PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 34, NO. 4, PP. 257–272 (1990)

INTELLIGENT SENSOR-INTEGRATED SYSTEMS BY MEANS OF ASSOCIATIVE PROCESSING

K. E. GROSSPIETSCH

Society of Mathematics and Data Processing
St. Augustin, Federal Republic of Germany

Received: Oct. 20, 1990; Revised: May 2, 1991.

Abstract

The concept of an innovative processor system is presented. It combines the inclusion of processing logic in the bit cells and word cells of a memory with the use of sensor devices for parallel input of optical data. Associative (content-addressable) features further facilitate the processing of data in this 'intelligent memory' approach. Moreover, changes in the memory decoder enable also the parallel access to memory cells.

The basic operations of the extended memory are explained, and the resulting instruction set is presented. Some potential applications in the field of parallel image preprocessing/image processing, and corresponding performance aspects are shortly discussed.

Keywords: Intelligent memory, associative processor systems, optical sensor systems.

1. Introduction

In the practice of data processing we can observe a growing demand for more and more 'intelligent' hardware systems, i. e. subsystems have to take over from the end user a growing spectrum of subtasks within the entire hardware system 'automatically'. For instance, integration of sensors into computing systems implies some kind of intelligent preprocessing; this means transfer of processing tasks from the central host to peripheral devices, thus causing considerable performance speedups.

If such extensions of functionality are realized only at the level of software, this usually causes unacceptable performance results. As an alternative, the additional functionality can be transferred into hardware outside the central host processor.

Today the advanced hardware technology (very large scale integration VLSI and wafer scale integration WSI) implies the potential to efficiently implement also new architectures which formerly could not be realized because of technological restrictions like pin limitations, too small bit capacities, etc. (SAMI and DISTANTE, 1990). In this context, especially those approaches appear interesting which are to eliminate the 'bottleneck' between processor and data: In conventional von Neumann machines, every time when data are to be manipulated, they have first to be fetched from memory and transferred to the processor. The result of the processing operation has to be stored back in memory. So, for many of these operations, most of the execution time is taken by data transfers.

This principle at the beginning of computing technology of course was dictated by the very high price of the hardware for the processing logic. Today, at the current state of hardware integration technology, this situation has changed. So, now an alternative to the processor/memory bottleneck could be to integrate more logic directly in the memory structure, i. e. to make the memory more intelligent.

In this context, as one interesting step towards smarter memories, also systems for associative (meant here always as a synonym for content-addressable) storage and processing of data can win importance again: Content-addressable memories (CAMs) have been discussed by system developers already for a long time. The benefit of such structures for speeding up search operations is unquestionable, but for a long time the state of technology allowed only very small CAM capacities.

It is now the first time that the state of art in hardware integration, i. e. the current VLSI technology and the advent of wafer scale integration (WSI), promises to enable the implementation of such memories with reasonable capacities and cost/bit ratio. Moreover, the extension of the principle of content-addressing from storage and retrieval of data also to processing of data has been proven to imply many interesting features for various fields of parallel processing, e. g. image processing, CAD support etc. (FERNSTROM et al., 1986, LEA, 1988).

In this paper, an innovative approach is discussed which combines processing logic with ordinary memory functions under associative control; so the resulting system can be called an 'intelligent memory'. Moreover, we will show how this system can be combined with parallel optical sensors; thus, direct parallel input of measured optical data, followed by some kind of processing, is possible. This feature seems especially interesting for applications in optical measurement systems, which, moreover, need internal processing intelligence. Apart from this aspect, the architecture that will be introduced, is a remarkable step towards the mentioned general goal of removing the von Neumann bottleneck.

First, in Section 2 the resulting architecture is introduced, and the side requirements of this specific architecture are discussed. Section 3 describes the user interface of the developed approach in terms of a set of machine instructions. In Section 4, a short report about the status of the implementation is given. Finally, in Section 5 we shall outline some application and performance aspects of the new architecture.

2. The Proposed Approach

Basic Principles and Requirements

The central idea is to achieve potential parallelism and performance increase not by massive replication of conventional processors (systolic arrays, etc.), but by an extension of the memory functionality ('intelligent memory'). So, compared with classical processor arrays, a much larger number of processing elements can be realized; correspondingly, these processing elements tend to be simpler than ordinary processors. A second basic motivation is the investigation of associative properties.

