Associative Processors and Memories: A Survey

Because of declining hardware prices, associative (content-addressable) architectures are again gaining importance, especially in artificial intelligence and database applications. This survey introduces the classical content-addressable memory and explains its realization at the transistor level. Then it describes some unorthodox CAM approaches, discusses associative processor systems, and classifies current approaches.

Karl E. Grosspietsch

German National Research Center for Computer Science

Computer architects have discussed associative systems since the early days of computer design. Associative system design features provide alternatives or additions to the classical von Neumann machine architecture.[1,2] In this brief survey of recent developments in the field, let's first consider the functional structure of a classical content-addressable memory (CAM) and its realization at the transistor level. Then we can look at some unorthodox approaches for CAMs, discuss associative processor systems, and classify the existing approaches in this field.

In this article, "associative" is synonymous with "content addressable." In some contexts, the term is linked with memory or processor structures based on neural net approaches, which we won't consider here.

Associative memories

In contrast with access via memory addresses in a conventional RAM, in a CAM the system accesses the content of a word cell by a comparison with a given search argument.[1] Functionally, the word cells of a typical CAM consist of two parts:

- a *search key field* whose contents can be compared with the search argument by some associative logic, and
- the *data field* built of conventional RAM bit cells.

If the search key field of a word cell i ($i = 0, ..., w - 1$) matches the search argument, a hit line emanating from the search key field activates the bit cells of the data field of word cell i (see Figure 1). Let k denote the length of the search key field, and l denote that of the data field; the entire word length is $n = k + l$. For $l = 0$ we have the special case of a CAM in which each bit slice has associative search logic.

When the system compares a search pattern with the search key fields, it loads the search pattern into the search argument register (SAR). It performs the subsequent comparison for each w word cell of the CAM concurrently. Inside each word cell, the comparison is presumed to be carried out bitwise in parallel. If the key field of a word cell i equals the given search argument, a hit signal results and is stored in cell i of the hit register (see Figure 1).

0272-1732/92/0600-0012$03.00 © 1992 IEEE

A second input register, the mask register, may mask the comparison with the contents of the SAR. If a bit j (j = 0, ..., k – 1) of this mask register is set to 1, in all bit cells of the corresponding bit slice j the comparison is not carried out. Instead, these cells always generate a local hit signal. In this way, the mask register can reduce the number of effective bits in the search key field.

This memory structure allows multiple hits within the CAM. In the case of a write operation, the same bit pattern can be written from the memory data register (MDR) into all word cells found via their search key fields. In the case of an associative read operation with more than one hit, the system must serialize the output of the words found. A priority logic does this by selecting one word cell from the set of word cells found—for example, the one with the highest internal address i, which is given by the bit position of the hit signal in the hit register. If word cell i has been selected, the corresponding output line of the priority logic is set to 1; all other output lines have the value 0.

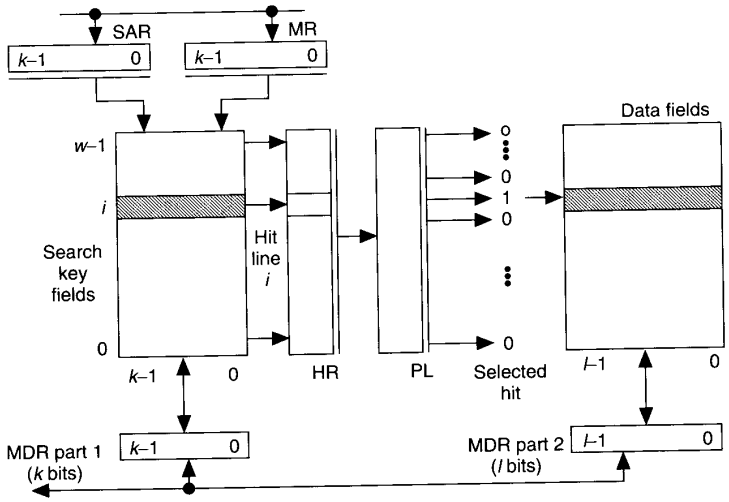

Figure 1. Architecture of a classical CAM. (HR indicates hit register; MDR, memory data register; MR, mask register; PL, priority logic; SAR, search argument register; and shaded areas, the two parts of word cell i).

