CA-RAM: A High-Performance Memory Substrate for Search-Intensive Applications

Sangyeun Cho, Joel R. Martin, Ruibin Xu, Mohammad H. Hammoud, and Rami Melhem
Department of Computer Science
University of Pittsburgh

{cho,pyrun,xruibin,mhh,melhem}@cs.pitt.edu

Abstract

This paper proposes a specialized memory structure called CA-RAM (Content Addressable Random Access Memory) to accelerate search operations present in many important real-world applications. Search operations can occupy a significant portion of total execution time and energy consumption, while posing a difficult performance problem to tackle using traditional memory hierarchy concepts. In essence, CA-RAM is a direct hardware implementation of the well-known hashing technique. Searchable records are stored in CA-RAM at a location determined by a hash function, defined on their search key. After a database has been built, looking up a record in CA-RAM typically involves a single memory access followed by a parallel key matching operation. Compared with a conventional CAM (Content Addressable Memory) solution, CA-RAM capitalizes on dense SRAM and DRAM designs, and achieves comparable search performance while occupying much smaller area and consuming significantly less power. This paper presents detailed design aspects of CA-RAM, to be integrated in future general-purpose and application-specific processors and systems. To further motivate and justify our approach, we present two real examples of using CA-RAM to build a high-performance search accelerator targeting: IP address lookup in core routers and trigram lookup in a large speech recognition system.

1 Introduction

High-performance general-purpose and embedded processor architectures have been continuously reshaped by both technology advances and ever-changing application requirements [8]. Unprecedented technology advances have enabled billion-transistor chip implementations [4], while presenting new design challenges such as elevated power density and a delay scaling discrepancy between transistors and global wires. At the same time, the computational and power consumption requirements become more stringent as complex and large applications, such as RMS (Recognition, Mining and Synthesis) and other high-performance embedded applications keep gaining momentum [6].

To effectively tackle the performance-power requirements of important applications, integrating *specialized hardware* on chip has become a clear design trend. For example, special-purpose hardware for graphics processing and network packet processing has already been integrated in a processor chip [2]. Hardware functions for speech recognition, natural language processing, and advanced image processing are seriously being considered for inclusion in future processor chips. There is a clear advantage of implementing a specific task in custom hardware; it is often two to three orders of magnitude more power-efficient than a general-purpose processor [3]. Given the critical benefit of specialized hardware and the ample transistor budget provided by an advanced process technology, this design trend will continue and attain more importance.

This paper presents a specialized, yet generic memory structure called CA-RAM (Content Addressable Random Access Memory) to accelerate *search operations*. Searching is a fundamental computer operation used in many nontrivial applications [14], often occupying a large portion of the total program execution time. Network packet filtering and routing applications, for example, require constant, high-bandwidth searching over a large number of IP addresses. Speech recognition applications spend over 24% of their CPU cycles dedicated to searching [9]. Traditionally, searching has been done through software algorithms such as list/tree traversal and hashing [14] or hardware schemes such as CAM (Content Addressable Memory) [15].

This work was supported in part by DARPA/IPTO in the Architectures for Cognitive Information Processing (ACIP) program, under the contract number FA8750-04-C-0266.

The basic idea of CA-RAM is simple; it implements the well-known hashing technique in hardware. CA-RAM uses a conventional high-density memory (*i.e.*, SRAM or DRAM) and a number of match logic blocks to provide parallel search capability. Records are pre-classified and stored in the memory so that given a search key, access can be made accurately on the memory row having the target record. Each match logic block then extracts a record key from the fetched memory row, usually holding multiple candidate keys, and determines if the record key in consideration is matched with the given search key.

CA-RAM achieves lower latency and higher search bandwidth compared with software techniques, while eliminating undesirable architectural artifacts such as cache pollution. A conventional search operation typically involves multiple memory accesses following a pointer-chasing pattern, which is difficult to fully optimize [12]. CA-RAM provides a similar search capability compared to CAM; however, its decoupled match logic can be easily extended to implement more advanced functionality such as massive data evaluation and modification. More importantly, the bit-density of CA-RAM is much higher than that of CAM, nearly five times if DRAM is used in the CA-RAM implementation. Furthermore, from the viewpoint of system design, CA-RAM can be easily plugged into the memory hierarchy as a regular address-based random access memory.

Due to *parallel matching*, a CA-RAM slice and a set-associative cache bear similarity in their hardware structure. However, the required and supported operations for CA-RAM and for caches are different. CA-RAM has explicit operations for data management based on possibly non-trivial hashing functions and can support massive data evaluation and update, while a cache supports only regular memory operations such as loads and stores. The goal of a CA-RAM design is to achieve *full content addressability* on a large database without the cost of exhaustively implementing hardware match logic for each memory element (as in CAM), by utilizing the general idea of hashing and a limited hardware support for parallel matching.

Our work described in this paper makes several important contributions in the area of associative memory design and its application. First, we introduce the notion of completely decoupling memory array and match logic, which promotes using best conventional memory designs and incorporating powerful data processing features. Second, we present a prototype CA-RAM design based on a standard design flow using a high-level hardware description language, identifying further design issues. Third, we give two detailed application studies using real-world data sets as an example use of CA-RAM showing the efficacy and feasibility of the proposed approach.

The rest of this paper is organized as follows. As a background, Section 2 presents a brief discussion on the software-based hashing and the hardware-based CAM techniques. Section 3 gives a detailed description of the CA-RAM structure. Basic design considerations, system interface issues, and performance/cost/power issues are discussed in detail. Two real applications are studied in Section 4, namely, IP address lookup in Internet core routers and trigram lookup in a speech recognition system. Other related works are summarized in Section 5. Finally, concluding remarks are given in Section 6.

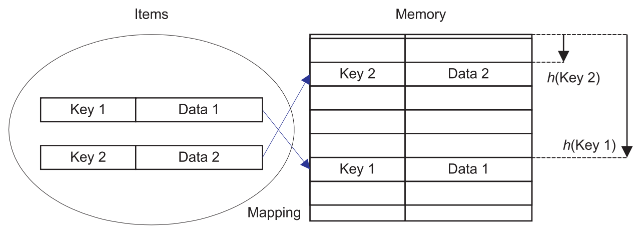


Figure 1. Mapping of data items to table entries using a hash function $h(\cdot)$ [15].

2 Background

2.1 Software-based hashing technique

Although there are many software-based searching techniques, such as linear searching, binary tree searching and ordered table searching [14], we focus in this subsection on the *hashing* technique because it often offers higher performance than other techniques, especially when a large database is used [15]. In addition, the use of hashing directly forms the foundation for our CA-RAM approach.

Searchable *data items* (or *records*) contain two fields: *key* and *data*. In general, the goal of searching is to find a record associated with a key K in the database. Hashing achieves fast searching by providing a simple arithmetic function $h(\cdot)$ (hash function) on K so that the location of the associated record in memory is directly determined, as shown in Figure 1. The memory containing the database can be considered a two-dimensional table (hash table) of M entries. Each table entry, also called a *bucket*, often holds multiple records, arranged in an array or chained in a linked list. If each bucket holds up to S records, the total size of the hash table in bytes is $(M \times S \times R)$, where R is the size of a record in bytes. If the database has N records, clearly, $(M \times S)$ should be equal to or preferably larger than N.

