Cascading Content-Addressable Memories

We survey the various methods of connecting multiple CAM devices to form a memory system of larger dimensions. The number of elements and the number of data digits per element can be increased relatively easily in contrast to increasing the number of label digits per element, which may be achieved using element, master-slave, or new trie cascades.

Tim Moors

Antonio Cantoni

University of Western Australia

A key factor in the economical use of the "ubiquitous 'random access' memory (RAM)" has been the designer's ability to "cascade" standard mass-produced components. Such cascades form memory systems of variable dimensions (for example, word widths) suitable for different applications. Similarly, combining content-addressable memory (CAM) devices to form a cascade of a larger dimension allows standard CAMs to support different applications without the need for custom devices. Although cascading is almost a trivial matter for RAMs, it is not for CAMs. Therefore, we focus on methods of cascading CAMs. Throughout this article readers may substitute CAMs and RAMs with their read-only counterparts (PLAs and ROMs).

Both CAMs and RAMs select elements of storage when every digit of a supplied "comparand" exactly matches the corresponding digit of the element's explicit or implicit label. We define "comparand" as the value the application supplies to the CAM for comparison. Associative memories, in contrast, select elements by inexact matches, for example, elements with the smallest Hamming distance from the comparand.

A common misconception is that RAMs and CAMs have complementary functionality. For example, when reading, a RAM uses a supplied address to read a value (using an 8-bit "house" number to read a 32-bit name of the "owner" of that number.) And a CAM, on the other hand, uses a supplied value to read an address. However, a RAM cannot distinguish names and house numbers; it just as readily uses a name to index a table of house numbers as it uses house numbers to index a table of names.

The key advantage of CAMs, for most applications, is that they can provide element storage for an arbitrary subset of N_e, possibly nonconsecutive, labels from the label space of 2^l labels. Here, N_e is the number of elements, and l is the number of bits in the comparand. This capability significantly lowers storage requirements when $N_e << 2^l$; that is, when names are used as labels ($l = 32$) rather than house numbers ($l = 8$). We look at other features that may distinguish a CAM from a RAM in the next section.

The added functionality of CAMs comes at a cost: Elements can no longer be identified by their spatial position as is done in a RAM. Instead, each element must include storage for both data and associated label(s), and the CAM must contain comparison logic for comparing these labels with the comparand. The combination of technical factors and lower market demand has limited the capacity of current commercial CAM devices[1-3] and ASIC blocks[4] to around 2^{16} digits, while 2^{22}-bit RAMs are commercially available.

Conventional software-searching algorithms, such as hashing,[5-7] are often inappropriate for hardware implementations because of the vari-

0272-1732/92/0600-0056$03.00 © 1992 IEEE

ance in their processing time. Although their mean delay may be small, the worst-case delay is often much larger, and it is the worst-case delay that is important for synchronous hardware implementations. (Pei and Zukowski[8] examined hardware implementations using tries,[5,7] a common software data structure. Later, we pursue the tries concept in the context of cascading CAMs.) Hence, despite their higher cost, CAMs have found widespread use[5] in such diverse applications as dataflow computers, address filtering for communications networks, and cached memory management.

However, the varied applications have differing requirements of the CAM in terms of its dimension, speed, and functionality. The overview of CAM operations in this article sets the context for a survey of known approaches for combining CAMs to form a cascade whose dimensions differ from that of the constituent devices. One of the methods examined for increasing the label size is a new trie cascade approach.

It is critical to maintain the perspective that RAMs are a specific, constrained type of CAM. RAMs can be used as the constituent "CAMs" in a cascade, provided they satisfy the requirements of a particular cascading method. Correspondingly, CAMs are similar to RAMs, and conventional RAM techniques, such as caching, can be applied to CAMs.

CAM operations

The four primitive operations available in a CAM are selection, multiple response resolution, reads, and writes.

Selection. In general, the comparand and elements of a CAM may use ternary logic, storing information as ternary digits (trits), which may assume the values 0, 1, or X (don't care). Here, the X value matches both 0 and 1. To interface ternary to binary logic, the CAM must use more than one bit to represent each trit. For example, a mask bit may specify whether a corresponding comparand bit should be interpreted as X. By definition, all element digits in the same position as a bit in the comparand form the element's label. The remaining element digits, corresponding to comparand X's, form the data for the element. We generally assume that any string of X's in the comparand is a known length. For example, to select three-character elements containing α anywhere (for example, $\alpha\delta\gamma$ and $\varphi\alpha\delta$), the CAM cannot use a single comparand *α*, where * denotes a variable number of X's. Instead, it must use the three comparands αXX, XαX, and XXα.

The application using the CAM can specify the element digits that are to be interpreted as the label by varying the position, and possibly the number, of X's in the comparand. It may be that some P "pure data" digits of the element cannot be used for labeling, but only for storing data. The comparand digits corresponding to these pure data digits implicitly carry an X value.

RAMs, in contrast, use binary logic for both data and labels. The positions of X values in the comparand are fixed and correspond to pure data digits of the element, with the address forming the remainder of the comparand. Since RAMs provide element storage for each possible comparand (address) ($N_l = N_e = 2^l$, where N_l and N_e are the number of distinct labels and elements in the CAM), a selection operation will select only one element.

We assume each CAM element has a fixed number E of digits, although labels and data may both be of variable size l and d, with $0 \le l \quad E - P$, and $d \equiv E - l$, as shown in Figure 1. Clearly, if either l or d of a CAM is larger than needed for an application, the application can pad its labels and data to match their size to that of the CAM.

Since all CAM comparand digits are treated equally (every digit must exactly match the corresponding label digit), digit ordering is irrelevant, provided the ordering used for the comparand is consistent with that used for the elements. This feature is the same for RAMs, although it contrasts with the more general associative memories. In associative memories, elements are selected if the value of their label is, in some way, associated to the value of a supplied label (for example, the memory selects elements with labels greater than the supplied label). Hence in associative memories, the meaning, and thus ordering, of digits may be important.

