns Update Latency

Chenchen DENG[1], Tianzhu XIONG[2], Zhaoshi LI[3], Zhiwei LIU[3], Yao WANG[3], Jianfeng ZHU[3], Jun YANG[2], Shaojun WEI[3] & Leibo LIU[3*]

[1]*Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing,* 100084*, China;*
[2]*National ASIC Center, School of Electronic Science and Engineering, Southeast University, Nanjing,* 210096*, China;*
[3]*School of Integrated Circuits, Tsinghua University, Beijing* 100084*, China*

Appendix A Motivation

Ternary Content-Addressable Memory(TCAM) is an indispensable component of lookup tables in switches or routers due to its matching flexibility and parallel search capability [1]. However, TCAM in today's hardware switches only supports around 40 to 50 rule updates per second [2], and network packets may drop or be forwarded incorrectly during rule installation latency up to hundreds of milliseconds [3]. The main reason is that TCAM stores rules from top to bottom in decreasing order of priority for disambiguation [4], i.e., a rule located at a higher physical address has higher priority. Figure A1(a) shows how rules ($R_0 \sim R_3$) are arranged physically. Therefore, when multiple matches occur and a single highest priority match is required, the entry with the highest address is selected (R_2 in our example). However, per-rule update time in a TCAM grows linearly with the number of inserted rules. As shown in Figure A1(b), rule insertion could lead to a substantial amount of moves for existing entries in TCAM to maintain priority ordering [5]. Thus, updating a TCAM is similar to the insertion sort that takes $O(n)$ time, where n is the number of inserted rules. Meanwhile, it considerably increases the energy consumption by several write operations. In this paper, a Constant-time Alteration Ternary CAM (CATCAM) that can accomplish both lookup queries and update requests in nanoseconds is proposed. It decouples rule priorities from physical addresses by augmenting traditional TCAM with a priority matrix which encodes the priority ordering between entries separately in an 8T SRAM array. To filter out the entry with the highest priority during lookup, logical operations required by traversal of the priority matrix is realized through Computing-In-Memory technique happened in-situ on bit-lines. Therefore, incoming rules can be written to any empty row in a conventional TCAM without the shuffles of existing entries.

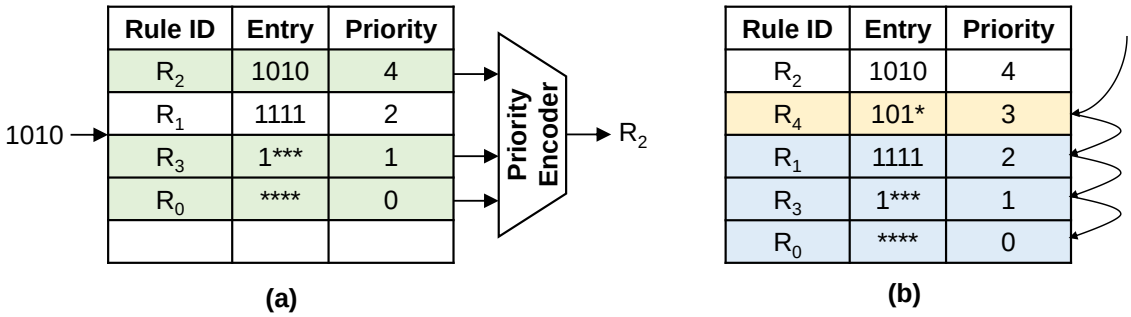

Figure A1 (a) The wildcard rules are prioritized in a TCAM for lookup. Note that only the entry field is stored. (b) The insertion of a new rule causes existing entries to shuffle.