Some approaches for CAMs also provide an individual "masked" state for every CAM bit cell. Therefore, such a memory (also called functional memory) requires a bit cell with at least three different storage states: 0, 1, and don't care. For a more detailed discussion of this aspect, see the contributions of Herrmann and Sodini and Moors and Cantoni in this issue, pp. 31–41 and 56–67.

Realization of CAM bit cells

One way to build a CAM bit cell is to extend the classical static RAM flip-flop cell.[1] Figure 2 shows such an approach. In addition to the six transistors of the flip-flop, three other transistors implement the match logic. Writing a 1 (0) is performed as in a RAM. The pass transistors T1 and T2 are opened via the word line. The bit line Bit is set to 1 (0), whereas the complementary line Bit′ is driven to the inverted signal of Bit. So transistor T6 (T5) is conducting, and T5 (T6) is nonconducting, thus representing the storage of the written value.

The comparison logic is simply realized by two pass transistors T7 and T8. If the system compares the cell's content with a given search bit s, the match line is first precharged to high. Then line Bit is set to the inverted search value s' (and, correspondingly, Bit′ to s). If a 1 was stored in the flip-flop so node 1 is high, transistor T7 propagates the value 0 from line Bit to the gate of T9. So this transistor remains nonconducting, and the match line is not discharged to ground.

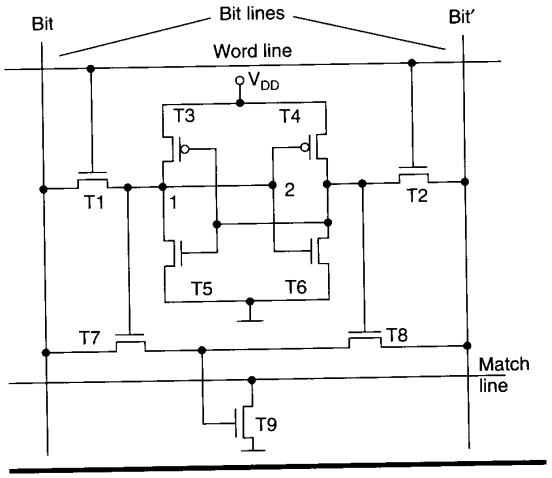

Figure 2. Structure of a nine-transistor CAM bit cell.

Any mismatch (in our example, a coincidence of a signal 1 of line Bit with the high value of node 1) opens T9, discharging the match line. Driving both bit lines to 0 masks the bit cell.

A similar approach simply uses four additional pass transistors to combine nodes 1 and 2 by negated XORs (see Fig-

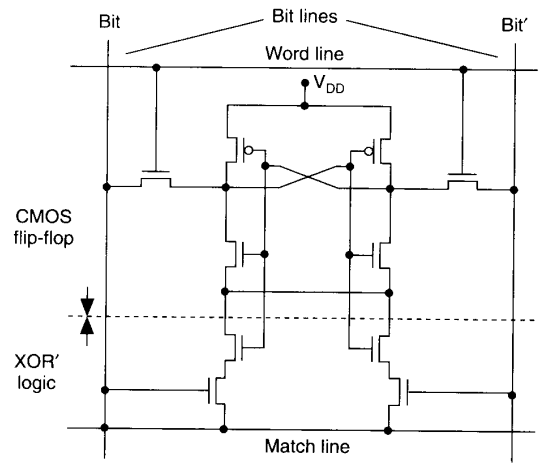

Figure 3. Ten-transistor CAM bit cell.[3]

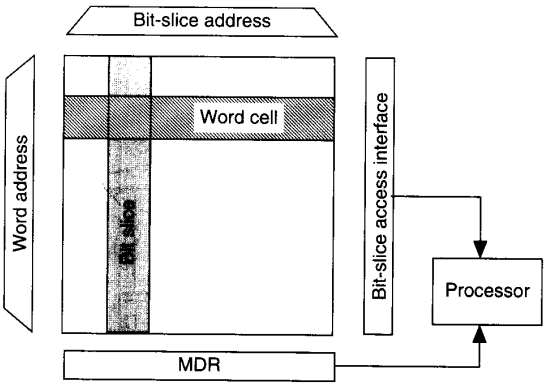

Figure 4. Orthogonal memory.

ure 3). The output of these XOR's is wired to the match line. In this 10-transistor cell, driving the line Bit to the noninverted search value s (and Bit' to $\bar{s}$) carries out a match against a stored bit. A mismatch again causes the discharge of the precharged match line; applying 0s to both bit lines masks the bit cell. On the basis of this circuit architecture, Kadota et al.[3] developed a CAM chip comprising 256 words of 32 bits; that is, the entire chip had a storage capacity of 8 Kbits. To avoid resistance-capacitance delays and minimize access time, they used low-resistance double-layer metallization techniques to lay out the bit and match lines.