The CAM can store the vector indicating which elements have been selected by a previous operation and use this as a state variable for controlling subsequent operations on (un)selected elements. Thus, CAM selection operations may be cascaded. For example, elements may be selected when they match the current comparand and the previous operation had selected an adjacent element.

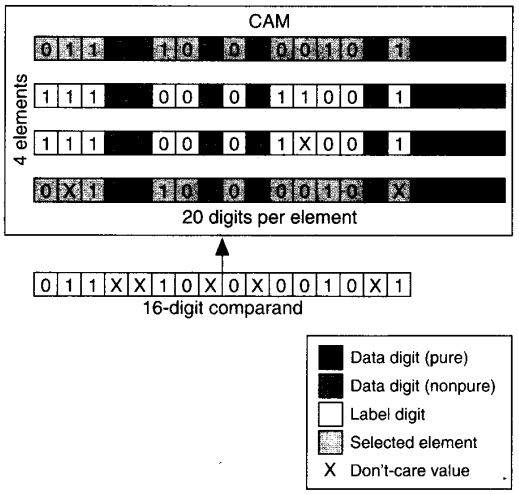

Figure 1. CAM parameters with values $N_e = 4$, $N_l = 2$, $P = 4$, $d = 9$, $l = 11$, and $E = 20$.

Multiple response resolution. After the selection operation(s), multiple elements may have been selected either because labels were not unique or because selections were Ored. RAMs differ in that only one element can be selected at any time. If the operation following selection is an element-serial one (reading data from a particular selected element or writing to *an* unused element), the CAM must resolve the multiple responses of the selection operation to establish the order of the processing. Since the CAM selects elements by exact matches to the comparand, no restriction exists on the order in which multiple response resolution must choose selected elements for serial processing.

Reads. The read operation returns an indicator of the number of selected elements (none, one, or multiple) and part or all of the information affiliated with these elements. Since CAMs generally provide a data bus capable of carrying information from only one element, the CAM must resolve the multiple responses prior to the element-serial reading of selected elements.

Writes. Write operations modify the label or data of all selected elements using supplied parameters that specify which digits to modify and how to modify them. The ability to selectively mask which data may be modified can be useful[9] (for example, when simulating multiple CAMs by time-multiplexing a single CAM). Selective masking avoids a read-modify-write sequence, which would also serialize the modification. The value to which a digit is modified may be an arbitrary function of the existing and supplied digits (Nand digits), although usually the supplied digit overwrites the existing digit.

Since multiple elements may be selected for a write operation, CAMs, in contrast to RAMs, may concurrently write common information to multiple elements in parallel. This powerful feature aids some applications, such as initializing memory with a test pattern.[10]

Operation application. As an example of the application of these primitive operations, consider Figure 2 showing a CAM used for cache memory management. Each element of the CAM contains storage for a cached block's main memory address, its attributes indicating how recently it has been accessed, and its cache address. To access the memory hierarchy, the processor supplies a main memory address as a comparand to the CAM, which selects any element describing a cached block with a matching main memory address.

A read operation then determines whether a matching label exists (a cache hit). If so, the cache controller reads the cache address for this block and uses it to address the cache memory. It overwrites the single-bit attributes field to record that the block has recently been accessed. If the selection failed to find a match for the main memory address, the cache controller must select a cache block for replacement using the attributes as a label. For example, it may select the least recently used block. (Designers place the block in the cache in anticipation that the same, or other, address(es) in the block will be accessed in the near future.)

The change in which element digits are to be interpreted as the label requires that the comparand mask be changed to indicate that the main memory block address bits are X. The cache controller then reads the cache address for this replacement block to control the transfer of the block from main memory to the cache and updates the main memory address data. If multiple blocks are equally suited to replacement (accessed at the same time), the CAM must resolve the multiple responses.

The fast component of this memory system

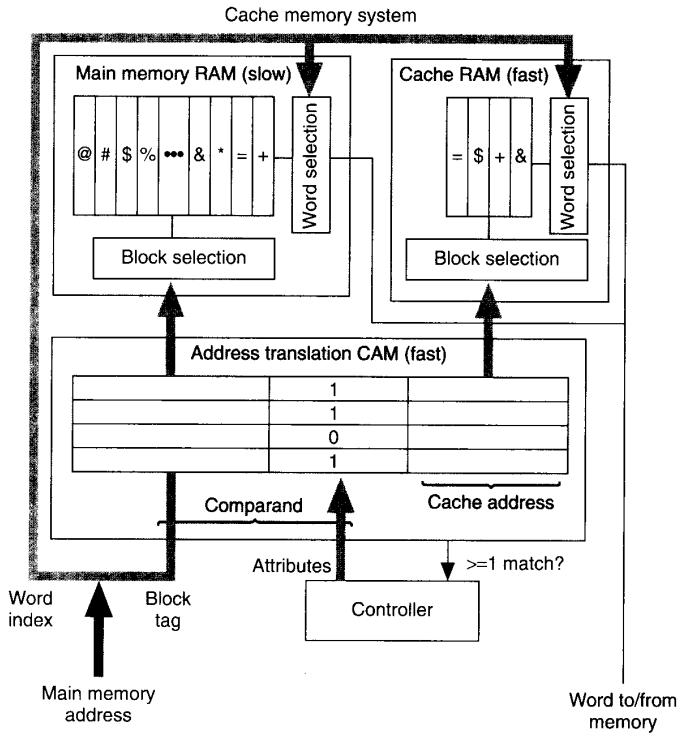

Figure 2. Application of a CAM to cached memory management. The cache address may be pure data and is often statically encoded. The symbols represent values of blocks of memory in the main memory and cache.

can be seen as being intermediate between a pure CAM and a pure RAM. Although storage is not provided for all word labels in the global label space, the fast component does provide storage for all labels in a number of local label spaces corresponding to blocks in the cache. Thus, it is evident that CAMs and RAMs form two extremes of a spectrum of memory systems that includes memories such as the fast component of this memory system.