Several interesting approaches for CAMs and content-addressable processor systems have been reported in the last years (FERNSTROM et al., 1986, KOHONEN, 1980, LEA, 1988, TAVANGARIAN, 1985, WALDSCHMIDT, 1987). The approach discussed here is mainly based on the ideas of (LEA, 1988) about associative processor systems; it tries to extend this solution especially with regard to the flexibility of the logic elements and to the combination of memory and processor functions.

This goal is to be achieved by the inclusion of the new features in an existing RAM structure. RAM, CAM and a content-addressable processor/register array (CAPRA) together form a kind of storage hierarchy where the main part can consist of an ordinary RAM (see *Fig. 1*).

Compatibility between these steps of a hierarchy is achieved in the sense that 'smarter' parts also provide the full functionality of the simpler ones. So, the CAM parts are requested also to be operable as RAMs; the CAPRA is implemented by some functional extension of the CAM architecture. As a consequence of the basic RAM property of all the three components, the additional 'smarter' components are included as additional memory segments within one uniform physical memory space. Their individual bit capacities can arbitrarily be tailored to the specific needs of the application, i. e. it depends on the application how many CAM or CAPRA words are added to the basic RAM parts.

With respect to more flexible logic features, the following main architectural properties were felt promising to support a more flexible use of content-addressable memory/processor systems:

- (a) With relatively low hardware effort the usual comparison logic of a CAM cell can be extended to provide an entire set of one bit Boolean operations.
- (b) Extension of arithmetic elements from one bit adder elements per word cell as used in (LEA, 1988) to larger 4 bit adder elements;

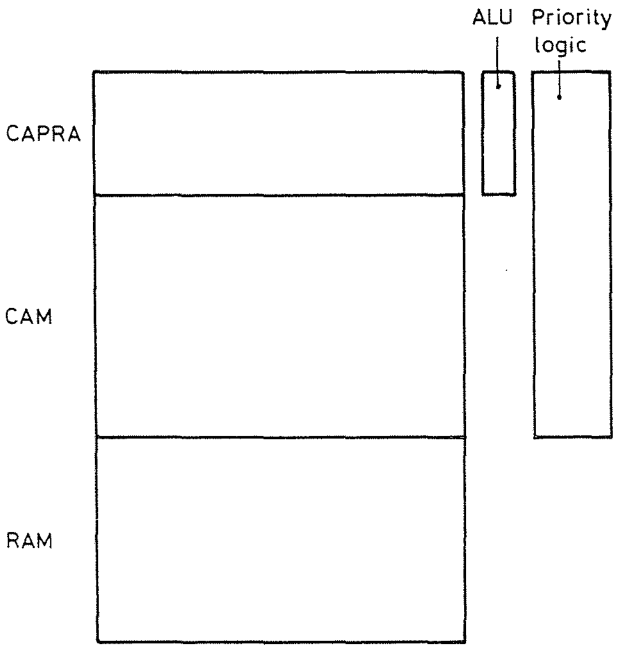

Fig. 1. Schematic structure of the proposed architecture. Dashed lines: boundaries of word cells, C CAM bit cell with comparison logic, E extended bit cell with Boolean logic, R normal RAM bit cell

moreover, for every ALU in the CAPRA array, a data path connecting it with its two nearest neighbour ALUs is provided: So, parallel exchange as well as parallel arithmetic processing of data can be provided.

- (c) Integration of an optical sensor directly into the extended bit cells; this enables a fully parallel input of optical data, thus removing the von Neumann bottleneck.

In the next Subsection, we shall discuss the properties of this requested architecture in more detail; it will turn out that the dimensionality in which the additional components can be realized, is strongly dictated by the implied hardware effort (measured in terms of integrated transistor functions or chip area).

The Resulting Architecture

Our approach specified to fulfil the mentioned requirements has the following characteristics:

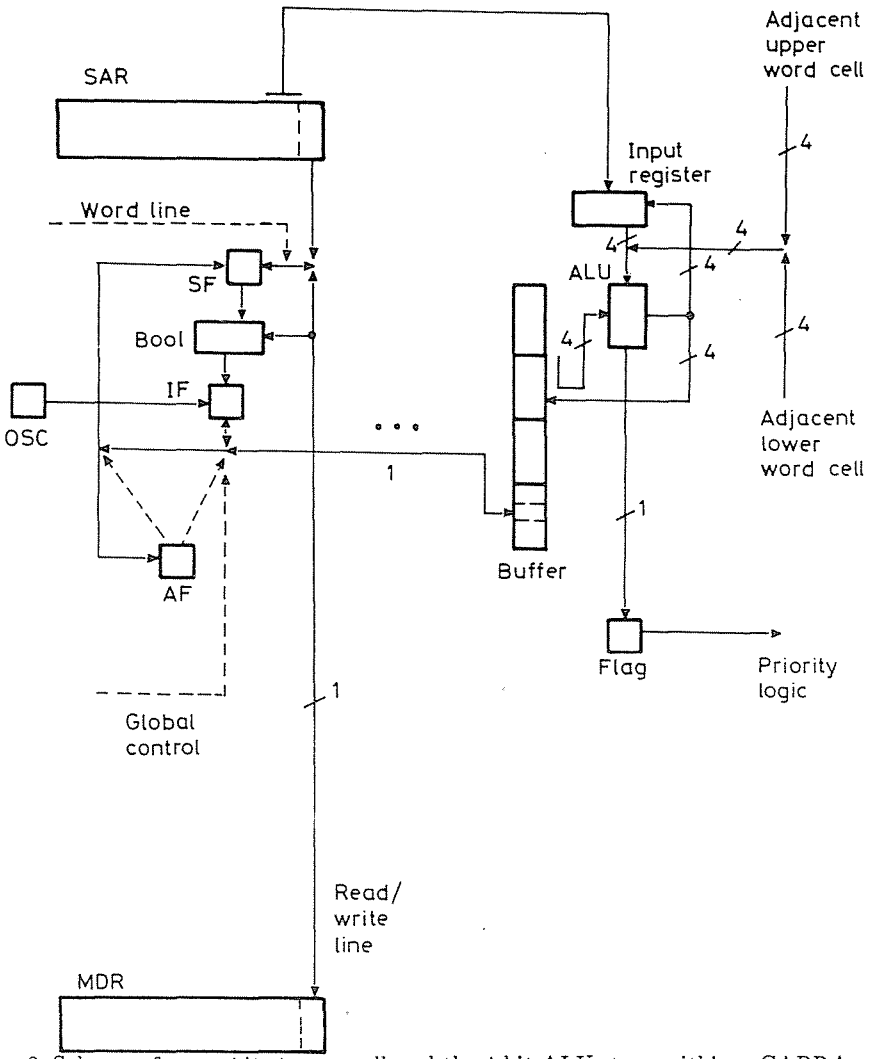

Fig. 2. Scheme of a one bit storage cell and the 4 bit ALU stage within a CAPRA word cell, AF — activity flag, BOOL — logic block for Boolean operations, BUFF — ALU input buffer, IF — intermediate flipflop, MDR — memory data register, OSC — optical sensor circuit comprising phototransistor, sense amplifier and analog/digital converter, SAR — search argument register, SF — storage flipflop, ⟶ data lines, -- → control lines

- The CAM has also an additional RAM access mode so that it can e. g., be loaded or read like ordinary RAM cells.
- In the CAPRA (see *Fig. 2*) segment, as the base cells we again have RAM bit cells; in addition, we have here the following features:

1) To every bit cell a simple logic block is associated which enables Boolean operations between two one-bit operands; this enables bit-parallel and word-parallel execution of a Boolean operation on all words of the CAPRA. It can be shown that such an extension can be realized by an acceptable hardware effort: In the simplest technology, 8 additional transistor functions are sufficient to realize the Boolean unit (MEAD and CONWAY, 1980); this seems acceptable when compared with the six transistor functions of a normal static RAM flipflop cell.

2) To every word cell in the CAPRA (the word length classically being $n = 32$ bits), a simple 4-bit adder/shifter unit is assigned; so, an arithmetic operation can be performed on all the words of the CAPRA segment in parallel in about 8 cycles. Two objectives led to this dimensioning of the ALU part: Such a data width could especially support the processing of pixels with 16 grey levels (coded in 4 bits) in image processing. On the other hand, estimations showed that such an ALU width causes only 30 percent additional chip area in the CAPRA segment when compared with a memory word of a classical length of 32 bits. So, this is an effort which is still in balance with the area effort of the bit cell array itself.

3) An additional flag bit is associated to each bit cell of the CAPRA; this enables us to flexibly define arbitrary 'activity patterns' for the cells of the array; so it is possible to process data not only on all processor elements, but instead on a previously defined arbitrary sub-pattern of processing elements.