In a comparable approach, Ogura, Yamada, and Nikaido[4] developed a CAM chip with a 4-Kbit capacity. It consisted of 128 words of 32 bits. Apart from retrieval functions, this chip could also write the same data into multiple words in parallel. Two years later, a more advanced solution for a 20-Kbit CAM evolved from this implementation.[5] The chip had a bit cell array containing 512 words of 40 bits and functional blocks for bit, word, and address operations.

These chips were more or less research prototypes. In 1987, Advanced Micro Devices introduced the first commercial version of a VLSI CAM chip, the Am95C85. It stored 1 Kbit of data. The memory could respond to an 8-bit key in about 10 μs. In 1989, the company followed up with the 12-Kbit Am99C10 chip. This chip uses a 48-bit-wide key that can be compared in parallel with 256 words. An intended application area is address management in local area networks.

Some approaches realize CAMs by extending dynamic RAM cells.[6] DRAM techniques achieve considerably larger bit capacities per chip because they have fewer transistors per bit cell (one or three transistors per cell). However, with these techniques either leakage currents or destructive read operations necessitate refresh operations. Because the system reads several bit cells simultaneously in a CAM, a destructive read operation as used in conventional dynamic RAMs is not possible. Also, data must be stored so a mismatch cannot destroy them. This can be accomplished by storing the data on the gates of two transistors. (For a detailed discussion of ways to implement such dynamic CAM cells, see Herrmann and Sodini's contribution in this issue.)

Unorthodox CAM approaches

An approach that differs significantly from the classical CAM structure is the orthogonal memory recently proposed by Kokubu, Kuroda, and Furuya.[7] (Batcher previously realized a similar structure in the Staran machine.[8]) The orthogonal memory does not provide comparison logic in the bit cells. Thus, the bit cell is quite similar to that of a RAM. However, two additional pass transistors permit the cell contents to be read out to the word line (see Figure 4). Apart from the normal random-access mode, the system can access an entire bit slice concurrently in a second memory mode and compare the slice outside the cell array, bitwise in parallel with the search pattern. Thus, it can easily perform a word-parallel, bit-sequential associative search.

Another very unorthodox approach recently proposed by Tavangarian[9] and Waldschmidt[10] is the associative random-access memory (ARAM). It is also called "location-addressable" associative memory and is realized mainly by a very simple change of the usual RAM decoder. In addition to the Nand gates $G_0 \ldots G_{w-1}$, we have the additional Nand gates $A_0 \ldots A_{2^m-1}$ ($m = ld\ w$ being the length of the memory address), which combine the input address with a mask pattern from an additional decoder mask (see Figure 5). If the mask is zero, the extended decoder functions like a normal 1-out-of-w decoder.

Otherwise, it carries out a multiaccess to all cells with addresses matching the unmasked bits of the input address.

The corresponding memory array is—in the simplest approach—formed by exactly one bit slice. Associative processing is then based on the following rule: The system stores an m-bit pattern $b_0 \ldots b_{m-1}$ in memory by setting a 1 as a flag to the cell with the address $b_0 \ldots b_{m-1}$. Analogously, the system searches for the pattern by checking whether the corresponding bit cell stores a 1 or a 0. The advantage of this architecture is the easy cascadability of the memory (for the bit cells themselves, standard RAM technology can be exploited). Moreover, the data stored in the ARAM are structured in ascending order. The data organization supports search operations such as queries about whether bit patterns larger or smaller than a given bound or patterns within a given interval of binary numbers are stored in the ARAM. The trade-off is that increasing the bit length of the patterns to be stored in the ARAM by i bits requires an increase in address space by a factor 2^i. So ARAM application is confined to data with relatively short bit length.