In this cache application the CAM elements will continually be modified as blocks transfer to and from the cache. It is necessary to maximize the throughput of CAM operations to match the rate at which the processor cycles through new addresses and desirable to minimize the delay of CAM operations to minimize the branch penalty.

For other applications the demands of the CAM may be different. For example, consider a CCITT Broadband ISDN switch[11] in which a CAM is used to determine the output port for an incoming cell of information via the cell's label. The switch may have relatively static CAM elements corresponding to connections through the switch. It may require high throughput (over a million searches per second for 620-Mbps links), although delays significantly larger than the cycle time (for example, milliseconds) may be considered acceptable. The dimensions of the CAM would vary between implementations. That is, the number of elements corresponding to the number of concurrent connections supported, the number of data digits per element to the number of output ports, and the number of label digits per element would depend on which components of the cell label are used for switching (for example, virtual path or channel identifiers).

Thus, the varied applications of CAMs have differing requirements of the CAM in terms of its dimension, speed, and cost. Different cascading methods may be better suited to different applications.

CAM realization

Though modern VLSI (very large scale integration) processes permit the integration of millions of devices on one chip, interchip connectivity remains a bottleneck. Limited bonding pads and packaging costs restrict connectivity to, at most, hundreds of connections of limited throughput. Indeed, packaging dominates the cost of many chips. For a given storage capacity of N_l elements with different labels, a CAM providing $N_l < 2^l$ will require larger labels l than an equivalent RAM ($N_l = 2^l$). Additional pins, leading to increased costs, can be avoided by time-multiplexing the pins, at the expense of the time it takes to transfer information. Often, some externally supplied information is essentially invariant (the comparand masks used in the cache memory example). Therefore, CAM chips can overcome the bottleneck created from the multiplexed inputs by providing registers internal to the CAM chip (a cache) for storing this information, as is done in the GEC Plessey PNC1480.[3]

Cascading CAMs

We can increase the size of a CAM in any of its dimensions: the number of elements N_e, the number of data digits per element d, or the number of label digits per element l.

We can compare the methods for cascading CAMs in terms of their cost, functionality, and speed. We can assess these parameters for the constituent CAMs, the interconnections between constituent CAMs, and for the cascade itself.

We assess the cost of the CAMs in terms of the number of storage cells required and the cell's type (pure data or label). The cost of the connectivity is based on the number of inter-CAM connections required.

Functionality covers such issues as the ability to use ternary rather than binary logic and the potential for parallel writing and for reading an indication of the number of selected elements. Cascade functionality also encompasses maintenance of the cascade: how elements are added/removed, how labels and data can be modified, and whether such modification interferes with other concurrent operations.

Speed concerns the measurement of both the delay t_d for an operation to be performed and the cycle time t_c between operations. Although different operations may have different speeds, we use the selection operation as a benchmark. It is a reasonable choice since the speeds of other operations are mostly invariant between different methods of cascading to extend the CAM in a given dimension.

We can use concurrency to reduce t_c below t_d by either replicating the memory or by pipelining memory stages. Pipelining is often the preferred technique since the hardware overheads are often lower and the problem of consistency between memories is reduced. Realization factors—such as the necessity to refresh dynamic storage after a destructive readout and overheads between pipeline stages—ultimately impose a lower limit on t_c that is achievable through pipelining. In applications exhibiting strong spatial locality, we can divide elements with adjacent labels between replicated memories rather than duplicating them in each memory. By doing this, we achieve improved memory utilization and throughput. This method is well known in the context of RAMs as interleaving.

Adding elements

To increase the number of elements, CAMs may share a common comparand bus, as shown in Figure 3. For selection, CAMs do not need to interact since the selection of an element is independent of that of other elements. Hence, selection within a CAM remains independent of that of other CAMs.

Complexities arise in the subsequent processing of selected elements. To read an indicator of the number of selected elements in the cascade, the cascade must have added the number selected in each CAM. For example, a simple indicator of the range of the number of selected elements is whether

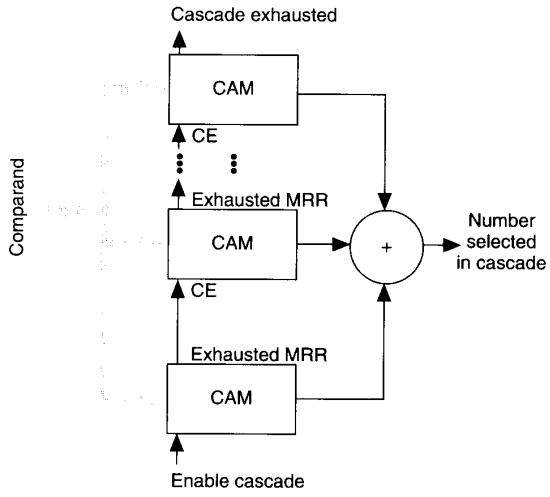

Figure 3. Increasing the number of elements. Element-serial operations require daisy-chaining connections.

at least one match for the comparand exists in the cascade. To determine if there is at least one match, the CAM must Or the match signals from the constituent CAMs.

Multiple response resolution, to be performed prior to serial operations, must account for selected elements in all CAMs of the cascade. Since multiple response resolution may be performed in arbitrary order for selected CAM elements, designers typically achieve the ordering by daisy-chaining CAMs.[12] As shown in Figure 3, a chip enable signal CE ripples through the cascade: A CAM enables the next CAM in the chain only if it has exhausted all of its selected elements. Carry-lookahead logic can increase the propagation speed of the enable signal.[13,14]

Although static element identifiers may be locally unique within each CAM, making them globally unique within the cascade requires that a unique identifier of the CAM in which the element is stored be added to the locally unique identifier.

Increasing data size

We may also want to cascade CAMs to increase the number of data digits per element because we have too few data digits per element, or because the data digits that are available are read-only. For example, many CAMs[1,3] statically assign each element a unique identifier (data) that must be used in a cascade to access a read/write memory.