Given M buckets, the hash function gives at most M different values, with $0 \leq h(K) < M$, for all keys K. It is possible that two distinct keys $K_i \neq K_j$ hash to the same value $h(K_i) = h(K_j)$. Such an occurrence is called *collision* (or *conflict*). When there are too many ($> S$) conflicting records for an S-entry bucket, some records must be placed somewhere other than the bucket that the records hash into, called *overflow area*. Overflow occasions are usually handled by reserving some place for spilled entries which is directly derivable from the place of collision. For example, locations with consecutive hash addresses (*i.e.*, buckets following the overflowing bucket) may be tried until a bucket with an empty record slot is found. Instead of this linear probing method, one can apply a second, alternative hash function to find a bucket with empty space. To obtain best performance out of the given hash table capacity, it is obvious that one must find a hash function which minimizes the probability of collision.

Given a database of N records and an M-bucket hash table, the average number of hash table accesses to find a record is heavily affected by the choice of $h(\cdot)$, S (the number of slots per bucket), and the *load factor* α, defined as $N/(M \times S)$. With a smaller α, the number of average hash table accesses can be made smaller, however at the expense of more unused memory space. α poses an important design trade-off for any hash-based searching scheme: area (*i.e.*, cost) versus search latency (*i.e.*, performance). It is further noted that when $(M \times S)$ is fixed, one can potentially reduce the number of collisions by increasing S (and decreasing M).

The steps of hash-based searching are: (1) generating hash address using $h(\cdot)$ given a search key K, (2) retrieving record(s) in the bucket at the generated hash address $h(K)$, (3) comparing K with the retrieved records' key to find a match, and (4) accessing alternative memory locations if the original bucket is full and the matching record has been placed in an overflow area. A proper implementation of each step yields the average search time $O(S)$.

2.2 Content-Addressable Memory (CAM)

CAM is a specialized memory used in high-speed search applications [15, 25]. Unlike traditional random-access memory (RAM) where the user provides an address and the RAM returns the data (content) stored at the address, CAM returns the location (*i.e.*, address) of a data when the data is presented by the user as an input. CAM searches its entire memory to match the input data ("search key") with the set of stored data ("stored keys"). When there are multiple entries that match the search key, a *priority encoder* will choose the highest-priority entry. Often, the returned search result is used to access a separate RAM which holds the data items associated with the stored keys.

To search the entire memory quickly, CAM employs separate match logic per each memory row, as is depicted in Figure 2. When a search key is presented by the user, each search key bit will become visible to all the match logic cells in a column at the same time. Each row of match logic cells will then perform a bit-by-bit comparison between the provided search key and the stored key in the associated memory row. Comparison results will be reduced to a binary value (*e.g.*, 1 for match and 0 for mismatch) and are fed into a priority encoder, which finally produces the search result. Since comparisons for all the memory rows are performed in parallel, CAM achieves high-bandwidth, constant-time search performance.

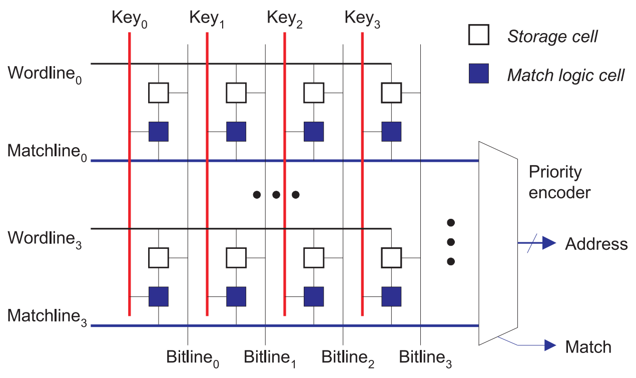


Figure 2. A simplified view of a 4$\times$4 CAM.

TCAM (Ternary CAM) has become a popular memory device with the booming Internet proliferation because of its ability to allow *don't care* bits in a search [7, 29, 32]. A typical TCAM storage cell comprises two bits to encode three symbols, *zero* ("0"), *one* ("1"), and *don't care* ("X"), adding flexibility to the search. For example, a stored key of "110XX" will match any of the four search keys "11000", "11001", "11010", and "11011". The TCAM's ability to perform this *ternary search* is especially useful for implementing IP address lookup, since search mask bits are often associated with each IP prefix.

Despite its desirable behavior, there are two critical disadvantages inherent in the CAM approach: cost and power consumption. Since match logic is contained in each memory row, the maximum achievable cell density of CAM is much lower than that of a conventional RAM. More than an order of magnitude lower storage cell density has been reported [20,24]. Moreover, since all the match logics operate in parallel on a search, required circuit activities grow in proportion to the number of CAM entries, resulting in high power consumption.

3 Content-Addressable Random Access Memory (CA-RAM)

The three main design goals for CA-RAM are: (1) search performance (*i.e.*, bandwidth and average latency) equivalent to that of conventional CAM; (2) area and power characteristics similar to those of conventional RAM; and (3)

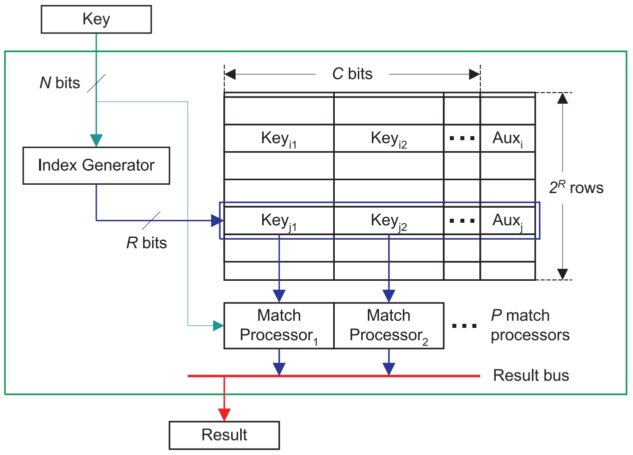


Figure 3. A CA-RAM slice.

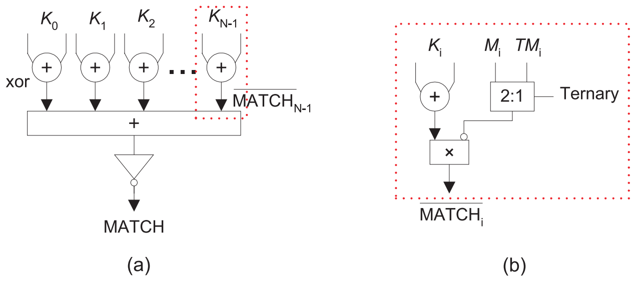


Figure 4. (a) An N-bit match logic. (b) A single-bit match logic element, extended with two "don't care" or mask inputs, M_i and TM_i.

straightforward and efficient processor/system integration. This section presents a detailed description of CA-RAM and discusses how the design goals can be achieved.