Appendix B Hierarchical Scalability

CATCAM features a hierarchical architecture to scale out. Lookup queries can be fully pipelined and the scheduling logic guarantees deterministic update latency, not subject to the number of existing rules or ruleset characteristics. To scale out, large lookup tables are partitioned into multiple sub tables so that each of them can be fitted into a match segment. Since the rule priority is generally specified as a finite integer, the entire range of rule priorities can be segmented into non-overlapping intervals. Each match matrix stores rules whose priority belongs to a certain interval to establish the priority ordering between match matrices because all the rules stored in one of them have higher (or lower) priority than another. It is encoded in a global priority matrix using the same scheme and in-memory implementation (16×16 array) as the local priority matrix. Similarly, the priority ordering is not related to the physical positions of match matrices. Search is performed hierarchically with the arbiter and is composed of three stages:

* Corresponding author (email: liulb@tsinghua.edu.cn)

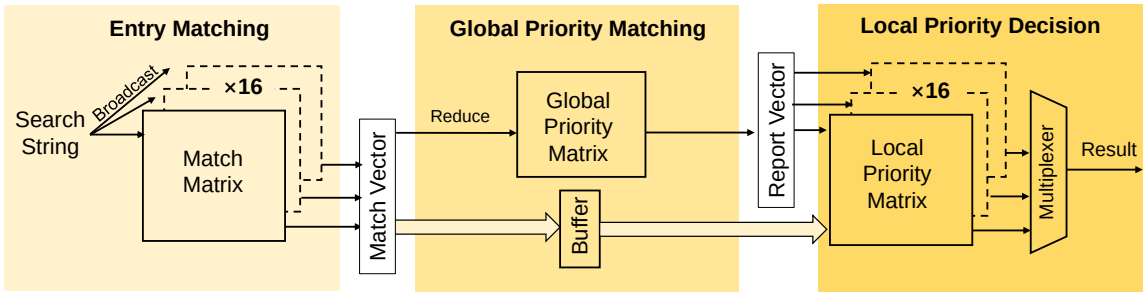

Figure B1 The three-stage pipeline for lookup queries.

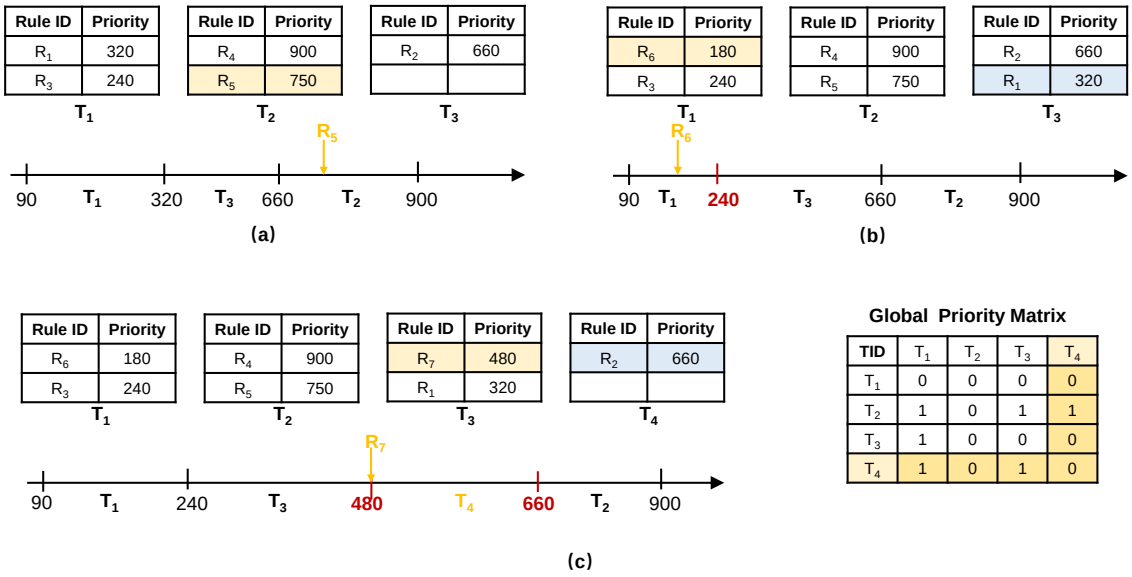

Figure C1 The demonstration of interval-based rule insertion.

entry matching, global priority decision, and local priority decision. The idea is that the highest priority matched entry must be among the highest priority match matrix with matched entries. Upon receiving a lookup query, the task allocator broadcasts the search string to each match matrix to obtain the local match vector. These local results undergo reduction OR gates to form the global match vector whose ith bit is "1" indicates there exist match entries in the ith match matrix. At the second stage, the global match vector is fed to the global priority matrix to generate the one-hot global report vector which indicates the match matrix holding the target entry. The third stage proceeds with the local priority decision process in the chosen match segment and the multiplexer selects the corresponding report vector to output.

