Associative Memory Designs for VLSI Implementation

Behrooz Parhami

Dept. of Electrical & Computer Engineering
Univ. of California
Santa Barbara, CA 93106, USA

ABSTRACT

Associative memories and algorithms for performing various operations on them have been studied extensively over the past three decades. Speedup analyses offered for such algorithms are based on the assumption of a fixed cycle time for the associative memory, independent of size. Whereas this is acceptable for small capacities which have been practical in the past, the potential for realizing very large systems by exploiting advances in the VLSI technology necessitates a reexamination of the above premise. In this paper, we offer designs for associative memories whose cycle times are realistically constant and independent of their size. The designs are based on well-known principles of pipelining and systolic operation using a collection of small building-block associative memories. Several alternative organizations, from a simple linear array to higher-dimensional meshes and trees are examined and evaluated with respect to cost and performance.

Keywords: Content-addressable memories. Parallel processing. Pipelining. Search processors. Systolic systems. Time complexity.

1. INTRODUCTION

Associative or content-addressable memories (AMs or CAMs) have been studied for over three decades as mechanisms for speeding up time-consuming searches and for allowing access to data by name or partial content rather than by location or address [HANL66], [PARH73], [KOHO87], [CHIS89]. Also, more functional variants of such systems known as associative or content-addressable processors [FOST76], [YAUS77] and database computers [SUSY88], [HURS89] have been the subjects of extensive research.

In its simplest form, an AM can be viewed as a hardware device consisting of N fixed-size cells, each being marked as empty or storing a data word or record. We denote the number of nonempty words by n.

When presented with a "search key" (also known as the "comparand") and a "mask" specifying the relevant field(s) of the stored words, the AM responds by "marking" all the words that "match" the specified key. Marking is done by setting or resetting a "response bit" or "tag" in the corresponding AM cell. The N response bits together form the AM's "response store." A "response indicator" mechanism may provide information on the multiplicity of responders (zero, one, several).

If there is no responder, the search outcome is negative and we can proceed to the next step of our algorithm. With one responder, the required data item has been located and can be appropriately dealt with by reading it out or modifying it in-place. If there are several responders, the appropriate course of action might be proceeding to further narrow down the search, simultaneously modifying all responders in-place, or examining the responders in turn through the use of a "multiple response resolver."

Many algorithms have been developed for performing search, retrieval, and arithmetic/logic operations on data stored in AMs. Although such operations can be programmed using the basic "masked exact match" search capability of simple AMs, provision of other types of hardware primitives can have a significant effect on performance.

A common flaw in the analyses offered by designers of AMs and of AM algorithms is the assumption that the basic cycle time of an associative memory is a constant independent of size, thus leading to the optimistic conclusion that an N-word AM offers n-fold speedup compared to linear search in an unordered list of n items ($n \leq N$). While this may be accurate for small N, it is grossly unfair to equate the cycle time of a large associative memory (say, one holding hundreds of thousands of data items) with that of simpler hardware used in conventional systems.

Realistic analyses of the speedup offered by AMs and AM algorithms would require the development of a model that predicts the cycle time of an N-word AM as a function of N. Such a model would depend on the technology used to implement the AM and on its architecture. The dependence of cycle time on N arises from two factors:

1. Broadcasting instructions to all N cells in the AM.
2. Performing global operations on N response tags.

The required cycle time for broadcasting obviously increases as we go from single-chip to multiple-chip and multiple-board systems, simply because longer interconnections will be involved and a larger number of loads must be driven. Although work has been done on modelling of propagation delays over long wires, no universal model exists for predicting the effect of such delays on the required cycle time [BILA82], [CHAZ85].

As for global operations, such as those required for multiple response resolution, the effect of long wires is compounded by the requirement for logic manipulation. It is well-known that even if propagation delays are insignificant, a global operation on N bits by limited fan-in circuits requires time that is at least proportional to $\log N$. Yet even recent works on associative memory architectures and algorithms assume constant time for N-operand global operations such as logical OR and logical AND (see, e.g., [LEED88]).

In this paper, we propose a class of architectures for systolic AMs whose basic cycle time is realistically constant and that achieve a linear speedup of n over an infinite sequence of independent AM operations through the use of pipelining. The actual speedup achieved will be a function of instruction inter-dependencies and will additionally depend on the amount of optimization performed for finding efficient sequencing of instructions. Other benefits of the proposed designs are modularity and scalability which have important implications in testing, configuration, and expansion of AM systems.

2. A SIMPLE ASSOCIATIVE MEMORY

To illustrate the design technique proposed here, we start by presenting the functional specification of a simple AM (SAM). These specifications are adapted from Chapter 4, pp. 39-56, of [FOST76] with minor changes and additions.

As shown in Figure 1, SAM can be viewed as consisting of N cells along with their associated tag bits, a Central Control which includes the Comparand and Mask registers, and a Global Tag Operations Unit that computes and sends a summary of tag bit values to the Central Control. The tag bits can be shifted up or down, thus providing a simple mechanism for inter-cell communication. The Central Control is responsible for fetching and broadcasting instructions and common data to all the cells. The instruction set of SAM is as follows:

SET — Initialize all tag bits to 1.