4) For the RAM access, a simple extension of the memory decoder as proposed in (TAVANGARIAN, 1985) is provided by means of an additional mask register; by setting bits of this mask register to 1, an arbitrary part of the address bits can be declared to be 'don't care' bits; thus, it is possible to implement concurrent access to word cells which in their addresses have common address bit subpatterns. So, it is possible to write the same bit pattern into all selected word cells, thus performing a very fast common initialisation of these word cells. Moreover, this mechanism is also used to trigger, for a given set of words, the corresponding ALUs to be active or passive, via the setting of an ALU activity flag.

5) Additionally a sensor element is provided for every bit cell of odd bit position i ($i = 1, 3, \ldots, n-1$; n being the length of the CAPRA data word) together with its right neighbour bit cell, i. e. one sensor is shared by a pair of two adjacent bit cells. This sensor element consists of a phototransistor, a sense amplifier and a programmable analog/digital (A/D) converter. The phototransistor is realized as a MOS transistor with floating bulk substrate; this was shown to be an efficient method to integrate photodetectors into

a CMOS process without process modifications (KLINKE et al., 1991). So, whereas most sensor solutions are implemented in an analog technology which is hardly compatible with components of digital MOS technology, our sensor solution can easily be combined with the other subcomponents of the extended bit cell. The sensitivity of the phototransistor depends on its operating point and on the wavelength of the incident light (KLINKE et al., 1991). To realize the A/D converter with programmable resolution, but without any precision elements, the cyclic conversion technique (ONODERA et al., 1988) was used. The output of the converter is shared by the corresponding bit cell and its right neighbour bit cell, so that digitized bits of the converted analog input can subsequently be stored in the flipflops of this bit cell pair.

Operation of the Extended Bit Cell

In the following, we shall focus on the operation of the CAPRA structure (see *Fig. 3*):
Data can be written into a word cell — controlled by the corresponding word line emanating from the memory decoder — via the MDR and read/write lines. For every bit of a word cell, its contents stored in the storage flipflop SF can be combined with the contents of the read/write line by a Boolean operation in the functional block BOOL; the result of this operation is latched in the intermediate flipflop IF. From there it can be propagated further (control line TRANSFER in *Fig. 3*) either to the adjacent ALU (line GLOBAL/LOCAL = 1), or to be memorized in the storage flipflop SF (GLOBAL/LOCAL = 0). These transfers are performed either unconditionally (control line UNCOND = 1) or dependent on the status of the cell which is memorized in the activity flag AF (control line COND = 1, if AF stores a '1'). As a third sink for the bit transfer from the intermediate flipflop IF, setting of the activity flag is possible (control line SET FLAG). This is performed either unconditionally (control line UNCOND = 1) or dependent on the present status of AF (control line COND' = 1); in the latter case, setting of AF is possible only if AF has not yet been set, i. e. if AF is storing a '0' (control line COND' then equals 1).

The intermediate flipflop can receive a data bit not only from the functional block BOOL, but, alternatively, also from the adjacent ALU (control line REC = 1), or from the optical sensor element (control line SCAN = 1).

Digitizing of analog optical data by the A/D converter is done by means of the cyclic conversion method. The converter performs a conversion of m bits accuracy in 3m cycles. For the purpose of storage in the