We can use the same brute-force approach used in RAMs to increase the number of element digits available for data storage. Specifically, we can replicate labels for each element in multiple CAMs and distribute the data for each element among the CAMs, as shown in Figure 4a. Clearly, this approach provides a cascade element data capacity of pd, where p is the number of parallel CAMs. Such additional storage can only be used for pure data; it cannot later be used for labeling elements when the comparand changes. Furthermore, we cannot use this approach to provide writable storage when the CAMs have read-only data.

If CAM elements are assigned unique identifiers (statically or using $\lceil \log_2(N_e) \rceil$ bits of the available data storage), we can use these identifiers to index another CAM/RAM, which provides additional data storage as shown in Figure 4b. Although the functions of element identification and of data element selection using this identifier effectively cancel each other, they do introduce a serial bottleneck that prevents parallel writing to the supplementary data storage. Again, the additional storage is for pure data only.

A third approach to increasing the number of digits avail-

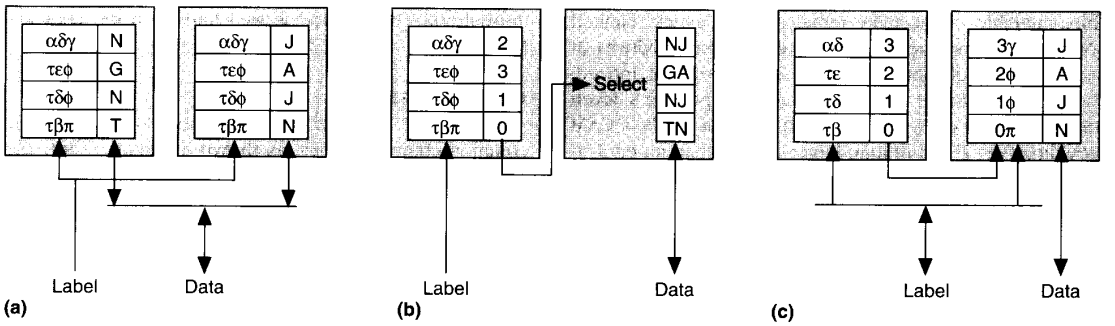

Figure 4. Cascading to increase data storage: replicating labels (a), identifiers indexing second RAM/CAM (b), and reducing label redundancy (c). Letters indicate the data associated with the labels.

able for data storage is to reduce the number of digits occupied by the label. Say $l > \log_b(N_l)$, where N_l N_e is the number of distinct labels in the CAM, and b is the base of the label digits (for example, $b = 2$ for binary). Then, a redundancy exists in the labels, which the additional CAM stage shown in Figure 4c can remove, and in doing so, increase the space available in the main CAM for data storage. This result comes at the expense of the additional CAM and increasing the selection delay to the sum of the delays in the constituent CAMs.

Increasing label size

In examining cascading to increase the label size, we assume that we seek cascade labels individually larger than the labels of constituent CAM chips ($\exists\ l_i : l_i > E - P$). We do not seek the combination of labels for an element to be larger than the space available in the CAM chips ($\Sigma\ l_i > E - P$ and $l_i < E - P$ $\forall\ l_i$). If the latter is true, we can use CAM chips in parallel without interaction, with as many labels per CAM chip as will fit. This structure is limited in that only the component of the element that was stored in the CAM matching the comparand will be selected. Other components, for example, other labels, will not be (directly) selected.

When labels are individually larger than that supported by the constituent CAMs, we must divide the comparand (and labels) into segments that are distributed between logically distinct CAMs. Some interaction must exist between these CAMs since it is not sufficient that each CAM finds a match for its segment of the comparand. The CAMs must match a common cascade element. For example, when a cascade of three CAMs stores the labels αδγ, τεφ, τδφ, and τβπ, with one character of each label per CAM, it should find no match for αδφ. However, matches would exist for each of these characters in the appropriate CAMs.

Before examining methods for cascading CAMs to raise the label size to that required by an application, we should examine how an application requiring smaller labels can use a CAM with larger labels. We need to examine this since it may be possible to manufacture CAMs with large labels and for applications to merge multiple smaller labels into the larger CAM label storage. As mentioned earlier, we can pad labels to match their size with that of a CAM; however we will end up poorly utilizing each CAM element.

By using ternary comparands, a CAM can simulate multiple CAMs of total label size $\Sigma\ l_i$ $E - P$, and total data size Σd_i $E - \Sigma\ l_i$. The CAM divides the E digits of the large elements between the smaller elements ($l_i + d_i$) and time-multiplexes the CAM. Large-element digits corresponding to other labels are masked out with an X in the comparand, as shown in Figure 5. From this, it is apparent that CAMs could be manufactured with extremely large elements, and applications requiring smaller elements could time-multiplex the large elements. However,

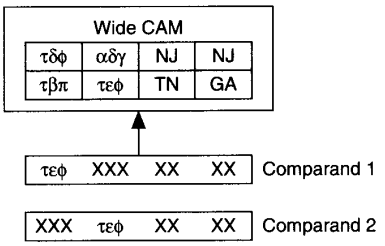

Figure 5. Using a wide CAM for thinner labels through time-multiplexing.

- The limited interchip connectivity would form a bottleneck for transferring large comparands to the CAM.
- As thinner elements are used, the CAM elements must be multiplexed more often for each operation, mitigating the speed advantage of CAMs resulting from operations being performed concurrently on all elements.
- To provide for modification of a subelement of a shared element, we require either maskable writing, or read-modify-write sequences that will degrade performance.
- Irrespective of the capacity of a single CAM chip, some applications may exist whose labels are larger than the total storage provided in a single CAM.

Thus, we seek methods for cascading thin CAMs to form a wide cascade. Since each method provides different cost, functionality, and speed, they suit different applications.