MATCH — Simultaneously compare the unmasked bits of the comparand to the corresponding bits of every cell. If the comparand and cell contents disagree in any unmasked bit, reset the corresponding tag bit to 0; otherwise, do not change the tag bit.

READ — Obtain the logical OR of the contents of all cells with their tag bits set. The OR function is selected for convenience because it is easy to implement in hardware. It can be replaced by another function, such as logical AND or arithmetic sum, if desired.

WRITE — Copy the unmasked portion of the comparand into the corresponding bits of all the cells with their tag bits on. Do not change the cell contents in the masked bit positions or the tag values. This is sometimes referred to as the "multiwrite" capability.

COUNT — Tell the Central Control how many responders there are (compute the sum of all tag bits).

SHIFT — Move each tag bit value up or down by one position (to the cell above or to the one below).

Many useful programs can be constructed with this simple instruction set. For example, to locate the largest unsigned integer in a particular field, we start with the instruction sequence

```
SET
MATCH  Comparand = X...X1X...X,  Mask = 0...010...0
COUNT
```

where the unmasked bit (the single 1 in the Mask Register) is the most significant bit of the desired field. If the result of COUNT is 1, we are done and a READ operation can supply the desired maximal value.

If the COUNT result is greater than 1, then we must further narrow down the search among already selected cells. In other words, there are several cells storing a 1 in the most significant bit of the search field and any of these cells may hold the desired maximal value. The next narrowing step consists of suitably modifying the Comparand and the Mask values and repeating the process with the next most significant bit of the field.

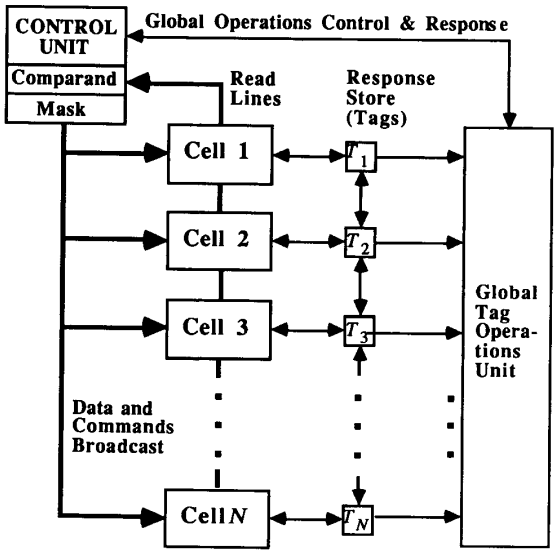

Figure 1. Functional View of SAM: Simple Associative Memory.

Finally, if the result of COUNT is 0, we must restore the previous states of the tag bits (assume we write the tag bits in a scratch field before each phase) and move on to the next bit position. The number of required phases in the worst case is twice the length of the field and is thus independent of the AM size. Foster [FOST76] presents a wealth of other examples for the use of such instructions.

3. SYSTOLIC ARCHITECTURES

To illustrate the design principles for the proposed systolic architectures, we will discuss how a basic AM might be configured from small SAM-like building blocks according to systolic design principles [KUNG80], [KUNG82]. The proposed techniques can be applied to more complex AMs as well as to associative processors by simple extension of the designs and concepts presented here.

3.1. The Building-Block SAM

Clearly a SAM with multiple cells can be realized as a single VLSI chip. The actual number of cells that can fit on such a chip will depend on several implementation details, including internal organization and word length.

The internal organization can be fully parallel, bit-serial, or word-serial [PARH73]. The fully parallel organization is the most complex and offers the highest speed. With the fully parallel organization, a VLSI chip can contain a few tens of cells. With the bit-serial organization, hundreds of cells per chip may become feasible, but speed is also reduced by a factor related to the average field length in associative operations. Finally, the word-serial organization is the least complex and can be implemented by high-speed shift registers. Because of high density storage and sharing of processing logic, thousands of cells may fit on a single VLSI chip, thus achieving significant cost benefits at the expense of further reduction in speed.

Some implications of the speed-cost tradeoffs offered by different internal building block organizations are discussed in Section 4 of the paper. In the rest of this section, we will only be concerned with the functional behavior of such building-block SAMs (BB-SAMs).

Let each BB-SAM contain m cells and its instruction cycle time be c_b. To build an N-word AM from such building blocks, we need $b = N/m$ building blocks or chips. To be able to connect BB-SAMs into a large system, special interface requirements must be met. Obviously, each BB-SAM must be capable of executing all SAM instructions and presenting the required outputs in suitable formats. In addition, extra inputs must be provided for receiving output information from other BB-SAMs in order to internally compute a combined output (remeber that in a systolic system, global control functions are disallowed). Figure 2 shows a possible interface specification for a BB-SAM.

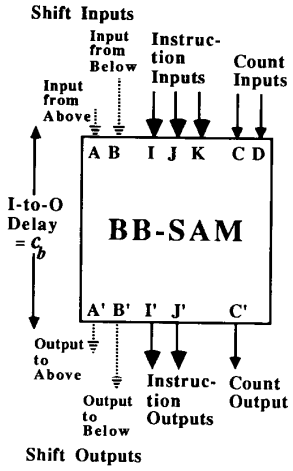

Figure 2. The I/O Interfaces of a Building-Block SAM (BB-SAM).