Associative processor systems

The notion of content-controlled access to data can be generalized to the processing of data: Not only are data retrieved according to their properties, but their updating or deleting is organized in that way. Such an associative processor system can, for example, be realized by placing a CAM as a data memory inside an environment of some processing logic, either a conventional host monoprocessor or arrays of processing elements. Several approaches for associative processor systems are based on assigning some processing logic to each CAM word cell, in addition to the priority logic that is also necessary. Some approaches also consider the integration of processing logic directly within the CAM bit cells.

Researchers investigated such processor systems already in the early days of computer science in the 1950s and 1960s. Probably the most well-known early approach for building an associative processor system was Batcher's Staran computer.[8] This system used a memory with access capabilities similar to the orthogonal memory: Apart from normal word access, it also enabled access to entire bit slices. Moreover, it provided access features between these two extremes: In a mixed mode, a regular diagonal pattern of bit groups was accessible throughout the memory.

Foster's book[11] and Thurber and Wald's and Yau and Fung's survey papers[12,13] provide good overviews of early approaches. Usually, these approaches could not transfer to the market because of high hardware costs.

AI applications

Several approaches for developing application-specific CAM architectures to support artificial intelligence features[14] have recently been reported. Kogge et al.[15] at Syracuse University developed a CAM tailored to specific AI applications. These applications use some form of If-Then rule programming. In languages that mainly use production rules, the system compares data against If parts until the arguments are satisfied. The Then parts usually indicate how the data are to be changed. Other more deductive languages such as Prolog perform matches between a goal—the truth value of which is usually not known—and the Then part of a rule.

Kogge et al. developed a VLSI coprocessor together with a CAM to work as a hardware accelerator for such tasks. The CAM is based on a special 10-transistor bit cell. For the applications considered, the flexible cascading of CAM chips is essential because of additional words and increased word length. In the approach, the basic word length is 32 bits. An

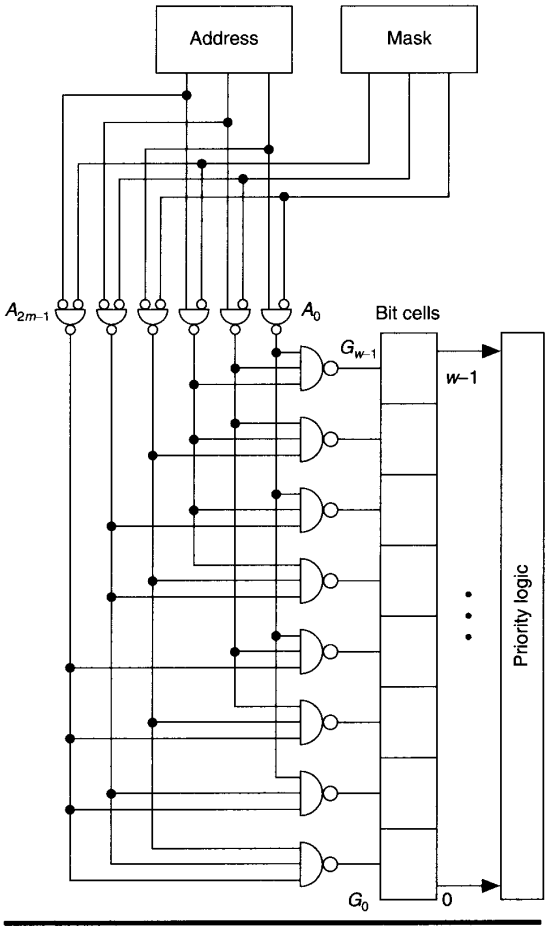

Figure 5. Architecture of the ARAM.[9]

additional counter field of 5 bits in each word implements extensions of this data word length. This field encodes 32 different positions in a larger extended data set. So the efficient word length can be varied between 32 bits, and 32×32 equals 1,024 bits.

The authors demonstrated several schemes using this CAM-oriented architecture for Prolog programming tasks such as variable binding, heap processing, and clause filtering. The system's host processor is a normal RISC-like processor. Several benchmark examples demonstrated the merits of the architecture by achieving very large processing speedups, while others showed smaller speedups.

Because CAMs have significant hardware costs, new approaches perform search operations directly at the interface between main memory and background mass memory media.