Element cascading. Since exact matches between the comparand and element are required in a CAM, the comparison between the comparand and element will produce a binary result. One approach for cascading CAMs is to divide the segments of the comparand into logically distinct CAMs, and to And the results from matching each segment of the comparand and label for each element. The cascade selects an element only if all segments match the corresponding comparand segments. This "element cascade" approach, shown in Figure 6a, requires at least one inter-CAM connection per element for Anding the match results. Thus, the inter-CAM connectivity will be high for even a moderate N_e (N_e = 256), and this approach is suitable not for cascading discrete CAM devices but rather for providing multidigit labels within a CAM device.

Rather than use physically distinct CAM chips, a single CAM chip can be time-multiplexed to emulate multiple CAM chips. This approach exploits the massive connectivity available on chip to provide the intraelement connectivity. In this approach,[2,13,15] we store segments of a cascade element in adjacent elements of the CAM. For example, segment s_i of Figure 6a is positioned below and adjacent to s_{i+1} as shown in Figure 6b. The CAM uses the first segment of the cascade comparand

to select cascade elements with matching first segments. Subsequent selection operations for other cascade segments select an element if it matches the comparand and its lower neighbor element matched in the previous selection operation.

To enable a CAM chip to be used with arbitrary length comparands, it should provide connectivity between all adjacent elements for the propagation of match results. We can do this by using a shift register, as shown by the screened connections of Figure 6b. The only required inter-CAM connectivity is two connections per CAM device to concatenate its shift register with those of adjacent devices. However, it is not sufficient for successive elements of the CAM to match the successive segments of the cascade comparand. These elements must also be in the same cascade element and not split across cascade elements. For example, in Figure 6b, no match should be found for φαδ, even though these characters are stored in consecutive cascade elements storing τεφ and αδγ.

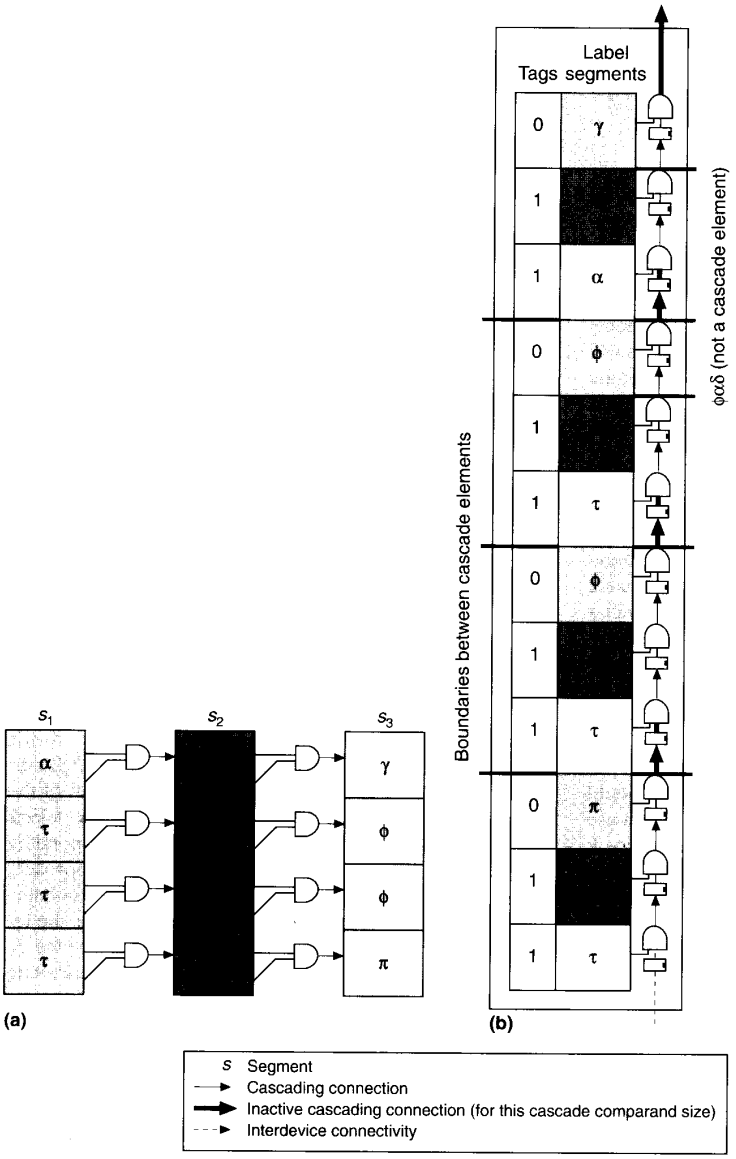

Figure 6. Element cascade: logical representation, for example, physically distinct CAMs (a) and within a single physical CAM, (b).

We can ensure the consecutive matches of cascade comparand segments occur within a common cascade element in a number of ways, including the following:

- *A maskable decoder/selector[13] can be used to limit the set of candidates for the first selection operation to the first element of each cascade element.* Often the decoder can only be masked to select CAM elements a power-of-2 elements apart, and thus cascade element lengths would be limited to CAM element lengths multiplied by a power of 2. This restriction on element lengths will affect the efficiency of memory utilization.
- *One or more of the segments of the comparand and elements can be anchored together,[16] ensuring that the cascade comparand segment only matches the corresponding segment of the cascade element.* We can implement this step in two ways. We can insert unique spacer code words that match only spacer words inserted in the comparand and mismatch the comparand segments. Or we can tag each segment of the element[15] and append the appropriate tag to each comparand segment.