3.1 Baseline CA-RAM architecture

A CA-RAM slice takes as an input a search key and outputs the result of a lookup. Its main components include an *index generator*, a memory array (either SRAM or DRAM), and P *match processors*, as shown in Figure 3. For the clarity of presentation, we will for now assume that the CA-RAM memory array contains only the key portion of stored records, as in a conventional CAM. That is, the data portions are assumed to be in a separate (data) memory array.

Given that the search key is N bits long, each C-bit row (*i.e.*, bucket) can store up to $\lfloor \frac{C}{N} \rfloor$ keys. If the number of rows equals to 2^R, an R-bit index is needed to select a row to access. The size of the memory array is $2^R \times C$ bits. Optionally, each row can be augmented with an *auxiliary field*, which is to provide information on the status of the associated bucket. For example, if the bucket had overflows and accordingly some records have been spilled to subsequent buckets, this field can keep a number indicating how far the extended search effort should reach. The location of an available key slot in the bucket can be also stored in the auxiliary field to aid proper record insert/delete operations.

The task of the index generator is to create an R-bit index from an N-bit key input ($R \leq N$). The actual function of the index generator will highly depend on the target application. In many applications, index generation is as simple as bit selection, incurring very little additional logic or delay. In other cases, simple arithmetic functions, such as addition or subtraction, may be necessary. Depending on the application requirements, a small degree of programmability in index generation can be employed.

Once the index is generated from the input key, the memory array is accessed and $\lfloor \frac{C}{N} \rfloor$ candidate keys are fetched simultaneously. The match processors then compare the candidate keys with the search key in parallel, resulting in constant-time matching. It is desirable that $P = \lfloor \frac{C}{N} \rfloor$; however, if N is a flexible parameter to make CA-RAM dynamically adjustable for keys of different sizes, it is possible that $P \neq \lfloor \frac{C}{N} \rfloor$. When $\lfloor \frac{C}{N} \rfloor \leq P$, matching of all the keys can be done in one step. Otherwise, necessary matching actions can be divided into a few pipelined actions.

Each match processor performs comparison quickly using an N-bit hardware comparator, as depicted in Figure 4(a). In essence, the N-bit comparator reduces the result of N single-bit comparators. By extending each single-bit comparator with two "don't care" conditions, CA-RAM can efficiently support both *search key bit masking* (*i.e.*, search with don't care bits in search key) and *record key bit masking* (*i.e.*, search with don't care bits in individual record key, also known as ternary search). To support searching with don't care bits, the N-bit input key can be extended with an N-bit mask, which is passed along to the match processors. To support ternary search, each key in the memory array can be associated with an N-bit mask to denote which bits in the stored key are don't care bits. As such, the number of records that can fit in a given CA-RAM will be halved when the ternary search capability is enabled, since each single symbol in the stored key (one of $\{0, 1, \mathrm{X}\}$) requires two bits to encode. Figure 4(b) shows how a single-bit comparator is extended to enable matching with don't care bits.

A large area saving in CA-RAM comes from decoupling memory cells and match logic. Unlike conventional CAM where each individual row in memory array is coupled with its own match logic, CA-RAM separates dense memory array and common match logic (*i.e.*, match processors) completely. Since the match processors are simple and lightweight, the overall area cost of CA-RAM will be close to that of the memory array used. At the same time, by performing a number of candidate key matching operations

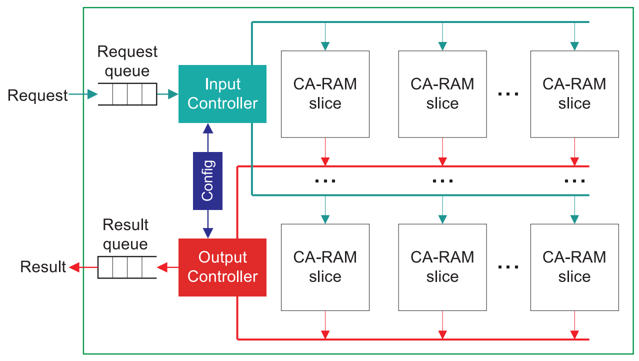


Figure 5. A CA-RAM-based memory subsystem.

in parallel, low-latency, constant-time search performance is achieved. We will further examine the related area and power issues in the following subsections.

3.2 Constructing a memory subsystem using CA-RAM

Based on the CA-RAM slice, a high-performance memory subsystem comprising multiple CA-RAM slices can be constructed, as in Figure 5. A CA-RAM memory subsystem provides two distinct *views* or *modes* to applications. First, it can be viewed as addressable random access memory ("RAM" mode). Second, it can be viewed as searchable, content addressable memory ("CAM" mode).

The RAM mode serves several different purposes. First, the main processor can construct and store a database in CA-RAM using this mode. For each record in the database, the processor will need to compute the location in CA-RAM where the key should be stored, using the same hash function to be used by the CA-RAM index generator. If the "hashed" database already exists at other memory location or in hard disk, the construction of a CA-RAM database can be done via a series of memory copy operations or using an existing DMA mechanism. Second, the available memory capacity in CA-RAM can be treated as on-chip memory space for various general uses. It can be utilized as non-cacheable, paged memory space under OS management, or alternatively, as a scratch-pad memory without address translation. This way, applications which do not utilize the lookup capability of CA-RAM can still benefit from having fast on-chip memory space. Lastly, various hardware- and software-based memory tests will be performed on CA-RAM using this RAM mode.

There are three main operations defined for the CAM mode: (1) *search*, (2) *insert*, and (3) *delete*. Search looks up the database in CA-RAM with the given search key. When a search request is submitted through the request port of the CA-RAM memory subsystem, it is forwarded by the input controller to a relevant CA-RAM slice. The selected CA-RAM slice then looks up in its memory array and returns the search result. Multiple lookup actions can be simultaneously in progress in different CA-RAM slices, leading to high search bandwidth. Requests and results are both queued for achieving maximum bandwidth without interruptions. Insert and delete operations are used to construct and maintain a database. The auxiliary bits in each memory row are consulted and updated by these operations.

The CA-RAM slices in the subsystem can each serve a different database. The set of CA-RAM slices can be partitioned into multiple groups depending on the application requirements, each of which can form a separate database search engine. Certain CA-RAM slices can be used to implement an overflow area to store spilled records. If implemented, such a CA-RAM slice can be accessed together with other slices that keep regular records in order to achieve lower average latency, similar to the popular victim cache technique [11]. Furthermore, a database can be implemented with multiple CA-RAM slices, arranged vertically (*i.e.*, more rows), horizontally (*i.e.*, wider buckets), or in a mixed way. For example, five slices can be allocated together with four slices used to extend the number of rows and the remaining one set aside for storing spilled records. Another very attractive design option is *storing data along with its key* in CA-RAM, which can effectively eliminate the data access step typically following a lookup. Such an optimization is not practical in CAM due to its capacity and structural constraints. The necessary configuration information is stored in the configuration storage and is used to guide handling requests.