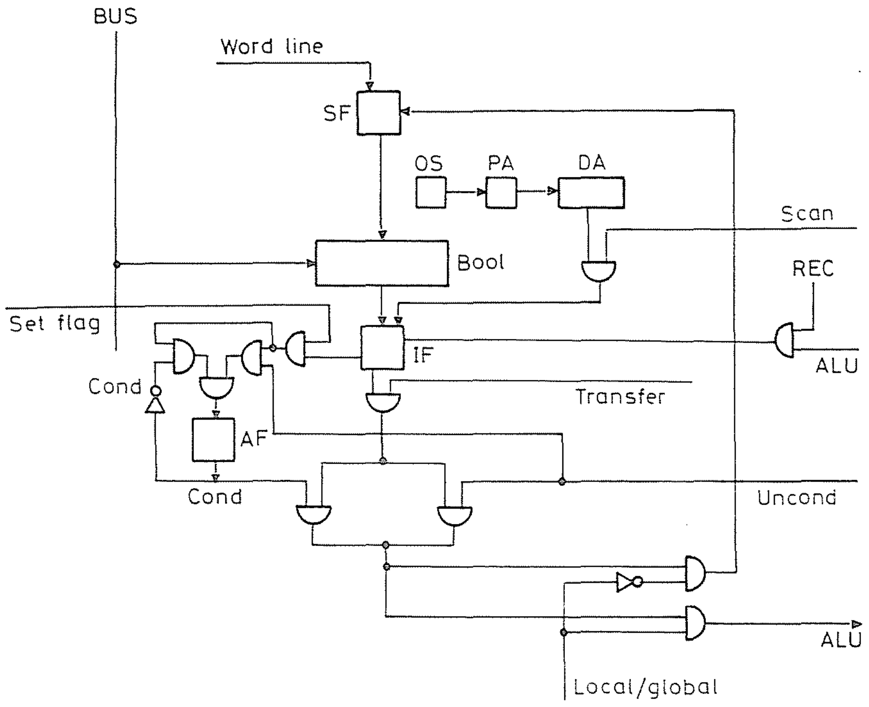

Fig. 3. Logic level structure of a bit cell. AD — analog/digital converter, PT — phototransistor, SA — sense amplifier

extended bit cells, an accuracy of 4 bits was specified. These 4 bits are subsequently stored (via the intermediate flipflops IF) in the SF and AF flipflops of the pair of bit cells, to which the sensor element is associated.

3. Elementary Machine Instructions

For the described memory structure we have defined the following set of machine instructions:

- WRITE, ADR; / Normal RAM write access (executable in all system parts).
- READ, ADR; / Normal RAM read access.
- MWRITE, ADR, MASK; / Masked RAM write access: multiple access to a set of word cells in memory which have some address subpattern in common.

- ASSOCOMP; / Word-parallel and bit parallel comparison of the contents of word cells with a predefined search pattern (executable in the CAM and in the CAPRA part).
- BOOLOP; / Boolean operation combining the bits of the memory words with the bits of an external operand.
- SCAN; / Transfers the digitized bits of the optical input from all the A/D converters to the intermediate flipflops of the corresponding bit cell pairs.
- STORE, COND (UNCOND); / Stores in the bit cells of the CAPRA part the result of the Boolean operation either unconditionally (i. e. for all bit cells in the CAPRA) or conditionally (i. e. depending on the local activity flags of each cell).
- SET AF, COND (UNCOND); / Transfers the contents of IF into the activity flag either unconditionally or conditionally (i. e. only for those bit cells where AF = FALSE).

The ALU operations in the CAPRA can be grouped into two classes:

- unconditional ALU operations
- conditional ALU operations

Operations of the second class exactly correpond in their structure to those of the first class except that they are executed in a local ALU only if an activity flag of this ALU is set to TRUE.

The unconditional operations all have the structure ALU OP, BUFFER(j), REGA (REGB, REGC, SAR), BUFFER(j) (REGA); thereby the first operand BUFFER(j) is always a 4-bit segment of all memory words; the position of this segment is given by the index j. As the second operand, we may use either the register REGA, REGB, REGC or the search argument register SAR (the latter register usually is an interface register for the CAM search operation). Here, REGA is a 4 bit register corresponding to each ALU in the words of the CAPRA. REGB and REGC just represent the REGA register of the upper (lower) neighbour ALU. As an alternative, the least significant 4 bits of the SAR can be used to provide one global 4 bit operand to all the ALUs of the CAPRA.

So, by selection of the second operand, it is possible to combine an operand held in a memory word either with local data residing in the corresponding ALU, in one of its neighbours (thus enabling communication between neighbours) or with a global operand provided from outside.

The ALU operations are executed either unconditionally on all word cells of the CAPRA or conditionally controlled by a local ALU flag. The setting of these flags is performed dependent either on the outcome of certain ALU operations (as usual also in conventional ALUs) or explicitly from outside. For this purpose we have the operation SET ALU FLAG, ADR, MASK. This operation provides — similarly to the masked write operation MWRITE — access to the flag of one or several ALUs in one cycle.

To summarize, the instruction set represents the set of operations of CAPRA which are directly supported by the hardware. All more complex algorithmic tasks are intended to be realized in terms of these basic programming building blocks. A number of programming examples is presented in (GROSSPIETSCH, 1991).