Since only one anchor is required per cascade element, we can use a single-bit delimiter for tagging.[17] Not only does this minimize the per-

element overhead for storing tags, but it also obviates the need to supply tags as part of the comparand. The delimiter bit can enable the shift register (delimiter bit set to True for last segment of cascade element). Furthermore, by using a single-bit tag, we can use comparands that have a variable-length string of X's before the binary digits in the comparand. Thus, we detect any string of consecutive segments in the CAM matching the string of comparand segments. After selecting the first binary comparand segment, the CAM selects elements if their lower neighbor was selected in the previous operation (don't-care comparand). Provided cascade elements are of uniform size, repeating this don't-care selection $\lceil l_C / l \rceil - \lceil c_C / c \rceil$ times ensures that a match signal will propagate to the last segment only when the string of segments was wholly within the one cascade element. (Parameters with a subscripted C suffix refer to the cascade of CAMs rather than to a single CAM.)

In any of its forms, this element cascade approach permits virtually unlimited increases in the cascade label size in increments of the CAM label size. Yet, it requires only two inter-CAM connections per CAM device. As the number of required devices is proportional to $\lceil l_C/l \rceil N_{e,C}$, we can trade increases in the element size for the number of elements, without the need for additional devices. Furthermore, this approach does not require either elements or comparands to be binary rather than ternary. The cost comes in the form of either the anchoring or the maskable decoder, and through a reduction in speed. Both t_d and t_c increase in proportion to $\lceil l_C/l \rceil$. Unfortunately, some CAMs[1,3] do not provide the required shift register for cascading selection operations. Furthermore, for some applications, high throughput is of paramount importance. Hence, we investigate other cascading methods.

Master-slave cascading. To reduce the inter-CAM connectivity from that of the element cascade without a shift register, each CAM can pass a unique identifier of candidate element(s) (of at least $\log_2(N_e)$ bits to ensure uniqueness) to the other CAMs. When there are multiple candidates, the CAM must pass the identifiers serially, degrading performance.

One approach, shown in Figure 7a, assigns a unique identifier to each element in the cascade.[18]

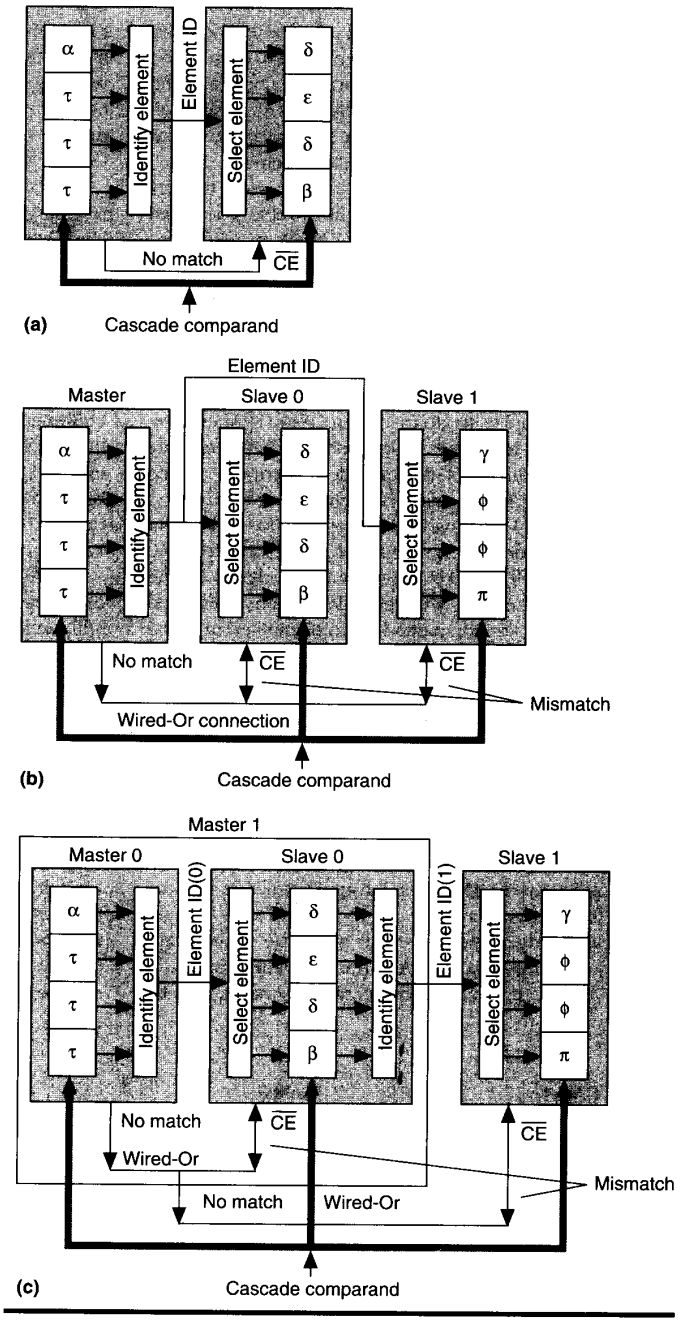

Figure 7. Master-slave cascading: one master and slave (a), master and multiple slaves (b), and multiple master-slave pairs (c).

A master CAM locates cascade elements that match the comparand in the master segment, and it passes the identities of these candidate elements to the slave CAM(s). Each of the slaves then attempts to match its comparand segment with the element identified as a candidate by the master. Only if all slaves match, as determined by Anding the match signals for each CAM chip, does the cascade select the element.

The master-slave approach suffers from the serial transfer of element identities from the master to slave(s). The worst case occurs when the master is full of labels matching the master's segment of the cascade comparand. This approach, then, takes a time proportional to N_e to select all matching elements or to ensure there are no matches. Even when N_e is only moderate, for example, $N_e = 256$, the worst-case delay will be extensive. Since we use CAMs rather than approaches such as hashing to avoid delay variance, the master-slave approach is not well suited to cascading CAMs. Furthermore, the slaves work in an element-serial fashion: They need only compare their segment of the cascade comparand to one label (as specified by the master's identifier) at any instant. Thus, any potential the slave CAMs may have for concurrent element comparisons (using distributed logic) is wasted. The slaves may as well be RAM lookup tables indexed by the identifier from the master, with the indexed entry compared to the segment of the comparand, as implemented in cache address comparator chips.[19]

We can extend the master-slave structure by using multiple slaves for the one master, as shown in Figure 7b. Alternatively, we can use a master-slave cascade as either the master or slave of a new cascade, as shown in Figure 7c. The latter approach would not be used for cascading since the selection delay would increase in proportion to the number of CAMs in the cascade (compared to the single-master approach in which the delay through the master and the slowest of the slaves). However, it forms a structure that can be used for trie cascading.

