Search and Data Selection Algorithms for Associative Processors

Behrooz Parhami
University of California
Department of Electrical and Computer Engineering
Santa Barbara, CA
parhami@ece.ucsb.edu

Abstract

Associative or content-addressable memories (AMs, CAMs) have been studied and used 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. Also, more functional variants of such systems known as associative or content-addressable processors (APs, CAPs) have been the subjects of extensive research. Since AMs are essentially simple APs, we will use the term "AP" to refer to both AMs and APs. Many algorithms have been developed for performing search, retrieval, and arithmetic/logic operations on data stored in APs. In this paper, search and data selection algorithms for both fully parallel and bit-serial APs are reviewed and analyzed. Search and selection functions covered include exact-match, inexact-match (numerical or logical proximity), membership (multiple match), relational (<, ≤, >, ≥), interval (between limits), extrema (max, min, next higher, next lower), rank-based selection (kth or k largest/smallest), and ordered retrieval (sorting). Hardware features that facilitate or speed up these algorithms are also described. In addition to several algorithms being presented for the first time, discussions touch on two topics that have been inadequately covered in previous treatments: (1) Worst-case and average-case complexity analyses and (2) effects of hardware features and implementation details on the performance of search and data selection algorithms.

1 Introduction

Associative or content-addressable memories (AMs, CAMs) have been studied 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, since the advent of electronic digital computers. Also, more functional variants of such systems known as associative or content-addressable processors (APs, CAPs) have been the subjects of extensive research and development. Because AMs are essentially simple APs, we will use the term "AP" to refer to both AMs and APs.

Table 1 summarizes the events and advances of the past half century leading to modern APs of the 1990s. It also points to some key references for further study. The term "bit-ops" in the performance column stands for "bit operations per second."

<table>
<thead>
<tr>
<th>Decade</th>
<th>Events and Advances</th>
<th>Technology</th>
<th>Performance</th>
<th>Key References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1940s</td>
<td>Formulation of need and concept</td>
<td>Relays</td>
<td></td>
<td>[3], [37]</td>
</tr>
<tr>
<td>1950s</td>
<td>Emergence of cell technologies</td>
<td>Magnetic, Cryogenic</td>
<td>Megabit-ops</td>
<td>[33]</td>
</tr>
<tr>
<td>1960s</td>
<td>Introduction of basic architectures</td>
<td>Transistors</td>
<td></td>
<td>[10], [12]</td>
</tr>
<tr>
<td>1970s</td>
<td>Commercialization and applications</td>
<td>ICs</td>
<td>Gigabit-ops</td>
<td>[22], [36]</td>
</tr>
<tr>
<td>1980s</td>
<td>Focus on system/software issues</td>
<td>VLSI</td>
<td>Terabit-ops</td>
<td>[4], [17]</td>
</tr>
<tr>
<td>1990s</td>
<td>Scalable and flexible architectures</td>
<td>ULSI, WSI</td>
<td>Petabit-ops?</td>
<td>[11], [19]</td>
</tr>
</tbody>
</table>
Many algorithms have been developed for performing search, retrieval, and arithmetic/logic operations on data stored in APs. Such operations can be programmed using the basic masked exact-match search capability of simple AMs, but the provision of other types of hardware aids can have a significant effect on performance. Starting with the pioneering works of Falkoff [7] and Estrin and Fuller [6], research in AP algorithms and associated hardware speedup techniques has continued to the present [5], [8], [15], [16], [20], [32].

Even though bit-serial arithmetic operations can be programmed on virtually all APs, and special hardware features have been implemented or suggested in AP systems for facilitating or speeding up numerical computations, the primary focus of AP implementation efforts and research studies has been searching and other nonnumeric functions. In addition to simple exact-match search, many other search types have been implemented as primitives for data manipulation and retrieval functions. The basic search types are defined below:

- **Exact-match search**—locating data based on partial knowledge of contents.
- **Inexact-match searches**—finding numerically or logically proximate values.
- **Membership searches**—identifying all members of a particular set.
- **Relational searches**—determining values that are less than, less or equal, etc.
- **Interval searches**—marking items that are between limits or not between limits.
- **Extrema searches**—max- or min-finding, next higher, next lower.
- **Rank-based selection**—selecting kth or k largest/smallest elements.
- **Ordered retrieval**—repeated max- or min-finding with elimination (sorting).

In this paper, we illustrate hardware/software algorithms for such search functions along with the rich collection of techniques used for implementing them efficiently on both fully parallel and bit-serial AP architectures. More important, we show that despite the simplicity and intuitiveness of some of the algorithms, sophisticated optimization and analysis techniques are needed to achieve the best possible speedups with given hardware capabilities.

## 2 Background and assumptions

In its simplest form, an AP can be viewed as a hardware device consisting of N fixed-size cells, each being marked as empty or storing a W-bit data word or record (see Figure 1). The number of nonempty AP cells is denoted by n (n ≤ N). Other notation is defined.

![Functional view of an associative processor.](image-url)
in subsection 2.2 and Table 2 following some needed background on AP architectures.