To integrate the CA-RAM memory subsystem efficiently, a processor may implement special instructions to submit requests and fetch results. Alternatively, request and result ports can be assigned a memory address, similar to memory-mapped I/O ports, so that ordinary load and store instructions can be used to access CA-RAM. For example, to submit a request, an application will issue a store instruction at the port address, passing the search key as the store data. Request and result ports can be extended by allocating multiple addresses to them, where a few address bits provide additional hints to the CA-RAM input controller. For example, each port address can be tied to a "virtual port" mapped to a specific database. In this case, the input controller can quickly determine which CA-RAM slice the request should be forwarded to, using the address bits. Also, if needed, request and result queues can be (physically) split into multiple queues for even higher bandwidth.

When writing programs that utilize CA-RAM, it is desirable to hide and encapsulate CA-RAM hardware details in a program construct similar to a C++/Java object which can be accessed only through its access functions. For ease of programming, CA-RAM-related operations can be best provided as a class library. Such operations include initializing an empty database, allocating/deallocating CA-RAM space (similar to `malloc()`/`free()`), defining slice membership and role (*e.g.*, use a slice as an overflow area), defining the hash function, declaring a record type and its format, enabling ternary searching, defining exception conditions, selecting operating modes, and setting power management policies. We note that the interface issues are important, but are beyond the scope of this paper. We leave them as a future work.

3.3 Preliminary implementation

To validate our CA-RAM ideas and identify further design and interface issues, a prototype CA-RAM slice was implemented. A standard-cell based design flow using the Verilog HDL and standard EDA tools, such as Synopsys Design Compiler, were adopted. As we used existing memory cores (*e.g.*, from a memory compiler), our design focus was on the processor interface, index generator, and especially, match processors.

Our design allows a configurable number of keys per bucket to increase the flexibility of use. This way, we can easily experiment with multiple applications requiring different key sizes without modifying the design. For simplicity, though, we limited the key size to be 1, 2, 3, 4, 6, 8, 12, and 16 bytes. Our design supports search with don't care bits in the search key or in the stored keys as discussed in Section 3.1. The implementation is similar to Figure 4(b). Also, data can be stored together with keys within CA-RAM. A lookup result is either a matched address or a matched data item. Control registers are provided in the form of memory-mapped peripheral registers to program various configuration options in our design.

The functionality of the match processor can be broken down into four steps: (1) expand search key; (2) calculate match vector; (3) decode match vector; and (4) extract result (*i.e.*, data). The first step is a consequence of the variable record sizes that the match processor can receive. Given that a bucket is C bits and depending on the stored key width, a C-bit expanded search key vector is generated having multiple search key appearances, each aligned to the stored key locations. This search key expansion step can be overlapped with memory access and thus its latency can be hidden.

In the second step, actual match actions are taken, comparing the search key with each stored key. A match vector is generated as a result, where each bit in the vector tells if the corresponding comparison resulted in a match. Since there can be multiple matches, we need to resolve this case using a priority encoder, which is done in the third step. Conditions where multiple matching records or no matching records are present are identified. As a result of the third step, a unique match location is known, if any. The matched data (not the key) is extracted in the last step. This step also takes into account the variable record size, which increases the design complexity. Note that in an application-specific CA-RAM design (*i.e.*, key length is fixed), much of this complexity will be removed.

Step	# cells	Area, μm^2	Delay, ns
Expand search key	3,804	66,228	(0.89)
Calculate match vector	5,252	10,591	0.95
Decode match vector	899	1,970	1.91
Extract result	6,037	21,775	1.99
Total	15,992	100,564	4.85

Table 1. Cell count, area, and delay for each stage of match processing.

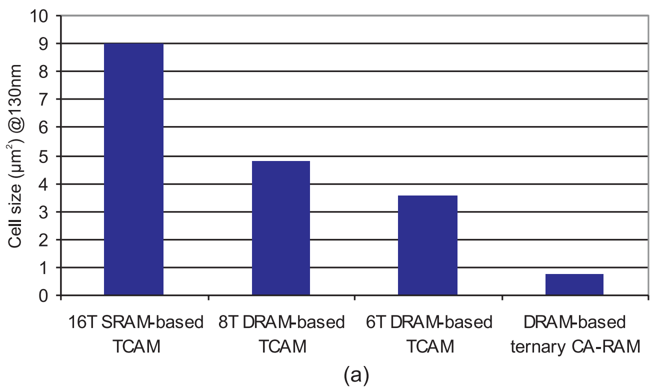
Table 1 reports our synthesis result of the match processor using a $0.16\mu m$ standard cell library. For the synthesis, we assume C is 1,600. We did not pipeline our preliminary design; still, we achieve a latency that will fit in a single cycle at over 200MHz. The match vector computation, including the expansion of the search key, requires the most cells. This is however a low delay stage as all the bit-by-bit operations are done in parallel. The decoding of the match vector and the multiplexing of the output results form the critical path as all of it's operations are serial in nature. Synopsys reported a worst-case dynamic power consumption of 60.8mW at V_{DD} =1.8V, switching activity $= 0.5$, and $T_{clk} =$ 6ns.

3.4 Performance, area, and power issues

Performance. The CA-RAM search latency, $T_{CA\text{-}RAM}$, is bounded by the access time of the memory, T_{mem}, and the time for key matching and result extraction, T_{match}. The CAM search latency, T_{CAM}, is determined by how quickly individual matchlines are precharged, pulled down, and sensed ($T_{matchline}$) and how quickly the priority encoder produces the match address [25]. When DRAM is used to implement CA-RAM and CAM, T_{mem} will be usually larger than T_{CAM} as it takes multiple cycles to access DRAM. If the latency for data access following a lookup is taken into account, however, $T_{CA\text{-}RAM}$ will be comparable to or even shorter than T_{CAM}, since the time to access data (T_{mem}) is fully exposed in CAM while it is effectively hidden in CA-RAM. Moreover, many recent CAM devices require multiple cycles to finish a lookup to save energy [7,24,25], making $T_{matchline}$ a multiple of the operating clock of CAM and rendering the actual CAM latency even longer.