4. State of the Implementation

A complete specification and simulation of the proposed architecture was recently completed. This description was performed in the hardware description language VHDL. It was realized at switch level, i. e. each individual transistor of the entire architecture is modelled, but a part of their electrical properties is neglected (these properties are studied by well-known transistor simulators like SPICE). The basic physical properties, however, of the MOS transistor, namely delay times, gate capacity, and leakage current, were included in our VHDL description. Our switch level model of the transistors was realised for NMOS and PMOS transistors, as well as CMOS complex gates. Based on these transistor primitives, in a second step we developed more complex VHDL constructions to describe gate and flipflop structures. All the introduced VHDL buiding blocks were then used to implement the simulation of the entire system. The complete description of the system (including also e. g. peripheral circuits, priority logic, etc. which are not discussed in this paper) comprises about 6,000 VHDL statements. A complete description can be found in (KOHN and SCHAEFER, 1991).

For several of the individual CAPRA components, the physical layout has already completely been designed (KLINKE et al., 1991). With regard to the rest, simulation at the electrical level (by means of SPICE) and the corresponding generation of layout structures are currently nearly completed. The resulting basic bit cell in CMOS implementation turned out to comprise 82 transistor functions.

For the ALU slices, we consider a simple and concise structure as proposed in (KAISER, 1980, TEXAS INSTRUMENTS, 1978). This ALU structure can perform 16 Boolean and 16 arithmetic operations; correspondingly, it needs 4 control lines for operation selection and 1 control line for the selection between the Boolean and the arithmetic mode.

It cannot be stressed too strongly that the investigations described here are basic research, and the resulting implementation is an experimental one. As the system is not yet completed, about performance aspects only some preliminary results can be presented.

In addition to the component analysis mentioned above, by means of the complete VHDL simulator, also studies for detailed performance evaluation were planned and have already started. A cross-assembler for the introduced machine operation language has been implemented. The assembler translates source programs into code that can be interpreted by the simulator. To show the 'software' properties of the CAPRA structure and of its potential use in interaction with classical RAM and CAM parts, a package of demonstration software for various applications is being developed. It is beyond the scope of this paper (and, for most applications also too early) to present detailed performance results. However, in the final section we shall shortly discuss some application areas where especially the combined use of the described processing logic together with optical sensor appears very promising, with regard to performance.

5. Potential Sensor-Oriented Application Fields

In general, two main classes of applications appear promising for CAPRA-like architectures:

- Applications where data are set-organized.
- Applications where data are array-organized.

The first class can mainly be found in the area of data base applications. Here, it was shown (GROSSPIETSCH, 1991) that the use of the CAPRA architecture (without sensors), especially of the comparison operations in combination with multiple cell activations, leads to tremendous performance improvements for relational data base operations.

With respect to sensor-oriented systems, however, the second class of applications seems more interesting. Array-structured data e. g. appear in many picture processing or pattern recognition tasks. In the following, we shall shortly examine some aspects of the CAPRA architecture for this field.

Neighbourhood Operations

One of the very basic operations for pixel processing is the comparison of a pixel point with its 4 or 8 nearest neighbours (GONZALEZ and WINTZ, 1987). In CAPRA, the comparison with an 8-neighbourhood can be solved in the following way (see *Fig. 4*): In a first phase, to compare a bit j in word cell i with its left and right neighbour $j-1$ and $j+1$, the ALU of word cell i has to fetch 3 bits from its own word cell. This operation and the subsequent comparison of the bits in the ALU can be performed concurrently for all words in the CAPRA representing pixel data. So, only all ALUs of the CAPRA have to be activated before by one masked write operation setting the ALU activity flag to 1.

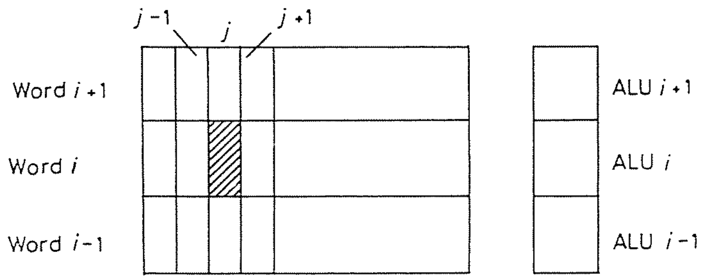

Fig. 4. Scheme of a neighbourhood comparison for pixel processing

In a second phase, each ALU i then has to fetch 3 bits from the bit positions $j-1$, j, and $j+1$ of its upper neighbour word cell via ALU $i-1$ and compare them with bit j of word i still held in ALU i. This operation can again be done concurrently for all the words in the CAPRA. In a third phase, a corresponding procedure is carried out to fetch bits from the lower neighbour word cell and to process them in the same way.