2.1 Architectural variations

Architecturally, associative processors come in four basic varieties depending on how they handle the processing of multiple words (records) and multiple bits in a word. These varieties, described in the following paragraphs are known as fully parallel, bit-serial, word-serial, and block-oriented [22].

*Fully parallel* (word-parallel, bit-parallel) APs have comparison logic associated with each bit of stored data. In simple exact-match searches, the logic associated with each bit generates a local match or mismatch signal. These local signals are then combined to produce the cell match or mismatch result. In more complicated searches, the bit logic typically receives partial search results from a neighboring bit position and generates partial results to be passed on to the next bit position. Although lookahead techniques can speed up the propagation of partial search results, they are seldom used in view of their high cost and marginal performance impact.

*Bit-serial* (word-parallel, bit-serial) systems process an entire bit slice of data, containing one bit of every word, simultaneously, but go through multiple bits of the search field sequentially. Bit-serial systems have been dominant in practice since they allow the most cost-effective implementations using low-cost, high-density, off-the-shelf RAM chips. However, fully parallel systems have also been implemented, particularly for applications that require high performance in the basic masked exact-match search capability.

*Word-serial* (word-serial, bit parallel) APs based on electronic circulating memories constitute hardware implementation of a programmed linear search. Even though several such systems were built in the 1960s, they are no longer cost-effective with today’s technology.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition or Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>Prime: Denotes logical complement (for example, $c'$ is the logical complement of $c$)</td>
</tr>
<tr>
<td>$b_{ij}$</td>
<td>Strictly between-limits signal for cell $i$ up to and including $d_{ij}$</td>
</tr>
<tr>
<td>$c_j, C_j$</td>
<td>Comparand's $j$th bit</td>
</tr>
<tr>
<td>$d_{ij}$</td>
<td>Bit $j$ of the data word in cell $i$</td>
</tr>
<tr>
<td>$e_{ij}, E_{ij}$</td>
<td>Equal (exact-match) signal for cell $i$ up to and including $d_{ij}$</td>
</tr>
<tr>
<td>$g_{ij}, G_{ij}$</td>
<td>Strictly greater-than signal for cell $i$ up to and including $d_{ij}$</td>
</tr>
<tr>
<td>$g_{ij}^{(m)}$</td>
<td>Minimally greater-than signal for cell $i$ up to and including $d_{ij}$</td>
</tr>
<tr>
<td>$h_j$</td>
<td>Hamming distance indicator for word $j$</td>
</tr>
<tr>
<td>$k$</td>
<td>Rank selection parameter (for example, $k$th largest or $k$ largest)</td>
</tr>
<tr>
<td>$l_{ij}, L_{ij}$</td>
<td>Strictly less-than signal for cell $i$ up to and including $d_{ij}$</td>
</tr>
<tr>
<td>$m_j$</td>
<td>Mask word’s $j$th bit; $m_j = 0$ means that bit-slice $j$ is not involved in the search</td>
</tr>
<tr>
<td>$N, n$</td>
<td>AP size in words and number of nonempty or active words (problem size)</td>
</tr>
<tr>
<td>$S_{wrx}$</td>
<td>Expected time for selecting the $r$th largest among $x$ candidates in a $w$-bit field</td>
</tr>
<tr>
<td>$T_{wx}$</td>
<td>Expected search time for bit-serial max-finding with $x$ candidates in a $w$-bit field</td>
</tr>
<tr>
<td>$T_{wx}^{(γ)}$</td>
<td>Same as $T_{wx}$ except that the bit-serial search is preceded by a search for the all-ones pattern in a subfield of length $γ$</td>
</tr>
<tr>
<td>$t_i$</td>
<td>Tag or response bit for cell $i$</td>
</tr>
<tr>
<td>$v$</td>
<td>The numerical value of the comparand $c$ as masked by $m$</td>
</tr>
<tr>
<td>$W, w$</td>
<td>Width of AP word/record and width of a (search) field, respectively</td>
</tr>
<tr>
<td>$x, y$</td>
<td>Number of candidate cells in various stages of a search</td>
</tr>
<tr>
<td>$γ$</td>
<td>Group length for multiple-bit parallel searches in max-finding algorithms</td>
</tr>
<tr>
<td>$ε$</td>
<td>Comparison error threshold for numerical proximity search</td>
</tr>
<tr>
<td>$φ$</td>
<td>Masked bit position</td>
</tr>
</tbody>
</table>
Finally, block-oriented (block-parallel, word-serial, bit/byte-serial) systems represent a compromise between bit-serial and word-serial systems in an effort to make large systems practically realizable. Some block-oriented AP systems are based on augmenting the read/write logic associated with each head of a head-per-track disk so that it can search the track contents as they pass underneath. Such a mechanism can act as a filter between the database and a fast sequential computer or as a special-purpose database search engine.

Several instances of these four architecture classes are described in [36]. These four basic architectures, along with intermediate or hybrid schemes, provide AP designers with a vast design space and speed-cost trade-offs. Examples of the available cost-performance trade-offs in the design of a class of VLSI-based search processors can be found in [25].