3.2. Mesh-Connected Organizations

The simplest possible interconnection scheme for BB-SAMs is the one-dimensional mesh or linear array (Figure 3). Although this organization is less efficient than the others to be described subsequently, it will be discussed in detail because its conceptual simplicity provides a basis for understanding the more complex organizations. Similar structures have been suggested with much simpler cells to perform relational database operations on externally supplied data [KUNG80a].

Instructions are fed to the topmost BB-SAM. Upon receiving an instruction, a BB-SAM executes it and forwards the result (if any) along with the instruction itself to the next BB-SAM in the array. Thus, the result of an instruction input at time t will emerge from the last BB-SAM at time $t + bc_b$, where c_b is the cycle time of a BB-SAM. However, the next instruction can be input to the topmost BB-SAM one cycle later (at time $t + c_b$). Instructions follow one another in the array, causing full utilization of the BB-SAMs in the ideal case of an infinite sequence of independent instructions.

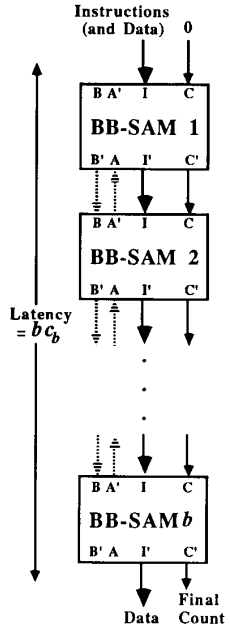

Figure 3. A SAM Structured as a Linear Array of BB-SAMs.

This, of course, is standard pipelining or overlap processing. The number b of stages in the pipeline is potentially very large and full utilization is quite unlikely. Consider for example the problem formulated at the end of Section 2. For this problem, the Central Control must wait for the result of the COUNT instruction after each phase in order to decide what to do next. Thus, only three instructions will be present in the pipeline and performance suffers. One way to improve performance is to save the partial search results and proceed with other computations while waiting for the outcome of some instruction. In the case of the above example, this can be done by

COUNT
WRITE Comparand = X...X1X...X, Mask = 0...010...0

where the single 1 in the Mask Register designates a "scratch" bit into which the current values of the tag bits are stored. The Central Control is now free to initiate other computations.

For example, if determination of the minimal and median values are also needed, phases of the three operations can be effectively interleaved. When the required COUNT value is available, the original tag values are restored by

MATCH Comparand = X...X1X...X, Mask = 0...010...0

and the next instruction in the sequence is issued.

The above discussion was intended as an illustration of the concept of interleaving operations through a pipeline of BB-SAMs. If, as stated earlier, the pipeline has thousands of stages, such interleaving is unlikely to lead to high utilization either because the required number of independent operations are not available in the application or because the number of scratch bits required becomes unacceptably large.

Higher-dimensional meshes are clearly more desirable because of their smaller graph-theoretic diameters. However, whereas the ordering of BB-SAMs in a linear array naturally corresponds to the ordering of SAM cells, there is no correspondingly natural ordering in the case of higher-dimensional meshes. Establishment of such an ordering is essential for proper handling of the SHIFT instruction. We will only consider the case of a two-dimensional mesh. However, extension of the techniques to meshes with higher dimensions is straightforward.

Figure 4 shows a SAM structured as a two-dimensional mesh (henceforth, simply mesh), with the ordering of BB-SAMs given by their index numbers. The mesh has $\sqrt{b}$ rows and $\sqrt{b}$ columns and completes the execution of one SAM instruction in $2\sqrt{b} - 1$ BB-SAM cycles.

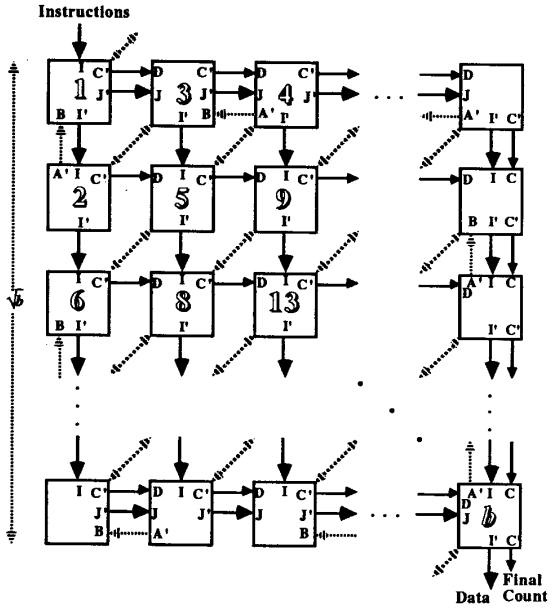

Figure 4. A SAM Structured as a 2-D Mesh.

Such mesh-connected structures have been proposed previously for VLSI dictionary machines [DEHN87], [PROB87], [SCHM85], which although similar to associative memories, have more limited capabilities. Here, we borrow heavily from the techniques offered in these proposals.