Ng, Clover, and Chung's approach is another example of such work.[16] The authors focus on hardware support for the binding of variables in Prolog program executions. In their approach, a CAM stores fact clauses as well as rule clauses. To maximize the speed of variable bindings, Ng, Clover, and Chung elaborated strategies to replace in parallel the variable contents of all matching expressions with their bound value. The system performs this directly when it detects a match—not only when it transfers variables to a special "bindings stack," as in former architectures for functional languages. The authors investigated different special CAM bit cells and developed a special 10-transistor CAM cell.

Naganuma et al. reported another associative approach for supporting Prolog execution.[17] Their processor consists of two subprocessors, one carrying out clause invocation, the other performing argument unification. Both subprocessors work in parallel. In this architectural scheme, CAM components mainly support a binding stack and a backtracking stack. These components are based on a CAM chip design of Ogura, Yamada, and Nikaido.[4]

Chae et al.[18] developed a CAM chip for a pattern inspection system. The chip comprises 256 twenty-five-bit words, together with shift registers and an address decoder/encoder. This storage memorizes 128 pattern words of 25 bits, each word with a corresponding 25-bit mask word. The size of the pattern words coincides with a 5×5 window in a binary pixel array; the corresponding mask word characterizes don't-care positions within this window.

Robinson's pattern-addressable memory (PAM) also uses single-instruction, multiple-data processor parallelism together with a memory that is content-addressable via hardwired pattern matching. The basic bit cell is a simple three-transistor DRAM cell, together with a comparator element formed by six transistors. Robinson presented a prototype PAM chip containing 1,152 twenty-bit words in 1989.[19] For a detailed description of the project's current status, see Robinson's contribution in this issue on pp. 20–30.

Another interesting approach for building processing logic around a CAM is the GLiTCH system,[20] used mainly to support vision processing. This chip contains 64 processing elements, each with 68 bits of local CAM. (Storer et al. present a detailed description of this architecture in another article of this issue, pp. 42–55.)

Mass memories and databases

CAM approaches have significant hardware costs. With the storage of large amounts of data, other essential limitations are

1) the memory must be loaded before it can be used for search operations, and
2) the fixed size of the array also necessitates a fixed comparison length and special efforts to change the format once it is selected.

Therefore, researchers have developed approaches to perform search operations directly at the interface between main memory and the slower background mass memory media (disk or tape). Such methods use the logic-per-track concept, which requires that comparison circuits be added to the read/write heads of disks or comparable media. The system checks for equivalence with search patterns on the fly when it transfers a stream of mass data to the main memory.

Using the logic-per-track concept, researchers have developed a number of "search engines." A major development was the Sure (Such-Rechner) system developed at the University of Braunschweig. Zeidler describes this system in a general survey of existing approaches.[21]

Lee and Lochovsky[22] describe Hytrem, a text-retrieval machine for large databases. It uses associative processing and a signature file that compresses the text pattern of a large database. Typically, this file takes up about 10 to 20 percent of the entire database. Hytrem has two major subsystems: a signature processor that compares a preprocessed search pattern against the signature file, and a pattern matcher that must eliminate false hits caused by considering only the compressed data.

Yamada et al.[23] developed another high-speed search machine. Besides processing logic, it uses a special 8-Kbit CAM, which holds 528 characters in a 16-bit character code. The CAM cell for storing one character consists of eight pair-bit CAM (PCAM) cells. Four conventional RAM bit cells and some hit logic make up a PCAM cell (see Figure 6). A PCAM cell can store one out of 10 different bit patterns, representing, for example, the four different combinations of two stored bits (0/0, 0/1, 1/0, 1/1), and also states like don't care or cleared.

Parhami recently proposed a serial/parallel architecture for a search machine.[24] This architecture consists of a linear array of processing elements, each element being connected to a circular shift register and a systolic array of comparator cells. Parhami shows how these elements cooperate to process a search in strings of data, and estimates the performance trade-off between serial and parallel search strategies. (Recently, Faudemay and Mhiri reported another approach for supporting string search operations with associative circuits.[25])

ARAM-based processors

From search operations, Tavangarian[26] generalized the flag-oriented ARAM concept to the processing of data. The operands of these generalized operations are flags. Initially, a number of u-bit data are compressed into one 2^u-bit-wide flag vector. A system using an extension of the ARAM structure (described earlier) can memorize several of these flag vectors and use appropriate Boolean functions to efficiently process them (often in parallel). Tavangarian describes a complete flag algebra of processing operations such as forming the union or intersection of flag vectors and checking for equivalence or antivalence between the vectors.