Trie cascading. Clearly, we can avoid the master-slave serialization by merging the CAM_1 (master) label storage for elements with common CAM_1 comparand segments. As multiple cascade elements can now share a common CAM_1 element, it is no longer possible to pass a cascade element identifier between CAMs and use the structure of Figure 7b. Rather, we must use the structure shown in Figure 7c, and the identity of the selected element in CAM_i is combined with segment $i + 1$ of the comparand to form the comparand for CAM_{i+1}, as shown in Figure 8.

We can understand this trie cascading approach[20] by representing the search space as a tree, as shown in Figure 9. Here, each path from the root node of the tree to a leaf node corresponds to a label stored in the tree, with data stored in the leaf nodes. Each CAM of the cascade corresponds to a level of the tree, and each occupied CAM element to a branch from that level. As the search progresses from level (CAM) i to the next, the search space reduces to the set of labels that have matched segments 1 to i of the comparand. This is equivalent to the searching algorithm used with trie data structures in software,[5,8,21] hence the name trie cascading. At the extreme, where each CAM comparand includes only a single bit of the cascade comparand, a binary tree forms, and the searching technique is equivalent to that used by Wolstenholme.[22]

Whenever more than one element is selected in a CAM, the identities must be passed serially between CAMs, which will result in the same performance degradation encountered in the master-slave approach. As a result, the labels and comparand can contain X digits in segment i only when all digits in subsequent segments are X's. This corresponds to terminating the search after reaching level i of the tree. Although the cascade elements matching the comparand will not have been selected (not all segments have been processed), this approach provides an indication of a match for the comparand within the cascade.

The identifiers from CAM_i divide the storage in CAM_{i+1} into logically distinct search spaces corresponding to branches from level i. By assigning each branch within the tree a unique identifier, we can store all branch-identifier/element-segment pairs within a single CAM. The advantage of distributing branch storage for different levels in different CAMs is that it enables

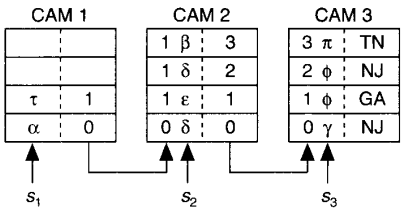

Figure 8. A serial trie cascade with s_x indicating segment x of the comparand.

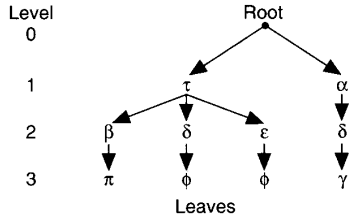

Figure 9. Tree representation of labels τβπ, τδφ, τεπ, and αδγ.

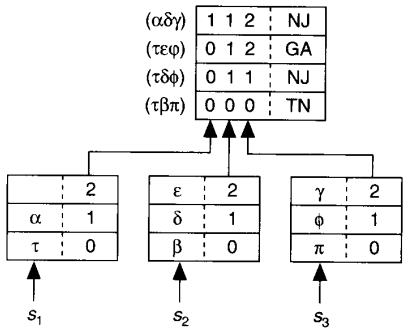

Figure 10. Parallel trie cascade.

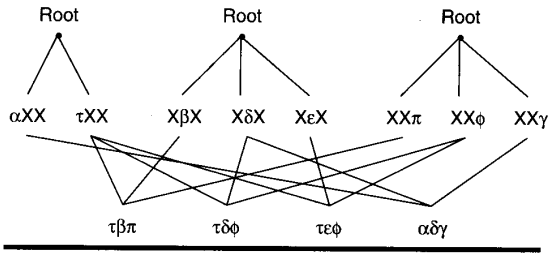

Figure 11. Forest representation of search space.

pipelining of CAM operations. A selection operation that has reached level i does not interfere with one at another level of the tree. Thus, the throughput of CAM operations can remain invariant as label sizes increase. With this serial trie cascade, the delay for CAM operations will increase in proportion to the number of levels in the searching process (label size).

We can reduce the delay for element selection by using multiple CAMs in parallel to concurrently reduce the search space, forming a parallel trie cascade as shown in Figure 10. As multiple CAMs at level i concurrently reduce the search space for level $i + 1$, the delay scales in proportion to the logarithm of the label size, although the throughput remains independent of the label size. This reduction in delay comes at the expense of additional hardware. In terms of the search space, this parallel trie cascade corresponds to a forest structure shown in Figure 11. Here, each root corresponds to a segment of the label, and leaf nodes correspond to target labels in the search space. Searching progresses concurrently from the roots of each of the trees until the search processes converge at a common leaf node.

The trie cascade approach requires inter-CAM connectivity proportional to the logarithm of the comparand size. The throughput is independent of the comparand size, and the delay can increase in proportion to the comparand size or to its logarithm, depending on the implementation. One weakness is that comparands and labels are effectively prevented from containing X's.

Maintenance of the trie cascade, that is, the addition and removal of elements, is also more complicated than for other cascading methods. When adding elements, we must search from the root level for the first level that does not have a branch matching the segment of the label. From this level onward, we must add a branch for each segment of the label. The only requirement of branch identifiers is that they be unique within a CAM. Hence, we can set them at CAM initialization and not have to modify them when elements are added or removed. To remove an element, we must search from the root level for the first level at which the branch used by the label is not shared with another label. This branch, and all branches for the label in subsequent levels, should then be removed. An implementation of element removal may use the CAMs to check for multiple matches of the identifier from CAM_i in CAM_{i-1} and prune the branches from CAM_i onward.