Instructions are input at the upper left corner and move downward and to the left. We will assume synchronous operation of BB-SAMs in the following exposition for the sake of clarity. Asynchronous operation, which is much more desirable, is only slightly more complicated in terms of hardware. With the assumption of synchronism, all BB-SAMs located on the same SW-NE diagonal will receive the same instruction simultaneously.

Execution of SET, MATCH, and WRITE instructions is straightforward, since each BB-SAM essentially operates independently. The outcome of READ depends on the contents of all BB-SAMs. However, because of the logical-OR combination rule, it does not matter how many times each BB-SAM affects the result being formed. If another combining operation is specified, then the considerations discussed below for the COUNT instruction apply to READ as well.

To obtain the proper result for the COUNT instruction, it must be ascertained that each BB-SAM affects the result only once. One way to do this is shown in Figure 4, assuming that the count inputs and outputs of a BB-SAM are related by

C' = C + D + Internal Count

and that unconnected count inputs carry zeros. As can be seen in Figure 4, all BB-SAMs except for the ones in the last column adjust the counts and pass them horizonatally (to the right) while those in the last column add the internal count to the sum of C and D and pass the result downward.

Finally, with the ordering of BB-SAMs shown in Figure 4, the SHIFT instruction can be easily executed through information passed on diagonal lines, horizontal lines in the first row, and vertical lines in the first column (dotted arrows in Figure 4).

Consider for example a "shift down" instruction. BB-SAM 1 receives this instruction first, executes it internally, and passes the shifted-out bit along with the instruction to BB-SAM 2, while BB-SAM 3 receives only the instruction. For these two BB-SAMs to execute the shift simultaneously, the last tag bit of BB-SAM 2 must be wired to BB-SAM 3 so that it is available to the latter at the beginning of the second cycle.

In general, the SW-NE diagonal connections carry the first and last tag bit values of a BB-SAM to its diagonal neighbors. In addition, unidirectional connections are required in the first row and the first column for proper execution of the "shift up" instruction for which data movement is against the instruction flow in the pipeline. The input and output terminals for the SW-NE diagonal connections have not been labelled in Figure 4 to avoid cluttering the diagram.

3.3. Tree-Structured Organizations

A tree-structured design seems quite natural for a search processor in view of the effectiveness of tree data structures in the efficient implementation of search operations on sequential computers. Again, similar structures have been suggested for implementing VLSI dictionary machines and database processors [ATAL85], [BENT79], [BROW79], [CHAN88], [GOYA88], [OTTM82], [SCHM85], [SOMA85], [SONG80].

Figure 5 shows a SAM structured as a tree. As was the case for mesh, executions of SET, MATCH, and WRITE instructions are straightforward. READ (even in its generalized form where logical OR is replaced by an arbitrary associative operation) and COUNT are also simple because the tree structure has a built-in

capability for summarizing subtree results without a danger of repetition. The only remaining problem is proper execution of the SHIFT instruction.

It is relatively easy to show that if the cells in each BB-SAM are considered to be consecutive, then no ordering of BB-SAMs in the binary tree structure of Figure 5 will allow the execution of the SHIFT instruction in the sense of assuring that all pairs of consecutive cells are either in the same BB-SAM or in neighboring BB-SAM (connected by a tree link). A more general result (corresponding to the case where the tree nodes and their built-in algorithms are not required to be identical) is stated and proved below.

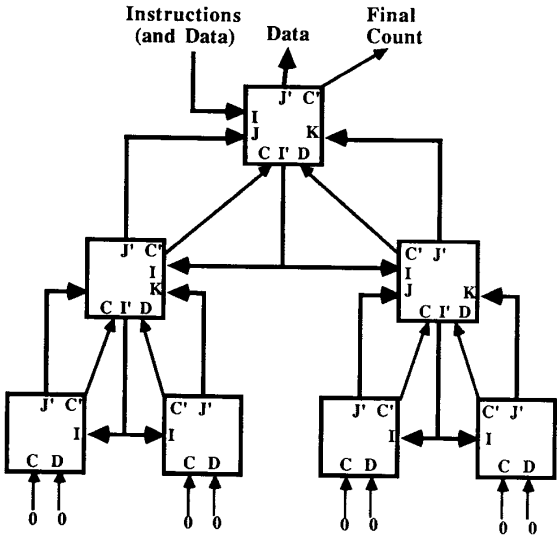

Figure 5. A SAM Structured as a Tree of BB-SAMs.

Theorem: Suppose that disjoint subsets of the set of integers $\{1, 2, \ldots, N\}$ defining a partition on the set are assigned to nodes of a complete binary tree having three or more levels such that for each node B, the corresponding subset consists of consecutive integers (say, $\{i_B, i_B + 1, \ldots, j_B\}$). Then, there exist integers k and $k + 1$ assigned to different nodes B_k and B_{k+1} such that neither node is a parent of the other (i.e., the nodes B_k and B_{k+1} are not directly connected).