Sequentially, the pattern of the basic pixels j which are to be compared with their neighbourhoods, is moved over the 32 bits of the CAPRA word length. So, to summarize, the execution of neighbourhood operations on the entire pixel array can be performed in the order $O(n)$ steps (n being the word length in CAPRA), independent of the number of pixels in the array.

More Complex Image Processing Algorithms

Many image processing algorithms are based on the use of neighbourhood comparisons as basic building blocks. It is beyond the scope of this paper to discuss these algorithms in more detail; a comprehensive discussion of such algorithms is presented in (FERNSTROM et al., 1986). This book also analyses the performance of associative processor systems for image processing tasks, by the example of the LUCAS array processor developed at the University of Lund. LUCAS is a word-parallel, bit-sequential associative processor. With regard to the considered algorithms, CAPRA's performance is of the same order as that of LUCAS; however, CAPRA is superior by a constant factor of about 4, as we have 4 bit ALUs instead of the purely bit-sequential processing in the LUCAS words.

Image Preprocessing

One class of such more complex image transformations is image preprocessing to reduce data and to extract image features. Here, the combination of optical sensors and processing power within the CAPRA words offers to directly manipulate optical data during the period of scanning them into the system. This is especially useful for applications where a reduction of the original image data is essential for subsequently processing them efficiently.

Many image preprocessing algorithms are again based on simple neighbourhood operations. We consider here only one characteristic example: An operator that is often used for this purpose, e. g. for image skeletonization or contrast enhancement, is the so-called Sobel operator (GONZALEZ and WINTZ, 1987). For every pixel of a pixel array, this operator generates gradient vectors G_x and G_y in the x- and y-direction. This is approximated by constructing a weighted sum of the values of neighbourhood elements:

$$G_x = (x_7 + 2x_8 + x_9) - (x_1 + 2x_2 + x_3)\,,$$

$$G_y = (x_3 + 2x_6 + x_9) - (x_1 + 2x_4 + x_7)\,.$$

Here, the centre of the neighbourhood is represented by pixel x_5, and the neighbours by $x_1 \dots x_4$, $x_6 \dots x_9$. Images generated from these gradient values enhance the contrast which existed in the original image.

It can be seen that the Sobel operator only needs some elementary operations as addition, subtraction, and multiplication by 2. These operations can easily be realized by the CAPRA ALUs.

Such data reduction might be of advantage also for adaptive systems, e. g. neural nets: With respect to the direct hardware implementation of neurons, there is still the problem that the internal complexity of neural representations usually is very high. So a preprocessing phase reducing data and extracting relevant features could considerably speed up neural algorithms (HOSTICKA, 1991). A further benefit would be if such preprocessing could directly be combined with the scanning of the original neural data. Here, especially for the recognition of optical patterns by neural algorithms, the CAPRA architecture seems very interesting as it enables the combination of direct parallel input of optical data together with parallel preprocessing. Such neural solutions, based on preprocessing phases, were recently proposed for applications like optical character recognition or optical inspection of chips (HOSTICKA, 1991).

General Performance Aspects of CAPRA

It has turned out that e. g. neighbourhood operations on a pixel array can be performed in the CAPRA system in the order $O(n)$, n being CAPRA's word length, independent of the number of words (i. e. also number of ALUs) in the CAPRA. Such a result is comparable to other processor arrays with the same number of ALU elements. This also holds for most other examples of simple numerical operations where neighbourhood regions can be processed more or less independently.

In general, it has to be remarked that when a comparison with other processor arrays (e. g. certain systolic arrays) is made for a given task example, the results often depend on whether the loading phase of the system is neglected or not (also most systolic arrays are intended to work as coprocessors for some host; so, usually they have to be sequentially loaded with data from the host).

As a purely numerical example, the multiplication of an $\mathrm{N} \times \mathrm{N}$ matrix with an N component vector (residing both in the system) can be shown to be carried out in CAPRA (by parallel multiplications and by adding up and bringing together the partial sums in a tree-like manner) in the order $O(\sqrt{\mathrm{N}})$. If loading e. g. of the vector is also taken into account, the entire procedure is of order $O(\mathrm{N})$, the same value as resulting for certain systolic array approaches (MEAD and CONWAY, 1980).

It has to be stressed that especially loading of optical data, or broadcasts to all processing elements, can be performed very efficiently in the order $O(1)$. Moreover, the selection of arbitrary subgrids of CAPRA' s bit cell array can be performed very flexibly in a few machine instruction cycles by combining the use of the ASSOCOMP search instruction, with masked WRITE instructions for multiple access to word cells or ALUs. With regard to all these innovative aspects, comparative performance figures from other processor array approaches are completely missing.