Another application of ARAM

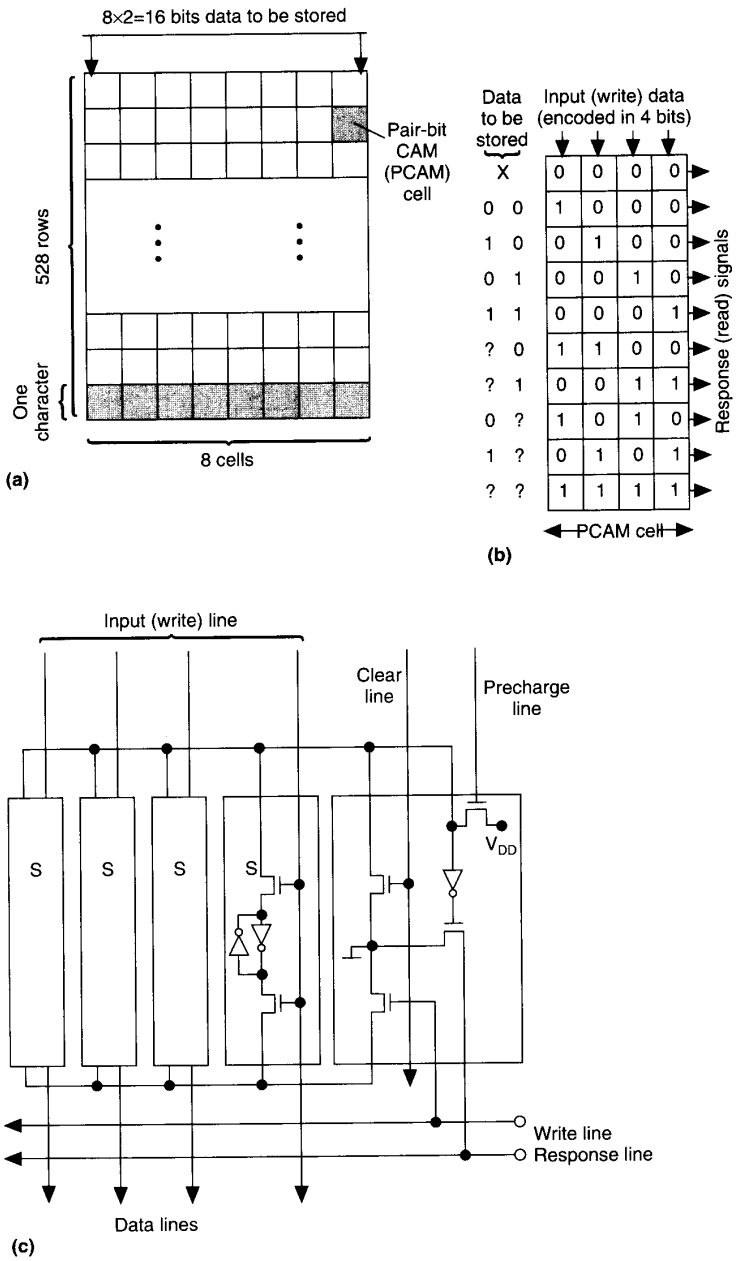

Figure 6. Pair-bit CAM cell: general bit cell array (a), coding of data (X indicates cleared, and ? indicates don't care) (b), and circuit structure (c). S indicates a SRAM bit cell.[23]

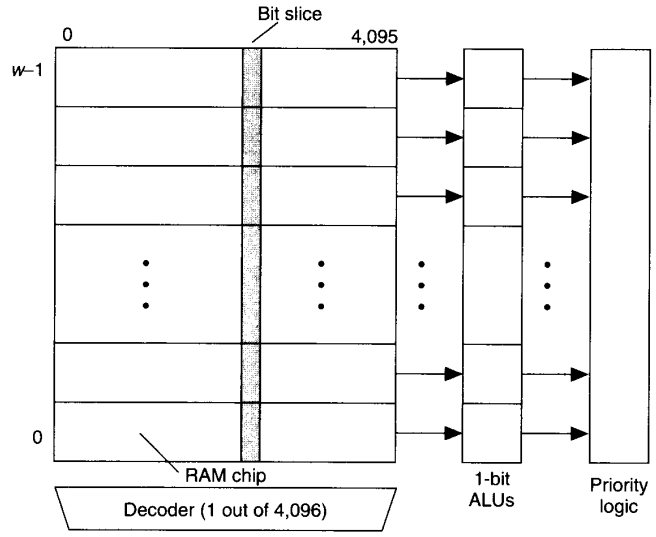

Figure 7. LUCAS architecture.[28]

components is their use in the associative processor AM³ (associative multipurpose microprogrammable monoprocessor). This processor, based on AMD 2900 chips connected with associative memory, is intended mainly for applications like sensor control and speedup of CAD workstations.[27]