The CA-RAM search bandwidth using a conservative, non-pipelined memory will be given as follows:

$$B_{CA\text{-}RAM} = \frac{N_{slice}}{n_{mem}} \times f_{clk}$$



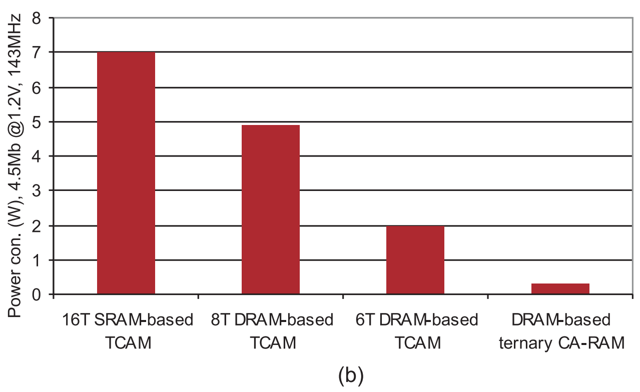


Figure 6. (a) Cell size of different schemes. (b) Power consumption of different schemes.

where N_{slice} is the number of CA-RAM slices independently accessible, n_{mem} is the minimum number of cycles needed between two back-to-back memory accesses, and f_{clk} is the clock frequency. Here, we assume that at least the match step is pipelined with memory access, dropping T_{match} in the calculation. The CAM search bandwidth is simply given as:

$$B_{CAM} = f_{CAM\_clk}$$

Usually, f_{clk} is much higher than f_{CAM_clk} using the same technology and design expertise (*e.g.*, over twice faster f_{clk} reported in [20, 24]), and increasing N_{slice} is straightforward in CA-RAM, which is in fact preferred for improved power control and aspect ratio fitting. CA-RAM is performance-competitive with CAM, in terms of both search latency and bandwidth.

Area. By decoupling memory array and match logic, CA-RAM achieves a smaller area than traditional CAM schemes. To validate this, we compare the cell size of competing schemes, including a 16T SRAM-based TCAM [23], an 8T dynamic TCAM [23], a 6T dynamic TCAM [24], and a DRAM-based ternary CA-RAM.

We use only actual product-grade implementation results published by a single research and development organization using the same advanced 130nm process technology to allow a fair comparison [23, 24]. To calculate the CA-RAM cell size, we use the result of an embedded DRAM implementation by the same research group [20] and consider the impact of adding match processors, derived from our prototype design. Since most of the existing works are on TCAM, we use two bits per cell in the case of CA-RAM, not to favor our own approach. We further assume that 16 CA-RAM slices are used (one slice for 64K cells), again not to underestimate the added overhead of the CA-RAM approach. We removed unnecessary features in the match logic prototype and scaled its size to the 130nm technology used. As a result, we determined a ~7% overhead due to the addition of match processors.

The comparison result presented in Figure 6(a) clearly demonstrates the advantage of CA-RAM. The CA-RAM cell size is over 12× smaller than a 16T SRAM-based TCAM cell, and 4.8× smaller than a state-of-the-art 6T dynamic TCAM cell.

Power. CA-RAM has a power figure that is dependent on three factors: the size and dimension of memory (w rows × n bits), the number of bits in each bucket that must be compared against the search key to determine if a match exists, and the operating frequency. CAM cells see much bigger loading capacitance since they are tapped by longer bitlines as well as additional matchlines, due to their larger sizes. More importantly, while CA-RAM typically accesses memory once per search followed by a match operation ($O(n)$), the CAM structure (in Figure 2) requires that all the searchlines and matchlines be activated at the same time ($O(w + n)$), and all the match transistors operate on each search ($O(w \cdot n)$). Their power consumption can be expressed as follows:

$$\begin{aligned} P_{CA\text{-}RAM\_search} &= P_{hash} + P_{mem}(w, n) + \\ &\quad P_{match}(n) + \\ &\quad P_{encoder}(w) \\ P_{CAM\_search} &= P_{searchline}(w, n) + \\ &\quad P_{matchline}(w, n) + \\ &\quad P_{encoder}(w). \end{aligned}$$

Using the same conditions as our area comparison, we compare the power consumption of different schemes in Figure 6(b). The result shows that CA-RAM is over 26 times more power-efficient than the 16T SRAM-based TCAM, and over 7 times improved over the 6T dynamic TCAM.

4 Application Study

The previous section demonstrated the potential of CA-RAM as a high-performance, low-cost, and low-power memory substrate for search-intensive applications. However, there may be limitations in using CA-RAM in certain cases. First of all, the performance of CA-RAM depends on the actual record distribution and how records are accessed. If many records have been placed in an overflow area due to collision, a lookup may not finish until many buckets are examined. Another limitation of CA-RAM is found when stored keys contain don't care bits in the bit positions used for hashing. To maintain the semantics of don't care bits, such keys must be duplicated in multiple buckets, leading to an increased capacity requirement. On the other hand, if the search key contains don't care bits which are taken by the hash function, multiple buckets must be accessed.

Collision is a unique problem in CA-RAM while not in a conventional CAM. There are several solutions to the problem. First, a more effective hash function reduces collisions. Second, allocating more capacity will also reduce collisions. Third, one can employ a CAM (alternatively a CA-RAM) to keep spilled records, similar to victim caching [11]. These solutions all pose interesting design trade-offs between performance, power, and area.

The goal of our application study in this section is twofold. First, the study aims to understand the impact of CA-RAM's possible limitations in real-world applications and how they affect the overall design trade-offs. Second, as a result, how CA-RAM compares with CAM (or TCAM) will be examined.

4.1 IP routing table lookup

Problem description. An IP router acts as a junction between two or more TCP/IP networks to transfer data packets among them. When an IP router receives a packet, it performs a series of operations such as checking the packet checksum and decreasing the time-to-live value of the packet. Usually, its performance bottleneck lies in the IP address lookup; the destination address of every incoming packet should be matched against a large forwarding table (routing table) to determine the packet's next hop on its way to the final destination. An entry in the forwarding table is called a *prefix*, a binary string of a certain length (also called *prefix length*), followed by a number of don't care bits. The adoption of *classless inter-domain routing* (CIDR) since 1993 resulted in the need for *longest prefix match* (LPM) [26].

There are two broad types of schemes to speed up IP address lookup: *software-based* and *hardware-based*. Software-based approaches usually require at least 4 to 6 memory accesses for forwarding one packet, and cannot keep up with the increasing throughput requirement imposed by modern networks, due to the relatively high latency and the limited bandwidth of current memory architectures [18]. Therefore, hardware-based solutions are preferred in order to scale performance to gigabit rates and beyond. TCAM is a current preferred solution because: (1) TCAM devices are available in commodities; (2) a single TCAM access is sufficient for an IP address lookup; (3) the don't care symbol in TCAM is suitable for representing don't care bits in a prefix; and (4) the priority encoder in TCAM can be used to perform LPM when prefixes in TCAM are sorted on prefix length [29].

Unfortunately, there are two serious impediments in using TCAM: its high cost and power consumption. According to the RIPE's routing information service project [27], the number of prefixes in the routing table of a core router has exceeded 200K, and is still growing. The size of a routing table will even quadruple as we adopt IPv6. Despite the current large TCAM development efforts [25], the sheer amount of required associative storage capacity remains a serious challenge. Moreover, the high power consumption of current TCAM devices is a pressing burden in large, reliable, and cost-effective system design [32]. New innovative memory design approach is needed to tackle these problems.

CA-RAM data mapping. For data mapping, we use the BGP (Border Gateway Protocol) routing tables of Internet core routers, obtained from the routing information service project [27]. We chose the routing table in the autonomous system AS1103 at the `rrc00.ripe.net` site for presentation. Results for other tables are similar. The prefix count in the table is 186,760.