Apart from that, the complexity of the individual processor elements in CAPRA is considerably lower than in most present processor arrays. For many tasks where only relatively simple operations of the array are required, this is, however, no tradeoff; on the other hand, the simpler structure of the processing elements allows to implement a much larger number of them on a given wafer area, compared with (systolic) arrays of more sophisticated processors.

The simple examples discussed seem very promising; nevertheless it should be kept in mind that these are first, preliminary results; evaluation of a comprehensive set of benchmark examples is in progress, but not yet completed. Additional performance results for a similar associative processor system can be found in (LEA, 1988).

Conclusion

In this paper, an innovative architecture for associative processing has been described. Its basic idea is to combine processor and memory function within one basic word cell ('processing memory'). The normal bit cell of RAM architectures is extended to include also Boolean logic and a sensor element; moreover, some ALU slices are assigned to each word cell.

It has been outlined that this architecture is interesting especially with regard to 'intelligent' preprocessing of scanned data, or to general image processing and pattern recognition tasks.

References

FERNSTROM, C. F. – KRUZELA, I. – SVENSSON, B. (1986): LUCAS Associative Array Processor. Springer-Verlag, Berlin, Heidelberg, New York.

GONZALEZ, R. C. – WINTZ, P. (1987): Digital Image Processing. Addison-Wesley Publishing Company, Reading (Mass).

GROSSPIETSCH, K. E. (1991): Associative processing (Invited contribution). In: Soucek, B. (ed.), Fast, Invariant, Dynamic and Parallel Intelligence, John Wiley and Sons, to appear.

HOSTICKA, B. J. (1991): Neue Architekturen für die Signalverarbeitung (New Architectures for Signal Processing). In: Zimmer, G. (ed.), *Proc. GME-Fachtagung 'Mikroelektronik'*, Baden-Baden 1991, VDE-Verlag, Berlin Offenbach, pp. 1–9.

KAISER, J. (1980): Fehlerdiagnose in einem hochzuverlässigen Einprozessorsystem (Fault Diagnosis in a Highly Reliable Uniprocessor System). *GMD-Studie* Nr. 52, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, Germany.

KLINKE, R. – BROCKHERDE, W. – HOSTICKA, B. J. – ZIMMER, G. (1991): Eine Photodetektor-Matrix mit Ausleseelektronik in Standard-CMOS-Technologie (A Photodetector Matrix with Readout Electronics in Standard CMOS Technology). In: Zimmer, G. (ed.), *Proc. GME-Fachtagung 'Mikroelektronik'*, Baden-Baden 1991, VDE-Verlag, Berlin Offenbach, pp. 175–180.

KOHN, M. – SCHAEFER, U. (1991): Entwurf und Simulation eines assoziativen Prozessorsystems (Design and Simulation of an Associative Processor System). *Diploma Thesis*, University of Bonn.

KOHONEN, T. (1980): Content-addressable Memories. Springer-Verlag, Berlin, Heidelberg, New York.

LEA, M. (1988): ASP: A Cost-Effective Parallel Microcomputer. *IEEE Micro*, October 1988, pp. 10–29.

MEAD, C. – CONWAY, L. (1980): Introduction to VLSI systems. Addison-Wesley Publishing Company, Reading (Mass.)

ONODERA, H. – TATEISHI, T. – TAMARU, K. (1988): A Cyclic A/D Converter Technique that Does not Ratio-matched components. *IEEE Journ. of Solid State Circuits*, Vol. SC-23, pp. 152–158.

SAMI, M. – DISTANTE, F., (eds.) (1990): *Proc. 3rd IFIP Workshop 'Wafer Scale Integration'*, Como 1989. North Holland, Amsterdam.

TAVANGARIAN, D. (1985): Ortsadressierbarer Assoziativspeicher (Location-addressable Associative Memory). *Elektronische Rechenanlagen*, pp. 264-278.

TEXAS INSTRUMENTS (1978): The TTL Data Book.
WALDSCHMIDT, K. (1987): Associative Processors and Memories: Overview and Current Status. In: Proebster, W. (ed.), *Proc. COMPEURO '87*, pp. 19–26.

Address:

Dr. K. E. GROSSPIETSCH
Institut für Systemtechnik
Gesellschaft für Mathematik und Datenverarbeitung (GMD)
Postfach 1240, D-5205 St. Augustin, Germany