Proof: Clearly the theorem does not hold for one-level (single-node) and two-level (three-node) complete binary trees. This is why the number of levels is constrained to be 3 or more. The proof is by contradiction. Assume that the required assignment is made such that no two adjacent values are in non-adjacent nodes. Consider the first three levels of a complete binary tree with the nodes named as follows:

Suppose that the root R is assigned the numbers $i, i + 1, \ldots, j$. Then, one of its children, say S, must contain the consecutive sequence $h, h + 1, \ldots, i - 1$ and the other, say T, the sequence $j + 1, j + 2, \ldots, k$. Furthermore, all numbers assigned to the subtree with root S must be less than i and all those residing in the subtree with root T must be greater than j. For if there are numbers less than i in both subtrees, the conclusion that adjacent values exist in non-adjacent nodes is immediate. Now the values assigned to the subtrees rooted at U and V must all be less than h, which is impossible without having adjacent values in non-adjacent nodes. ■

The above theorem indicates that numbering of the cells in each BB-SAM should not be consecutive. Non-consecutive numbering of cells does not introduce any additional complexity in the execution of SET, MATCH, READ, WRITE, and COUNT instructions. However, it does complicate the logic required for SHIFT and may lead to increased communication complexity between neighbors for the exchange of tag values.

Fortunately, a numbering scheme can be devised that implies insignificant logic overhead and no additional wires for inter-node communication. The propopsed numbering scheme for the cells is shown in Figure 6 for a seven-node tree of BB-SAMs with 5 cells per node.

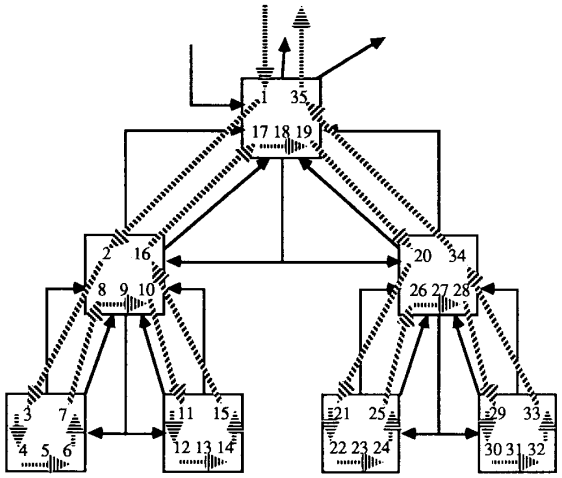

Figure 6. Ordering of Cells in a Tree-Structured AM.

In general, assuming that $m = j - i + 3$, the m cells of a given BB-SAM are numbered $h, i, i + 1, \ldots, j, k$, where all cells in the left subtree have numbers between h and i and all cells in the right subtree have numbers between j and k. This is a generalization of a data storage pattern suggested for systolic search tree structures [CHAN88]. The slight additional complexity in shifting results from the fact that the end cells of a BB-SAM should behave differently according to whether the corresponding node is a leaf or an internal node of the tree.

The number of levels in a balanced binary tree with $N = 2^a - 1$ nodes is a or $1 + \lfloor \log_2 N \rfloor$. Instructions travel from the root of the tree to the leaves and back to the root. Thus, the result of an instruction input at time t will emerge from the root BB-SAM at time $t + (2a - 1)c'_b$, where c'_b is the cycle time of a BB-SAM

for tree interconnection. However, the next instruction can be input to the topmost BB-SAM one cycle later (at time $t + c'_b$). The cycle time c'_b for a tree BB-SAM is larger than the cycle time c_b for a mesh BB-SAM, both because the former is more complex internally and because it is connected to its neighbors by longer wires.

Even though strictly speaking, tree-structured systems cannot be systolic because they require arbitrarily long wires as the number of nodes grows [PATE81], the wire length growth rate is manageable for associative memory sizes likely to be of interest in the foreseeable future.

3.4. Applicability of Other Interconnections

The hypercube network has become a very popular alternative for interconnecting the components of multicomputer systems and has also been proposed for the design of special-purpose dictionary machines [DEHN88], [OMON87], [SCHW87]. Unfortunately, the hypercube itself is not scalable (because its nodes require r connections for a 2^r-node system) and thus does not satisfy the requirements for modular expansion and fixed cycle time. However, a variant known as cube-connected cycles [PREP81], with three links per node, can conceivably be used.

Another potential candidate is the the well-known perfect shuffle network. There also exist a rich array of other interconnection schemes that have been proven useful in different contexts for parallel computers and multiprocessors (see, e.g., [SIEG85]).

A natural question is whether any of these interconnection schemes offers advantages over the mesh and tree structures discussed earlier. Although the application of various interconnection schemes for implementing AMs deserves further study, the following informal argument shows that the answer to the above question is negative when simple AMs such as our SAM system are to be realized.

The cube-connected cycles network provides logarithmic delay but has a richer interconnection structure than the complete binary tree (an N-node binary tree is interconnected by $N - 1$ links whereas an N-node cube-connected cycles network requires roughly $3N/2$ links). Since the binary tree already offers this level of performance, nothing is to be gained from the additional connectivity to compensate for the degradation resulting from longer wires and more difficult wiring and layout. The perfect shuffle interconnection has comparable complexity to that of the cube-connected cycles network but is slightly more difficult to lay out [PREP81].

Similar arguments apply to other interconnection schemes. In any useful scheme, the nodes must have three or more links (the only connected graphs that can be built from nodes of degree 2 are linear arrays and rings). Therefore, the nodes are at least as complex as the tree nodes and the latency is at best logarithmic.