First, we need to determine the hash function. Our hash function is based on the *bit selection scheme* by Zane *et al.* [32], which simply uses a selected set of bits (or hash bits) from IP addresses. According to Huston [10], also confirmed by our experiments, over 98% of the prefixes in the studied routing table are at least 16 bits long. Therefore, we restrict the choice of the hash bits to be from the first 16 bits of an IP address. When the number of rows of CA-RAM is 2^R, we apply the algorithm in [32] to find the best set of R bits which distributes the prefixes most evenly to buckets. After experiments, we determined that choosing the last R bits in the first 16 bits results in the best outcome. Note that if a prefix has n don't care bits in the hash bit positions, it must be duplicated and placed in 2^n buckets.

Next, we explore the design space by considering the key design parameters: C, R, and N, as described in Section 3.1. Because a prefix consists of 32 ternary bits, the length of the key (N) is 64. R can take any value from 1 to 16 and C is either $32\times N$ or $64\times N$. C is set to be comparable to the row width of recent DRAM bank designs [19,20]. We use a simple linear probing technique as described in

	R	C	# of Slices	Arrangement	Load Factor (α)	Overflowing Buckets (%)	Spilled Records (%)	AMAL$_\mathrm{u}$	AMAL$_\mathrm{s}$
A	11	32×64	6	horizontal	0.47	12.21%	15.82%	1.476	1.425
B	11	32×64	7	horizontal	0.40	5.42%	5.50%	1.147	1.125
C	11	32×64	8	horizontal	0.36	2.64%	1.35%	1.093	1.082
D	12	64×64	2	horizontal	0.36	6.67%	8.03%	1.159	1.126
E	12	64×64	3	horizontal	0.24	1.03%	0.72%	1.072	1.068
F	12	64×64	2	vertical	0.36	15.56%	29.63%	1.990	1.875

Table 2. Designs of CA-RAM for IP address lookup.

Section 2.1 to deal with bucket overflows. The main metric used is the *average number of memory accesses per lookup* (AMAL).

Evaluation. Table 2 shows 6 feasible CA-RAM designs. For example, in design A, there are 6 CA-RAM slices placed horizontally (to widen buckets). 12.21% of the total 2^{11} buckets overflow and the load factor (α) is 0.47. The AMAL metric for each design is also affected by the prefix access frequencies. Since we do not have IP lookup traces of core routers, we first assume a uniform access pattern for all prefixes, and compute "AMAL$_\mathrm{u}$". Then we assume a skewed access pattern [22], where some prefixes are accessed more frequently than others. In this case, we sort the prefixes on their prefix length (for LPM) and access frequency before placing in CA-RAM. The column labeled "AMAL$_\mathrm{s}$" shows the result. Although the skewed access pattern we use is an artifact, it demonstrates that access patterns can be taken into account in CA-RAM design to improve the lookup latency.

The result shows that with the same hash function (determined by R), investing more area (*i.e.*, reducing α) results in lower AMAL. It is also shown that for the same area (same α), the design with the hash function that distributes the data more evenly wins. This is evident from designs D and F. Design E, with the lowest load factor, achieves the best AMAL.

The number of duplicated prefixes due to don't care bits in the hash bit positions is modest – a 6.4% increase (12,035 additional entries) regardless of the design. This is because (1) the minimum length of the prefixes is 8 (*i.e.*, the first 8 bits of all prefixes are never don't care bits); and (2) the hash bits always cover the lower 8 bits in the first 16-bit positions ($R > 8$).

4.2 Trigram lookup in speech recognition

Problem description. A speech recognition system takes an acoustic language input and translates it into a corresponding textual representation. The three most important system design issues are: recognition accuracy, translation speed, and memory requirements.

In a modern speech recognition system like CMU-Sphinx [5], a sophisticated acoustic model based on the Hidden Markov Model (HMM) is used to simplify and improve the task of decoding. A language model is used to recognize and differentiate among the millions of human utterances that make up a language. Sphinx uses a conventional unigram, bigram, and trigram back-off model. The accuracy and speed of acoustic and language models rely heavily on searching a large database. For example, in a system with a ∼60,000-word vocabulary, the "N-gram memory" for language modeling is over 240Mbytes [17]. Unfortunately, the large amount of data to search against and the random access patterns in searching result in poor memory performance even with a large L2 cache in state-of-the-art processors. An efficient mechanism for searching will not only improve the speed of speech recognition, but also reduce the related power consumption, making it feasible to employ speech recognition in more performance- and cost-sensitive systems.

CA-RAM data mapping. We employ the trigram database used in the CMU-Sphinx III system [5], which comprises 13,459,881 entries in total. Because of its large size, we take a partitioned database approach and focus only on the entries with 13–16 characters. The resulting data set has 5,385,231 entries, equivalent to 40% of all the entries in the original database. Since each entry has up to 16 characters, the length of a key (N) is $16 \times 8 = 128$ bits. Ternary searching is not required in this application. We choose to store 96 keys in each bucket, and accordingly, C is $96 \times 128 = 12{,}288$ bits. The total data to store in CA-RAM amounts to 86Mbytes.

We use a different hash function strategy for this application. Instead of a simple bit selection method, we use the *DJB hash function*, which is an efficient string hash function [30]. The function looks like: $hash(i) = [hash(i-1) \ll 5] + hash(i-1) + str[i]$. This method has been also used in the software hashing technique in Sphinx. When a bucket overflows, we use a simple linear probing technique. The number of buckets of each slice is fixed to 2^{14}.

Evaluation. Four different designs are depicted in Table 3.

	R	C	# of Slices	Arrangement	Load Factor (α)	Overflowing Buckets (%)	Spilled Records (%)	AMAL
A	14	128×96	4	vertical	0.86	5.99%	0.34%	1.003
B	14	128×96	5	vertical	0.68	0.02%	0.00%	1.000
C	14	128×96	4	horizontal	0.86	0.15%	0.00%	1.000
D	14	128×96	5	horizontal	0.68	0.00%	0.00%	1.000

Table 3. Designs of CA-RAM for trigram lookup in speech recognition.

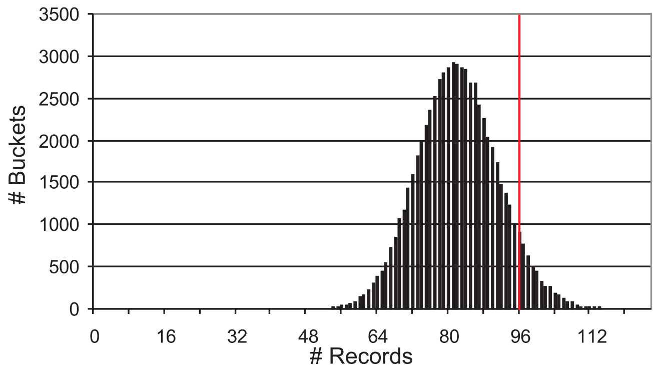


Figure 7. The distribution of buckets having a different number of records for design A in the trigram lookup application. The bucket size of 96 records will put a majority of buckets in the non-overflowing region.