Parallel cellular logic

Some associative systems not only place CAM bit cells into a surrounding processing logic, as described earlier, but additionally integrate functionally complete arithmetic or Boolean processing elements into the individual word or bit cells.

A first approach toward this goal was the LUCAS (Lund University Content-Addressable System) architecture developed by Fernstrom, Kruzela, and Svensson.[28] LUCAS is a bit-sequential, word-parallel associative processor system. The CAM words are very long (4,096-bit word length), and a 1-bit arithmetic logic unit is associated with each word (see Figure 7). In hardware, one ordinary 4-Kbit RAM chip realizes each memory word. The system implements the entire search by transferring the memory content—bit slice by bit slice—to the ALU slice. The ALUs compare the bits of a bit slice in parallel with the bits of the given search pattern.

The advantage of this architecture is its easy implementation with already existing hardware building blocks. The tradeoff is the (often awkward and time-consuming) bit-slice-sequential organization of data processing and even simple retrieval operations. The authors also present a set of benchmark examples for evaluating the time complexity of basic application tasks for CAMs.

A recent promising approach for general-purpose associative systems is the associative string processor (ASP) developed by Lea at Brunel University.[29] The ASP system is a dynamically reconfigurable structure of communicating ASP substrings. Each substring contains a set of identical associative processing elements. Each element consists of an n-bit data register (n can vary, dependent on the application class, from 32 to 128 bits), an a-bit activity register (a varies from 4 to 8 bits), an $n + a$-bit parallel comparator, a single-bit full adder, and four status flags.

In another recent approach[30] I've tried to combine the ideas of Lea,[29] Tavangarian,[9] and Waldschmidt[10] and extend their solutions to achieve the following objectives:

- increase the flexibility of the logic elements, and
- combine processor cell arrays with ordinary CAM and RAM parts.

The main architectural features are

- extension, with relatively low hardware effort, of the usual comparison logic of a CAM cell to provide an entire set of 1-bit Boolean operations;
- extension of arithmetic elements from one sequential 1-bit adder element per word cell to at least 4-bit adder elements; and
- features for multiaccess to memory cells and ALUs.

A more detailed discussion of this architecture will appear in a future issue of *IEEE Micro*.

THIS ARTICLE SURVEYS THE GROWING number of recent research projects and implementations in the field of associative processors and memories. The increasing spectrum of approaches shows that we now have the potential for a renaissance of interest in these structures, especially as add-ons to existing architectures. Associative architectures can considerably improve the performance of today's computing systems in a growing spectrum of tasks; some of these potential applications will be discussed in more detail in the

subsequent articles in this issue. We might be seeing now—for the first time in the history of designing associative systems—a real chance to market them.