Entry matching for the current search string can be overlapped with global priority decision for the preceding global match vector which can be further overlapped with local priority decision for the previous local match vector. A three-stage pipeline is therefore designed, as shown Figure B1. Although in theory the local priority matrix and the global priority matrix can be traversed simultaneously, it leads to redundant local priority decision since only one match segment needs to report. In favor of energy consumption, only one local priority matrix is traversed after global arbitration and does not affect the overall throughput due to the pipeline. Meanwhile, the local match vector is buffered to make room for the following entry matching.

Appendix C Interval-based Rule Insertion

Lookup tables are not static. To maintain the priority ordering between match matrices while updating the lookup table, the scheduling logic inserts rules based on dynamic intervals. These non-overlapping intervals are segmented by the highest priority rule in each match matrix and recorded in the metadata register. It also keeps the availability flag indicating the vacancies in each match matrix and the successor pointer to the match matrix whose priority interval follows. Upon receiving an update request, if it is an insertion, the priority of the new rule is extracted and compared against all priority intervals to locate its position, the associated match matrix is the one to be inserted. If the availability flag of the current match matrix is true, the first available entry according to its valid vector is allocated for the new rule, as shown in Figure C1(a). Writes to the match matrix and the priority matrix can be performed concurrently since they are two separate memory arrays. The valid vector is modified and undergoes reduction OR to update the availability flag.

If the availability flag is false, the highest priority rule in the current match matrix is relocated to make room for the new rule, whose address is maintained in the metadata register. To maintain the priority ordering between match matrices, it should be relocated to the match matrix whose priority interval follows assuming it is available. To this end, this entry is read from the match matrix along with its priority from the priority store and reinserted into the following match matrix designated by the successor pointer. The new rule can thus be inserted into the current match matrix and replace the relocated rule, as shown in Figure C1(b).

Table D1 Prototype chip specifications.

Technology	28nm
Supply Voltage(Vdd_Low)	0.6-1V(0.3-0.6V)
Frequency	470MHz@0.9V
Die Size	$2.1 \times 2.7mm^2$
Matrix Size	Match: $0.17\times0.39mm^2$ Priority: $0.13\times0.34mm^2$
Configuration	256×160b×16(640Kb)
Power Consumption	0.35W@0.9V

Table D2 Prototype chip specifications.

Task	Cycles	Latency	Throughput	Energy
Search	7	14.9 ns	470 MOPS	752 pJ
Insertion w/o Relocation	11	23.4 ns	43 MOPS	271 pJ
Insertion w/ Relocation	16	34 ns	29 MOPS	414 pJ
Insertion New Match Segment	19	40.3 ns	25 MOPS	477 pJ
Deletion	2	4.3 ns	235 MOPS	42 pJ

To locate the new highest priority rule, the valid vector is applied to the priority matrix as if all valid entries are matched so that the output report vector indicates the one with the highest priority. Its address and associated priority intervals in the metadata register are updated accordingly.

However, if the availability flag of the successor match matrix is false, when the priority ordering between match matrices is fixed, the above situation may iterate between neighboring match matrices until an available entry is found. Similar to rule insertion in a single TCAM, such overhead can be reduced with the flexibility of the global priority matrix. To accommodate the relocated rule, a new match segment is assigned with available memory resources. Its priority ordering should be between the current match matrix and its following match matrix. To generate updates to the global priority matrix, the priority of the relocated rule (which would become the highest priority rule after insertion since the match matrix is empty) is compared against those of other match matrices from the metadata register. In addition, the successor pointer of the current match matrix is redirected to the new match matrix and the successor pointer of the new match matrix is directed to the original successor of the current match matrix. The priority interval of the current match matrix is therefore segmented into two and the boundary is determined by the new highest priority rule of the current match matrix, as shown in Figure C1(c). The new match segment would accommodate future rule relocations as well as rule insertions and further adjust the boundary in-between.