In design A, for example, four CA-RAM slices are placed vertically to increase the number of buckets. At a relatively high load factor of 0.86, only 5.99% of all the buckets have overflows and consequently spilled records. Investing more area, as in design B, results in a lower load factor but better performance. Designs A and C or designs B and D show the trade-off between horizontal vs. vertical slice arrangement.

4.3 Discussions

From the two application studies, we observe that there is a trade-off between area (or α) and AMAL; the more area is spent (*i.e.*, the lower α is), the smaller AMAL gets. The ratio of changes in these two values ($\Delta\text{AMAL}/\Delta\alpha$) however depends on the application, the hash function, and the value of α. For instance, the benefit of spending more area is minimal in the trigram lookup application (*e.g.*, design A vs. B or design C vs. D). A smaller design having a slightly worse AMAL may turn out to be more viable; latency penalty can be easily compensated by employing a commensurately faster clock.

Compared with the IP address lookup application, the trigram lookup application achieves lower AMAL at much higher α, due to the hash function it uses. Figure 7 shows that the hash function of the trigram lookup application distributes the record very evenly, and as a result many buckets have a similar number of records, centered around 81. Choosing the bucket size of 96 resulted in only 0.34% of all the records spilled to other buckets in this design. It is very clear that the cost and performance of CA-RAM is contingent upon the effectiveness of the hash function.

On the other hand, it is less straightforward to optimize the IP address lookup application in terms of AMAL due to the difficulty of finding a more effective hash function. For this application, instead, we assessed the effect and cost of employing a small TCAM as the common area to store spilled entries. If this TCAM is accessed simultaneously with the main CA-RAM, AMAL becomes 1. Designs C and E require 1,829 and 1,163 entries be moved to the overflow area. In comparison, designs A and F have over 6,000 and 21,000 entries spilled, placing more pressure on the overflow area capacity. Implementing a separate overflow area should be carefully considered because it incurs power and area overheads.

Finally, we compare the area and power consumption of the TCAM (or CAM) design and the CA-RAM design for the two applications studied in this section. Based on the published implementation results, we estimate the cell area and the operating power consumption of different schemes, as we did in Section 3.4. Results are presented in Figure 8.

The TCAM estimate is derived from an optimistic scaling of the result of Noda *et al.* [24]. We assumed a 143MHz TCAM operation. The CA-RAM estimate is based on the DRAM implementation result of Morishita *et al.* [20] and an adaptation of the result in Section 3.3. We chose design D in Table 2 and further sliced it to create eight vertical banks, in order to obtain higher overall bandwidth, as discussed in Section 3.4. We take into account the load factor for area calculation, and assume a more aggressive 200MHz CA-RAM operation to make sure the CA-RAM design offers competitive search bandwidth as TCAM, assuming that the memory access latency is at least 6 cycles (DRAM).

For the result of the trigram lookup application, we referred to Yamagata *et al.* [31] to estimate the necessary cell area of a dynamic CAM. Again, we performed an optimistic area scaling. For CA-RAM, we chose design A in Table 3.

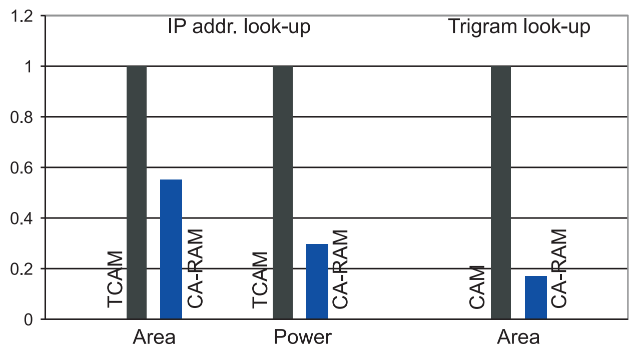


Figure 8. Area and power comparison of different schemes; TCAM [24] vs. CA-RAM for the IP address lookup application and CAM [31] vs. CA-RAM for the trigram lookup application. Results are scaled relative to TCAM or CAM.

We do not compare power consumption because the implementation in [31] does not have any advanced power reduction techniques, and accordingly a meaningful comparison would not be possible. We could not find a better reference because the most recent high-capacity development efforts were focused on TCAM devices.

The results shown in Figure 8 confirm our previous analysis in Section 3.4. Even with a low load factor, CA-RAM achieves a 45% area reduction compared with TCAM. Operating power saving is even greater, that is, 70% over TCAM. Compared with CAM, CA-RAM realizes a 5.9× area reduction. CA-RAM performs well for large real-world applications at much reduced power and area cost.

5 Related Works

5.1 Large capacity CAM organizations

Motomura *et al.* [21] presented a large-capacity CAM design for dictionary lookup applications in natural language processing. Their CAM array is divided into 16 *categories*, and matching actions are confined to a single category given a search key. The target category is determined by first looking up in a control-code CAM (C^2CAM), which stores indexes for the available categories. Their CAM structure achieves higher capacity by time-sharing a common match logic among the 16 categories. The concepts introduced by this work was later refined by Schultz and Gulak [28] as “pre-classified CAM”. Since both the works still bury match logic and match line inside the memory array, the maximum achievable area efficiency is severely limited.

Another straightforward technique for improving the density of CAM is to use dynamic memory cells rather than static memory cells [23, 24]. Instead of a conventional 16T SRAM-based TCAM cell implementation ($\sim 9\mu m^2$), Noda *et al.* [23] designed an 8T dynamic TCAM cell ($4.79\mu m^2$) using an advanced 130nm process technology. They further developed a 6T dynamic TCAM cell ($3.59\mu m^2$) [24]. It is noted, however, that an embedded DRAM cell implemented by the same authors using the same process technology ($0.35\mu m^2$) is an order of magnitude smaller than their smallest TCAM cell, and the resulting DRAM array can be operated at over twice the clock rate of the TCAM [20].

Lastly, more sophisticated encoding schemes can reduce the number of necessary entries in TCAM. Hanzawa *et al.* [7] proposed a new TCAM encoding scheme called “one-hot-spot block code”, which reduces TCAM entry count for an IP lookup application by 52%. Since each TCAM symbol uses 2 bits to represent one of the 3 values $\{0, 1, \mathrm{X}\}$, a two-symbol code (4 bits) has a redundancy of 7 out of 16. Their scheme is in essence a dense encoding scheme which fully utilizes this redundancy. We believe that this encoding scheme can be combined together with other high-capacity schemes, including CA-RAM.

5.2 Low power CAM techniques

The most common low-power CAM technique is to employ selectively accessible banks [13]. For example, Zane *et al.* [32] uses a two-phase lookup scheme where the first lookup is used to select a TCAM partition in the second, main table lookup phase. This bank selection strategy reduces overall power consumption in proportion to the number of partitions. For example, employing four partitions ideally reduces the power consumption by 75% compared to a regular CAM. In CA-RAM, even better, a memory access is made on a single row most of the time. The hash function used in CA-RAM replaces the more expensive first-phase lookup table in the banked CAM scheme.