4. ORGANIZATIONAL TRADEOFFS

As mentioned in Subsection 3.1, the number m of cells that can fit on a chip will depend on the internal organization of the BB-SAMs. A slower word-serial organization accomodates more words per chip, thus reducing the number b of chips, and thus the cost, at the expense of a larger pipelining period c_b (lower throughput). A fast fully parallel organization will maximize the throughput but will also increase the cost and possibly the response latency (which may become quite unacceptable in the case of a linear array). A wide range of tradeoffs are thus available to the designer.

Performance/cost tradeoffs must be considered in the design of any parallel system. For example, it has been pointed out in connection with dictionary machines [CARE84], [FISH84] that processor-intensive designs do not necessarily yield significant performance improvements over architectures that match the requirements of the application area but utilize fewer processors.

To illustrate these tradeoffs in more concrete terms, let us examine in some detail the design of a BB-SAM for the linear array interconnection scheme. We consider three implementation alternatives of fully parallel, bit-serial, and word-serial, with the number of cells that would fit on a chip being m, m', and m'', respectively. In general, the parameters for the fully parallel, bit-serial, and word-serial organizations will be denoted by simple, primed, and double-primed variables.

We are interested in two performance indicators: the response latency $L = bc_b$ and the pipelining period $P = c_b$. The hardware cost will be roughly proportional to b, the number of chips. We have $b = N/m$, $b' = N/m'$, and $b'' = N/m''$. Thus, to compare the performance parameters for the three organizations being considered, estimates for the relative magnitudes of c_b, c_b', c_b'' and m, m', m'' are needed.

To obtain the required estimates for c_b, c_b', and c_b'', let us look at the operation of each scheme. In the fully parallel organization, m comparators are built into each chip. Because of the relatively long words that are typical in AMs (say, 128 or 256 bits), rippling time for the match signals in a comparison cycle, and thus c_b, may well exceed 1 microsecond.

Provision of m lookahead logic circuits to speed up the propagation of match signals may of course be contemplated. But this is clearly impractical for all but very small values of m. Such a small value of m will in turn lead to an increase in the number of chips and thus adversely affect the system cost.

In the bit-serial organization, depicted in Figure 7, data is stored in a conventional memory module with bit-slice access. Because bit slices can be read out selectively, the time needed for comparison is proportional to the average length k of the search or data field in MATCH, READ, and WRITE instructions. Thus, we have

$$c_b' = kc_m$$

where c_m is the on-chip memory cycle time for reading out a bit-slice inside the BB-SAM (processing steps in the bit-serial organization can be overlapped with memory access and thus need not be considered separately). Clearly, $c_m < c_b$, with the ratio c_b/c_m potentially exceeding 10.

Finally, in the word-serial organization of Figure 8, the words stored in each BB-SAM are shifted sequentially into the processing section and compared to the common search key or are appropriately modified. Because there is only a single processor in each BB-SAM, it is possible (with low additional cost) to fully pipeline its function such that the processing rate is dictated by the much smaller shift cycle c_s rather than the full comparison cycle c_b. Thus:

$$c_b'' = m''c_s$$

With proper design, c_s should be much smaller than c_m and like c_b/c_m, the ratio c_m/c_s can easily approach or exceed 10.

The relative magnitudes of m, m', and m'' cannot be as easily deduced without the benefit of actual hardware realizations. Taking the intuitive discussion of Subsection 3.1 as a guide, we

can write $m'/m \cong 10$. Combining the above relationships and estimates, we obtain:

$$P = c_b \cong 1\ \mu\text{sec}$$

$$P' = c'_b = kc_m = k(c_m/c_b)c_b \cong (k/10)P$$

$$P'' = c''_b = m''c_s = m''(c_s/c_m)(c_m/c_b)c_b \cong (m''/100)P$$

$$L = Nc_b/m \cong N/m\ \mu\text{sec}$$

$$L' = Nc'_b/m' = Nkc_m/m' = k(c_m/c_b)(m'/m)L \cong (k/100)L$$

$$L'' = Nc''_b/m'' = c_s = m(c_s/c_m)(c_m/c_b)L \cong (m/100)L$$

Because both k and m are almost certainly less than 100, the bit-serial and word-serial organizations perform better with respect to the latency parameter L. On the other hand, since k is likely to be greater than 10 and m'' greater than 100, the fully parallel organization has the edge in terms of pipelining period P or throughput.

Admittedly, the above estimates are quite rough and are therefore subject to refinement. However, the preceding discussion has hopefully convinced the reader that the parameters for the various organizations are close enough to warrant detailed inspection of the three alternatives for any specific application. In particular, it is quite surprising that the very inexpensive word-serial BB-SAM implementation can come so close to matching the performance parameters of the most expensive fully parallel version.