Appendix D Measurement results

The prototype chip of CATCAM is implemented in the 28 nm CMOS technology as a proof of concept. The chip specifications are listed in Table D1. A single match matrix is 170×390 μm^2 with 86% area efficiency and a single priority matrix is 130×340 μm^2 with 80% area efficiency. The peripheral logic accounts for 22% area (excluding test circuitry) and the priority store contributes to 80% of it due to the register files. Table D2 summarizes the performance of various lookup table tasks ranging from lookup, deletion to all three types of insertion. Since memory operations dominate the latency of each task, the set operating frequency of 470 MHz leaves enough margin for peripherals and interconnection. Figure D1 shows the measured frequency and energy efficiency of the priority matrix and the search operation across different supply voltages. At 0.9 V, the average frequency of 470 MHz results in 1.14 fJ/bit per search. The best energy efficiency is achieved at 0.6 V and 118 MHz, resulting in 0.5 fJ/bit per search. Compared to the match matrices, the priority matrices and the peripheral logic incur 16% energy overhead and 75% area overhead. Note that the overhead is associated with the number of entries in a match matrix, not the width of the entry, which means it could be further lowered with wider rules. Although the prototype chip only implements 160b width, it could be extended to 640b comparable to a commercial off-the-shelf TCAM, and is estimated to incur 4% energy overhead and 20% area overhead. The capacity of the prototype chip is limited by the die size, and if the area permits, the number of match segments can be scaled up to 256. In this way, the CATCAM may have a capacity of 10Mb similar to a commercial off-the-shelf TCAM [6]. The simulated results of this design show the number of compare operations could reach 31 TOPS while consuming 5.6W. Figure D2 shows the shmoo plot and frequency of the column-wise write for the priority matrix with Vdd and Vdd_Low swept. The column-wise write achieves a maximum frequency of 480 MHz when Vdd_Low = 0.5 V, with an operational margin of 200 mV.

Experimental results show that the proposed approach outperforms state-of-the-art TCAM update methods by four to six orders of magnitude. Current methods to incremental TCAM updates (RuleTris [7], FastRule [8], COLA [9], FastUp [10]) exploit the minimum dependency graph to decrease entry movements for rule insertion. Although they manage to minimize the number of TCAM moves, their performance is hindered by the prohibitive time spent on the firmware to calculate the update sequence. The reason is that they are still based on the premise that a TCAM prioritizes rules by physical addresses. Constructing and maintaining the dependency relationship with a graph is complicated and time-consuming, especially for large rulesets. Besides, the average latency of a TCAM move is relatively slow (approx. 0.6 ms). On the contrary, CATCAM schedules newly inserted rules based on priorities rather than dependencies, eliminates the firmware overhead and guarantees deterministic update latency that is not subject to the number of existing rules or ruleset characteristics. Since there is no hardware implementation of above mentioned TCAM update scheme, Table D3 compares the proposed CATCAM with conventional TCAM designs. Note that the match matrix is our in-house alternative to the conventional TCAM which is comparable to existing TCAM designs [6, 11, 12]. The proposed CATCAM trades some certain area and energy overhead for flexible updates without affecting search performance and achieves a speedup by several orders of magnitude in update time. More importantly, the concept of CATCAM is also compatible with existing TCAMs, more area/energy-efficient TCAM designs could be utilized to further reduce such overhead.

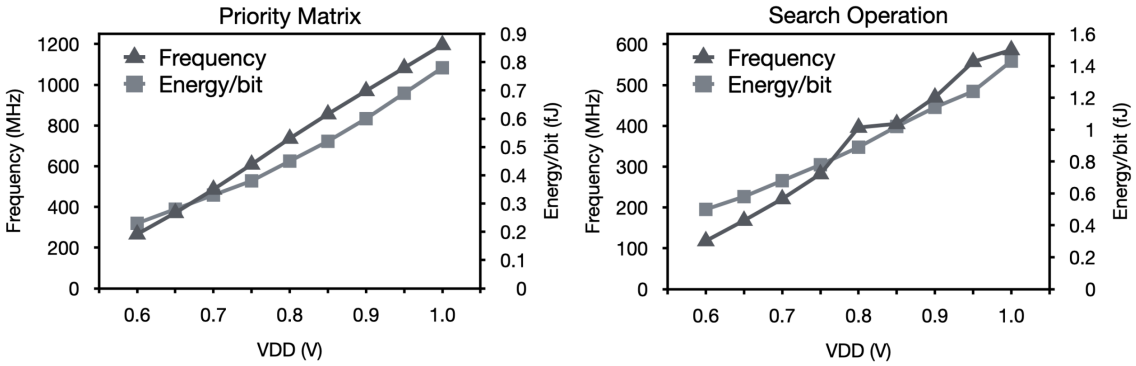

Figure D1 Measurement results.