Lin *et al.* [16] proposed a precomputation-based search scheme capable of decreasing the search power in conventional binary CAMs. This approach also uses a two-phase lookup scheme, where the first lookup is to match the precomputed signature, such as the number of 1’s in the search key. As a result of the initial lookup, the second search is performed on a limited number of entries in the main table. This scheme however is applicable to only binary CAMs.

There are other circuit-oriented low-power CAM techniques, such as low-swing match line and selective precharge schemes [25]. Since there are other comparable circuit-level low-power techniques for SRAM and DRAM designs, we do not further discuss these circuit techniques.

6 Conclusions

This paper presented CA-RAM, a high-performance memory substrate to accelerate important search operations present in applications, and analyzed its performance. The basic idea of CA-RAM is surprisingly simple; nonetheless, it is very powerful, and naturally opens up many opportunities for optimization. Our study demonstrates that CA-RAM achieves low-latency and high-bandwidth search performance, large capacity, and low power consumption. Using two real-world applications, we validate our approach and discuss important design trade-offs that are unique to CA-RAM. Experimental results showing the area and power savings of 50–80% corroborate the promise of the CA-RAM approach. The design flexibility stemming from completely decoupling memory array and match logic will lead to more exciting applications, as well as making CA-RAM suitable for integration in future general-purpose and embedded processors and systems [2].

In our current and future works, we will refine and optimize our prototype design. We will also uncover and study more applications that can benefit from using CA-RAM. We find cognitive applications especially interesting in this direction [6]. For example, a large-scale system implementing a cognitive model such as ACT-R [1], will benefit from employing CA-RAM, as it requires much search and data evaluation capabilities.

References

[1] ACT-R cognitive architecture. `http://act-r.psy.cmu.edu`.

[2] S. Borkar *et al.* "Platform 2015: Intel Processor and Platform Evolution for the Next Decade," *Platform 2015 White Paper*, Intel Corp., Mar. 2005.

[3] R. Brodersen. "Low Voltage Design for Portable Systems," *Proc. Int'l Solid State Circuits Conf.*, pp. 566–570, Feb. 2002.

[4] D. Burger and J. R. Goodman. "Billion-Transistor Architectures: There and Back Again." *IEEE Computer*, 37(3):22–28, Mar. 2004.

[5] The CMU Sphinx Speech Recognition Engine. `http://www.speech.cs.cmu.edu/sphinx/`.

[6] P. Dubey. "Recognition, Mining and Synthesis Moves Computers to the Era of Tera," *Platform 2015 White Paper*, Intel, Mar. 2005.

[7] S. Hanzawa *et al.* "A Large-Scale and Low-Power CAM Architecture Featuring a One-Hot-Spot Block Code for IP-Address Lookup in a Network Router," *IEEE J. Solid-State Circuits*, 40(4):853–861, Apr. 2005.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture A Quantitative Approach*, 3rd Ed., Morgan Kaufmann, 2003.

[9] D. Huggins-Daines *et al.* "PocketSphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, pp. I-185–I-188, May 2006.

[10] G. Huston. "Analyzing the Internet's BGP Routing Table," *The Internet Protocol J.*, Vol. 4, 2001.

[11] N. P. Jouppi. "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. Int'l Symp. Computer Architecture*, pp. 364–373, June 1990.

[12] M. Karlsson *et al.* "A Prefetching Technique for Irregular Accesses to Linked Data Structures," *Proc. Int'l Symp. High-Performance Computer Architecture*, pp. 206–217, Jan. 2000.

[13] G. Kasai *et al.* "200MHz/200MSPS 3.2W at 1.5V Vdd, 9.4Mbits Ternary CAM with New Charge Injection Match Detect Circuits and Bank Selection Scheme," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 387–390, Sept. 2003.

[14] D. E. Knuth. *The Art of Computer Programming*, Vol. 3 (Sorting and Searching), Addison-Wesley, 1973.

[15] T. Kohonen. *Content-Addressable Memories*, Springer-Verlag, 1980.

[16] C.-S. Lin, J.-C. Chang, and B.-D. Liu. "A Low-Power Precomputation-Based Fully Parallel Content-Addressable Memory," *IEEE J. Solid-State Circuits*, 38(4): 654–662, Apr. 2003.

[17] E. Lin. "A First Generation Hardware Reference Model for a Speech Recognition Engine," *MS Thesis*, School of Electrical and Computer Eng., Carnegie Mellon Univ., May 2003.

[18] H. Liu. "Reducing Routing Table Size Using Ternary-CAM," *Proc. Symp. High-Performance Interconnects*, Aug. 2001.

[19] Micron Technology. DRAM products. `http://www.micron.com/products/dram/`.

[20] F. Morishita *et al.* "A 312-MHz 16-Mb Random-Cycle Embedded DRAM Macro with a Power-Down Data Retention Mode for Mobile Applications," *IEEE J. Solid-State Circuits*, 40(1):204–212, Jan. 2005.

[21] M. Motomura *et al.* "A 1.2-Million Transistor, 33-MHz, 20-b Dictionary Search Processor (DISP) ULSI with a 160-kb CAM," *IEEE J. Solid-State Circuits*, 25(5):1158–1165, Oct. 1990.

[22] G. Narlikar and F. Zane. "Performance Modeling for Fast IP Lookups," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems*, pp. 1–12, June 2001.

[23] H. Noda *et al.* "A Cost-Efficient Dynamic Ternary CAM in 130nm CMOS Technology with Planar Complementary Capacitors and TSR Architecture," *Proc. Int'l Symp. VLSI Circuits*, June 2003.

[24] H. Noda *et al.* "A Cost-Efficient High-Performance Dynamic TCAM With Pipelined Hierarchical Searching and Shift Redundancy Architecture," *IEEE J. Solid-State Circuits*, 40(1):245–253, Jan. 2005.

[25] K. Pagiamtzis and A. Sheikholeslami. "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE J. Solid-State Circuits*, 41(3):712–727, Mar. 2006.

[26] Y. Rekhter and T. Li. "An Architecture for IP Address Allocation with CIDR," RFC 1518, 1993.

[27] Routing Inf. Service. `http://www.ripe.net/ris/`. 2006.

[28] K. J. Schultz and P. G. Gulak. "Fully Parallel Integrated CAM/RAM Using Preclassification to Enable Large Capacities," *IEEE J. Solid-State Circuits*, 31(5):689–699, May 1996.

[29] D. Shah and P. Gupta. "Fast Updating Algorithms for TCAMs," *IEEE Micro*, 21(1):36–47, Jan./Feb. 2001.

[30] S. Vakulenko. *Hash Function Efficiency*. `http://www.vak.ru/doku.php/proj/hash/efficiency-en`.

[31] T. Yamagata *et al.* "A 288-kb Fully Parallel Content Addressable Memory Using a Stacked-Capacitor Cell Structure," *IEEE J. Solid-State Circuits*, 27(12):1927–1933, Dec. 1992.

[32] F. Zane, G. Narlikar, and A. Basu. "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," *Proc. IEEE Conf. Computer Communications*, pp. 42–52, Apr. 2003.