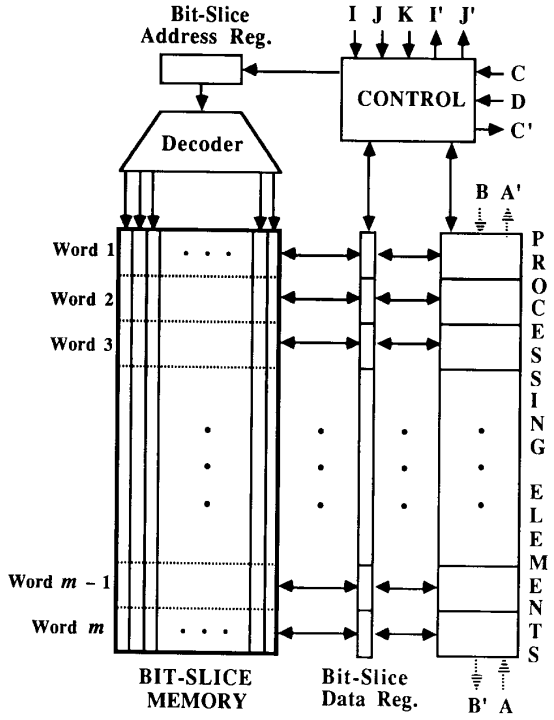

Figure 7. Internal Organization of a Bit-Serial BB-SAM.

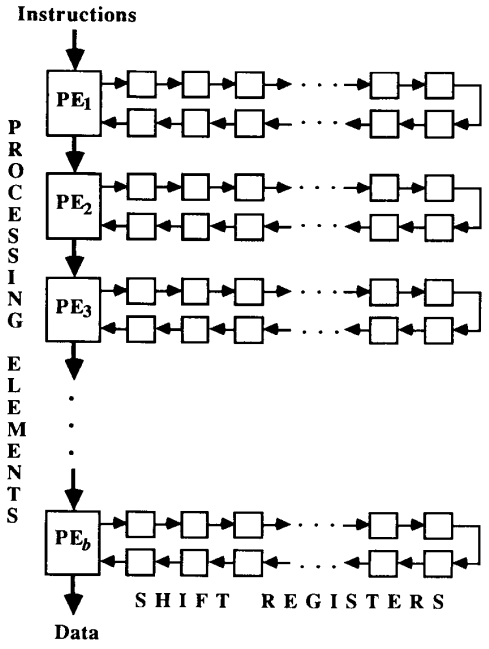

Figure 8. Organization of an AM Based on Word-Serial BB-SAMs.

5. CONCLUSION

We have suggested systolic architectures for associative memories, resulting in systems whose performance parameters are realistically independent of size for long sequences of operations with proper optimization of instruction sequencing. The proposed architectures should lead to practical VLSI realizations of large associative memories which would be impossible to implement under the "operand-broadcasting" and "reduction-by-wired-logic" paradigms.

Although the presentation was in terms of a synchronous mode of operation for the building blocks (BB-SAMs), asynchronous operation is more appropriate in practice and quite easy to incorporate into the proposed systems. Similarly, an actual system is likely to have a richer instruction set consisting of a variety of searches, storage and manipulation of multiple tags in the response store, and various modes of read and write operations. Such additional instructions can be accomodated with virtually no change in the conceptual framework of the proposed designs.

Further research can be carried out in many directions. First, the analyses offered in this paper for the linear-array structure need to be refined and also extended to other interconnection schemes. Second, numerous "optimistic" results on the complexity of associative memory algorithms must be reexamined in light of the new model, leading to fairer and more realistic results. Third, the simple search and retrieval operations considered in this paper can be augmented to include more sophisticated data manipulation and retrieval primitives.

Additionally, the simple linear intercommunication capability of the AM cells (provided through the SHIFT instruction) can be extended in several different ways.

It would be interesting to observe the effects of such extensions on the various organizational tradeoffs. Finally, the problem of optimal instruction sequencing in the context of various applications needs to be studied in order to assure efficient utilization of the proposed systolic architectures.

REFERENCES

[ATAL85] Atallah, M.J. and S.R. Kosaraju, "A Generalized Dictionary Machine for VLSI," *IEEE Transactions on Computers*, Vol. C-34, No. 2, pp. 151-155, Feb. 1985.

[BENT79] Bentley, J. and H. Kung, "A Tree Machine for Searching Problems," *Proc. of the Int'l Conf. on Parallel Processing*, 1979, pp. 265-266.

[BILA82] Bilardi, G., M. Pracchi, and F.P. Preparata, "A Critique of Network Speed in VLSI Models of Computation," *IEEE Journal of Solid-State Circuits*, Vol. SC-17, pp. 696-702, Aug. 1982.

[BROW79] Browning, S.A., "Computations on a Tree of Processors," *Proc. of the Caltech Conf. on VLSI*, Jan. 1979, pp. 453-478.

[CARE84] Carey, M.J. and C.D. Thompson, "An Efficient Implementation of Search Trees on $\lceil lg\ N + 1 \rceil$ Processors," *IEEE Transactions on Computers*, Vol. C-33, No. 11, pp. 1038-1041, Nov. 1984.

[CHAN88] Chang, J.H., O.H. Ibarra, M.J. Chung, and K.K. Rao, "Systolic Tree Implementation of Data Structures," *IEEE Transactions on Computers*, Vol. 37, No. 6, pp. 727-735, June 1988.

[CHAZ85] Chazelle, B. and L. Monier, "A Model of Computation for VLSI and Related Complexity Results," *Journal of the ACM*, Vol. 32, No. 3, pp. 573-588, July 1985.