References

1. T. Kohonen, *Content-Addressable Memories*, Springer-Verlag, Berlin, 1980.
2. L. Chiswin and R.J. Duckworth, "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *Computer*, Vol. 22, No. 7, July 1989, pp. 51-64.
3. H. Kadota et al., "An 8-Kbit Content-Addressable and Reentrant Memory," *IEEE J. Solid-State Circuits*, Vol. 20, No. 5, Oct. 1985, pp. 951-956.
4. T. Ogura, S. Yamada, and T. Nikaido, "A 4-Kbit Associative Memory LSI," *IEEE J. Solid-State Circuits*, Vol. 20, No. 6, Dec. 1985, pp. 1277-1282.
5. 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.
6. J.P. Wade and C.C. Sodini, "Dynamic Cross-Coupled Bit-Line Content-Addressable Memory Cell for High-Density Arrays," *IEEE J. Solid-State Circuits*, Vol. 22, No. 1, Feb. 1987, pp. 119-121.
7. A. Kokubu, M. Kuroda, and T. Furuya, "Orthogonal Memory—A Step Toward Realization of Large Capacity Associative Memory," *Proc. Int'l Conf. VLSI*, North-Holland, Amsterdam, 1985, pp. 159-168.
8. H.E. Batcher, "STARAN Parallel Processor Hardware," *AFIPS Conf. Proc.*, Vol. 43, 1974, pp. 405-410.
9. D. Tavangarian, "Ortsadressierbarer Assoziativspeicher," *Elektronische Rechenanlagen, (Location-Addressable Associative Memory)*, Vol. 25, No. 5, 1985, pp. 264-278.
10. K. Waldschmidt, "Associative Processors and Memories—Overview and Current Status," *Proc. Compeuro*, 1987, pp. 19-26.
11. C.C. Foster, *Content-Addressable Parallel Processors*, Van Nostrand Reinhold, New York, 1976.
12. K.J. Thurber and L. Wald, "Associative and Parallel Processors," *Computing Surveys*, Vol. 7, No. 4, Dec. 1975, pp. 215-255.
13. S.S. Yau and H.S. Fung, "Associative Processor Architecture—A Survey," *ACM Computing Surveys*, Vol. 9, No. 1, Mar. 1977, pp. 3-27.
14. J.G. Delgado-Frias and W.L. Moore, eds., *VLSI for Artificial Intelligence*, Kluwer Academic Publishers, Boston, 1989.
15. P. Kogge et al., "VLSI and Rule-Based Systems," in *VLSI for Artificial Intelligence*, J.G. Delgado-Frias and W.R. Moore, eds., Kluwer Academic Publishers, Boston, 1989, pp. 95-108.
16. Y. Ng et al., "Unify with Active Memory," in *VLSI for Artificial Intelligence*, Kluwer Academic Publishers, 1989, pp. 109-118.
17. I. Naganuma et al., "High-Speed CAM-Based Architecture for a Prolog Machine (ASCA)," *IEEE Trans. Computers*, Vol. 37, No. 11, Nov. 1988, pp. 1375-1384.
18. S.-I. Chae et al., "Content-Addressable Memory for VLSI Pattern Inspection," *IEEE J. Solid-State Circuits*, Vol. 23, No. 1, Feb. 1988, pp. 74-78.
19. I.N. Robinson, "The Pattern-Addressable Memory: Hardware for Associative Processing," in *VLSI for Artificial Intelligence*, Kluwer Academic Publishers, 1989, pp. 119-130.
20. A.W.G. Duller et al., "Design of an Associative Processor Array," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 374-382.
21. C. Zeidler, "Content-Addressable Mass Memories," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 351-356.
22. D.L. Lee and F.L. Lochovsky, "HYTREM—A Hybrid Text-Retrieval Machine for Large Data Bases," *IEEE Trans. Computers*, Vol. 39, No. 1, Jan. 1990, pp. 111-123.
23. H. Yamada et al., "A High-Speed String-Search Engine," *IEEE J. Solid-State Circuits*, Vol. 22, No. 5, Oct. 1987, pp. 829-834.
24. B. Parhami, "The Mixed Serial/Parallel Approach to VLSI String Processors," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 202-211.
25. P. Faudemay and M. Mhiri, "An Associative Accelerator for Large Databases," *IEEE Micro*, Vol. 11, No. 6, Dec. 1991, pp. 22-34.
26. D. Tavangarian, "Flag-Algebra: A New Concept for the Realization of Fully Parallel Associative Architectures," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 357-365.
27. M. Schulz et al., "An Associative Microprogrammable Bit-Slice-Processor for Sensor Control," *Proc. Compeuro*, Part 3, 1989, pp. 87-95.
28. C.F. Fernstrom, I. Kruzela, and B. Svensson, *LUCAS Associative Array Processor*, Springer-Verlag, Berlin, 1986.
29. M. Lea, "ASP: A Cost-Effective Parallel Microcomputer," *IEEE Micro*, Vol. 8, No. 5, Oct. 1988, pp. 10-29.
30. K.E. Grosspietsch, "An Architecture for an Intelligent Multi-Mode WSI Memory," in *Wafer Scale Integration III*, M. Sami and F. Distante, eds., North-Holland, Amsterdam, 1990, pp. 211-222.

The author's biography, picture, and address appear on p. 11 in this issue.