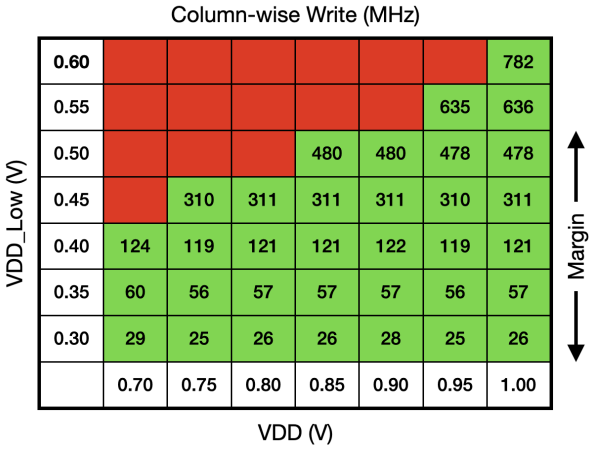

Figure D2 Shmoo plot of the column-wise write with Vdd and Vdd_Low swept.

Table D3 Comparisons with existing TCAM designs.

	[6]	[11]	[12]	This work
Technology	28nm	28nm	28nm	28nm
Bit cell	16T	12T	32T	16T
Area/cell	0.625 μm^2	0.304μm^2	2.65μm^2	0.79 μm^2
Frequency	400MHz	370MHz	2560MHz	470MHz
Energy/search	2.02fJ/bit	0.74fJ/bit	0.42fJ/bit	1.14fJ/bit
Array size	4k× 80	32× 64	32× 64	256× 160

References

1 Pagiamtzis K,Sheikholeslami A. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. IEEE Journal of Solid-State Circuits, 2006, 41:712–727.
2 Katta N, Alipourfard O, Rexford J, et al. Cacheflow: Dependency-aware rule-caching for software-defined networks. In: Proceedings of the Symposium on SDN Research, 2016.1–12.
3 Kuzniar M,Peresini P, Kostic D. What you need to know about SDN flow tables. In: Proceedings of International Conference on Passive and Active Network Measurement, 2015. 347–359.
4 Shah D, Gupta P. Fast updating algorithms for TCAM. IEEE Micro, 2001,21:36–47.
5 Song H, Turner J. Nxg05-2: Fast filter updates for packet classification using TCAM. In: Proceedings of IEEE Globecom, 2006.1–5.
6 Nii K, Amano T, Watanabe N, et al. A 28nm 400MHz 4-parallel 1.6 Gsearch/s 80MB ternary CAM. In: Proceedings of IEEE International Solid-State Circuits Conference(ISSCC), 2014. 240–241.
7 Wen X, Yang B, Chen Y, el al. Ruletris: Minimizing rule update latency for TCAM-based SDN switches. In: Proceeding of IEEE 36th International Conference on Distributed Computing Systems (ICDCS). 2016. 179–188.
8 Qiu K, Yuan J, Zhao J, et al. Fastrule: Efficient flow entry updates for TCAM-based openflow switches. IEEE Journal on Selected Areas in Communications, 2019, 37:484–498.
9 Zhao B, Li R, Zhao J. Efficient and Consistent TCAM Updates. In: Proceeding of IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020. 1241-1250
10 Wan Y, Song H, Che H, et al. FastUp: Fast TCAM Update for SDN Switches in Datacenter Networks. In: Proceeding of

IEEE 41st International Conference on Distributed Computing Systems (ICDCS). 2021. 887–897.
11 Jeloka S, Akesh N, Sylvester D, et al. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6t bit cell enabling logic-in-memory. IEEE Journal of Solid-State Circuits,2016, 51: 1009–1021.
12 Fan X, Meyer N and Gemmeke T. Compiling all-digital-embedded content addressable memories on chip for edge application. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2022,41:2560-2572.