[CHIS89] Chisvin, L. and R.J. Duckworth, "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *Computer*, Vol. 22, No. 7, pp. 51-64, July 1989.

[DEHN87] Dehne, F. and N. Santoro, "Optimal VLSI Dictionary Machines on Meshes," *Proc. of the International Conf. on Parallel Processing*, University Park, PA, Aug. 1987, pp. 832-840.

[DEHN88] Dehne, F. and N. Santoro, "An Optimal VLSI Dictionary Machine for Hypercube Architectures," School of Computer Science, Carleton Univ., Ottawa, Canada, Report SCS-TR-135, Apr. 1988.

[FISH84] Fisher, A.L., "Dictionary Machines with a Small Number of Processors," *Proc. of the International Symp. on Computer Architecture*, pp. 151-156, 1984.

[FOST76] Foster, C.C., *Content Addressable Parallel Processors*, Van Nostrand Reinhold, 1976.

[GOYA88] Goyal, P. and T.S. Narayanan, "Dictionary Machine with Improved Performance," *The Computer Journal*, Vol. 31, No. 6, pp. 490-495, Dec. 1988.

[GUIB82] Guibas, L.J. and F.M. Liang, "Systolic Stacks, Queues, and Counters," *Proc. of the MIT Conf. on Advanced Research in VLSI*, 1982, pp. 155-164.

[HANL66] Hanlon, A.G., "Content-Addressable and Associative Memory Systems: A Survey," *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 4, pp. 509-521, Aug. 1966.

[HURS89] Hurson, A.R., L.L. Miller, and S.H. Pakzad (Editors), *Tutorial: Parallel Architectures for Database Systems*, IEEE Computer Society Press, 1989.

[KOHO87] Kohonen, T., *Content-Addressable Memories*, Springer-Verlag, Berlin, 2nd Edition, 1987.

[KUNG80] Kung, H.T. and C.E. Leiserson, "Algorithms for VLSI Processor Arrays," in *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[KUNG80a] Kung, H.T. and P.L. Lehman, "Systolic (VLSI) Arrays for Relational Database Operations," *Proc. of ACM-SIGMOD International Conf. on Mangement of Data*, May 1980, pp. 105-116.

[KUNG82] Kung, H.T., "Why Systolic Architectures?" *Computer*, Vol. 15, No. 1, pp. 37-46, Jan. 1982.

[LEED88] Lee, D.-L. and W.A. Davis, "An $O(n + k)$ Algorithm for Ordered Retrieval from an Associative Memory," *IEEE Transactions on Computers*, Vol. 37, No. 3, pp. 368-371, Mar. 1988.

[OMON87] Omondi, A.R. and J.D. Brock, "Implementing a Dictionary on Hypercube Machines," *Proc. of the International Conf. on Parallel Processing*, University Park, PA, Aug. 1987, pp. 707-709.

[OTTM82] Ottman, T.A., A.L. Rosenberg, and L.J. Stockmeyer, "A Dictionary Machine for VLSI," *IEEE Transactions on Computers*, Vol. C-31, No. 9, pp. 892-897, Sep. 1982.

[PARH73] Parhami, B., "Associative Memories and Processors: An Overview and Selected Bibliography," *Proceedings of the IEEE*, Vol. 61, No. 6, pp. 722-730, June 1973.

[PATE81] Paterson, M.S., W.L. Ruzzo, and L. Snyder, "Bounds on Minimax Edge Length for Complete Binary Trees," *Proc. of the ACM Symp. on the Theory of Computing*, May 1981, pp. 293-299.

[PREP81] Preparata, F.P. and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of the ACM*, Vol. 24, No. 5, pp. 300-309, May 1981.

[PROB87] Probst, D.K. and H.F. Li, "Optimal Systolic Dictionary Machines on Mesh-Type Architectures," *Proc. of the Canadian Conf. on VLSI*, Winnipeg, Oct. 1987, pp. 247-252.

[SCHM85] Schmeck, H. and H. Schroder, "Dictionary Machines for Different Models of VLSI," *IEEE Transactions on Computers*, Vol. C-34, pp. 472-475, 1985.

[SCHW87] Schwartz, A.M. and M.C. Loui, "Dictionary Machines on Cube-Class Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 1, pp. 100-105, Jan. 1987.

[SIEG85] Siegel, H.J., *Interconnection Networks for Large-Scale Parallel Processing*, Lexington Books, 1985.

[SOMA85] Somani, A.K. and V.K. Agarwal, "An Efficient Unsorted VLSI Dictionary Machine," *IEEE Transactions on Computers*, Vol. C-34, No. 9, pp. 841-852, Sep. 1985.

[SONG80] Song, S.W., "A Highly Concurrent Tree Machine for Database Applications," *Proc. of the International Conf. on Parallel Processing*, Aug. 1980, pp. 259-268.

[SUSY88] Su, S.Y.W., *Database Computers: Principles, Architectures, and Techniques*, McGraw-Hill, 1988.

[YAUS77] Yau, S.S. and H.S. Fung, "Associative Processor Architecture -- A Survey," *Computing Surveys*, Vol. 9, No. 1, Mar. 1977.