A NUMBER OF APPROACHES EXIST FOR CASCADING CAMs. We can daisy-chain CAMs to increase the number of elements and possibly use carry-lookahead logic to increase the speed at which the enable signal propagates through the cascade. To the data size, we can replicate the labels in distinct CAMs, use the data storage available in a primary CAM to index a secondary CAM/RAM, or reduce redundancy in labels.

To increase the label size, we can use an element cascade with or without a shift register, a master-slave cascade, or a trie cascade. The element cascade without a shift register requires one inter-CAM connection per element; with a shift register it requires two inter-CAM connections per device. With the shift register, the selection delay increases in proportion to the comparand size.

The master-slave cascade requires inter-CAM connectivity proportional to the logarithm of the number of elements, and the worst-case selection delay increases in proportion to the number of elements.

The trie-cascade method also requires inter-CAM connectivity proportional to the logarithm of the number of elements. It provides a throughput independent of the comparand size, and a delay proportional to the comparand size or to its logarithm, depending on the implementation.

Acknowledgments

Grants from the Australian Telecommunications and Electronics Research Board and the Department of Industry, Technology and Commerce, Industry Research and Development Board supported this work.

References

1. "Am99C10 256x48 Content Addressable Memory," *AMD Memory Products*, Advanced Micro Devices, Sunnyvale, Calif., 1989, pp. 4.119-4.144.
2. "CRC32256 and Coherent Processor Product Announcements," Coherent Research Inc., East Syracuse, N.Y., 1991.
3. "PNC1480 64-Kbit Content Addressable Memory: Advance Information," GEC Plessey Semiconductors, Swindon, Wiltshire, UK, June 1991.
4. "LEA100K Embedded Array Series Product Overview," LSI Logic, Milpitas, Calif., May 1991.
5. T. Kohonen, *Content-Addressable Memories*, 2nd ed., Springer-Verlag, Berlin, 1987.
6. D. Knuth, *The Art of Computer Programming, Sorting and Searching*, Vol. 3, Addison-Wesley, Reading, Mass., 1968.
7. E. Fox et al., "Practical Minimal Perfect Hash Functions for Large Databases," *Comm. ACM*, Vol. 35, No. 1, Jan. 1992, pp. 105-121.
8. T. Pei and C. Zukowski, "VLSI Implementation of Routing Tables: Tries and CAMs," *Proc. Infocom*, IEEE Computer Society Press, Los Alamitos, Calif., 1991, pp. 515-524.
9. F. Hermann et al., "A Dynamic Three-State Memory Cell for High-Density Associative Processors," *IEEE J. Solid-State Circuits*, Vol. 26, No. 4, Apr. 1991, pp. 537-541.
10. G. Giles and C. Hunter, "A Methodology for Testing Content Addressable Memories," *Proc. Int'l Test Conf.*, CS Press, 1985, pp. 471-474.
11. "Switching for Broadband Telecommunications," *IEEE J. on Sel. Areas Comm.*, Vol. 5, No. 8, Oct. 1987, pp. 1217-1376.
12. T. Ogura et al., "A 4-Kbit Associative Memory LSI," *IEEE J. Solid-State Circuits*, Vol. 20, No. 6, Dec. 1985, pp. 1277-1281.
13. J. Wade and C. Sodini, "A Ternary Content Addressable Search Engine," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 1003-1013.
14. A. McAuley and C. Cotton, "A Self-Testing Reconfigurable CAM," *IEEE J. Solid-State Circuits*, Vol. 26, No. 3, Mar. 1991, pp. 257-261.
15. T. Ogura et al., "A 20-Kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 1014-1020.
16. K. Takahashi et al., "A New String Search Hardware Architecture for VLSI," *Proc. 13th Ann. Int'l Symp. Computer Architecture*, CS Press, June 1986, pp. 20-27.
17. M. Hirata et al., "A Versatile Data String-Search VLSI," *IEEE J. Solid-State Circuits*, Vol. 23, No. 2, Apr. 1988, pp. 329-335.
18. T. Nikaido et al., "A 1-Kbit Associative Memory LSI," *Jap. J. Appl. Phys.*, Vol. 22, Supp. 22-1, 1983, pp. 51-54.
19. *Cache Memory Management Data Book*, Texas Instruments, Dallas, Tex., 1990.
20. T. Moors et al., "Cascading CAMs for Increased Label Size," Univ. Western Australia Networking Research Laboratory report NRL-TR-3, 1991.
21. E. Fredkin, "Trie Memory," *Comm. ACM*, Vol. 3, No. 9, Sep. 1960, pp. 490-499.
22. P. Wolstenholme, "Filtering of Network Addresses in Real Time by Sequential Decoding," *IEE Proc.*, Part E, Vol. 135, No. 1, Institute of Electrical Engineers, UK, Jan. 1988, pp. 55-59.

Tim Moors is a PhD student in electronics engineering at the University of Western Australia, where he is investigating the interconnection of metropolitan area networks. His technical interests include computer architecture and networking. Moors received a BE degree in electronics engineering from UWA and is a student member of the Institute of Electrical and Electronics Engineers Computer and Communications societies.

Antonio Cantoni is an associate professor in the Department of Electrical and Electronics Engineering at UWA. Earlier, he was director of the Digital and Computer Systems Design Section of QPSX Communications Ltd. in Perth for the development of the DQDB metropolitan area network. He has also been a lecturer in computer science at Australian National University in Canberra and chair of computer engineering at the University of Newcastle in Shortland, New South Wales, Australia.

Cantoni received his BE and PhD degrees in electronics engineering from the University of Western Australia, Nedlands. He is a senior member of the IEEE and a member of the Association of Computing Machinery.

Direct questions concerning this article to Tim Moors, Networking Research Laboratory, Dept. of Electrical and Electronics Engineering, University of Western Australia, Nedlands, WA 6009, Australia; or e-mail at tim@swanee.ee.uwa.oz.au.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 168 Medium 169 High 170