2.2 Basics of AP search and arithmetic

Data stored in an AP cell may be numeric, nonnumeric (bit string), or a combination of both. Numeric values will be assumed to be unsigned integers or fixed-point values in much of our discussion. Modifications required for search and data selection with signed fixed-point and floating-point numbers will be briefly discussed in section 7.

When presented with a comparand (search key) c, a mask m specifying the relevant field(s) of the stored words, and possibly an instruction containing the type of search, a fully parallel AP responds by marking all the words that satisfy the search criteria. Marking is done by setting or resetting a response bit or tag (bit) t associated with the AP cell. A bit-serial AP performs its comparison by reading out an unmasked bit slice into a bit-slice register, comparing these bits to the corresponding bit of the comparand, and setting the tags accordingly. In addition, both fully parallel and bit-serial APs are usually capable of reading out bit slices into bit-slice registers and performing logical operations between these bit-slice registers. When bit-slice registers are used to indicate membership of cells in various subsets of the N AP cells, logical operations can be used to compute set union, intersection, and difference in a simple way.

The N response bits together form the AP's response store. A response indicator mechanism may provide information on the multiplicity of responders (zero, one, several) or an exact count of the number of responders (see subsection 2.3). The response indicator, multiple-response resolver, and several bit-slice registers are parts of the global tag operations unit shown in Figure 1. Other terminology and notation will be introduced as needed. Table 2 contains a complete list of notation and symbols for reference.

Despite our focus on search algorithms, we need to understand how addition can be performed on multiple AP words for certain numerical search algorithms. Subtraction is the same as addition with the compare negated. Bit-serial addition is straightforward and consists of repeated manipulation of bit slices, keeping the carries in a bit-slice register. If each AP cell is provided with a ripple-carry adder, then parallel addition is possible, provided that the carries are generated as the logical AND of actual carries and the mask bits (that is, carry should not propagate into or out of masked positions—those with \( m_j = 0 \)). With parallel addition, the sum (difference) overwrites the current field contents, but since one operand is common to all operations, the original values can be restored by a final subtraction (addition) if desired.

2.3 Dealing with multiple responders

Each AP search operation yields a set of responding cells or candidates. When the set of responders is empty, the search has failed (not necessarily bad, as for example in searching for potentially hazardous situations) and the algorithm must follow an alternate branch. A unique responder usually means that the required data item has been identified and can be dealt with by reading it out or modifying it in place. With multiple responders, the appropriate course of action might be proceeding to further narrow down the set of candidates, simultaneously modifying all responders in place, or reading the responders out sequentially and dealing with each one individually. Hence, information might be needed both on the multiplicity of responders and on their physical locations in the AP array. Both of these are provided by the global tag operations unit depicted in Figure 1.

Response multiplicity indicators range from a simple binary "none/one" flag, obtained as the logical OR of all tag bits, at one extreme, to an exact count of responders, obtained by hardware in the form of parallel counter circuits [34], at the other. Intermediate solutions include the provision of a ternary "none/one/several" response indicator and counting or sorting the tags by way of high-speed shifting.

Response location information can be provided by a multiple-response resolver circuit [9]. The simplest (and slowest) implementation of a multiple-response resolver consists of cascaded OR gates producing an "inhibit" signal for cell \( i \) based on the logical OR of the tags for cells 0 through \( i - 1 \). This is essentially a priority circuit. Faster priority circuits have tree-structured designs. Many such designs are immedi-
ately derivable based on a parallel-prefix formulation [21] of the response resolution problem. It is also possible to obtain the address of the first responding cell through an N-input priority encoder.

3 Simple search algorithms

We cover the basic exact-match search in subsection 3.1. Inexact or approximate match can be defined in many different ways corresponding to various application-dependent notions of proximity. Our discussion will be limited to one-dimensional searches. Multidimensional searching, using Euclidean distance or other measures of proximity in higher-dimensional space, can be programmed by first computing the distances and then using the techniques discussed here. Two measures of proximity will be considered in subsections 3.2 and 3.3: Numerical proximity (approximate equality of numerical values with a comparison threshold ε) and logical proximity (closeness of bit strings measured by their Hamming distance).

3.1 Exact match search

Early APs were specifically designed to implement exact-match searches and very little else. As the name implies, in exact-match search, a data word must completely match the comparand in all unmasked bit positions in order to qualify as a responder.

In fully parallel architectures, the jth bit position within the ith cell produces the local match signal \( m_j \) with the cell then setting \( t_i \) to the logical AND of the match signals by implementing the recurrence

\[
e_{ij} = e_i, \; j \pm 1 \left( m_j + (c_j \oplus d_{ij}) \right), \text{ initially 1}
\]  

where the plus or minus sign in the \( j \pm 1 \) index represents right-to-left or left-to-right signal propagation and combining, respectively. The tag \( t_i \) is then set to \( e_{i0} \) or \( e_{iw} \). It is also possible to find local mismatch signals \( m_j(c_j \oplus d_{ij}) \) and OR them to derive the cell mismatch result \( t_i' \).

The use of wired logic can speed up the formation of the W-input AND (OR) function for reasonable values of the word width W. A tree-structured combining circuit can also be used, but such a circuit adds complexity to each cell and makes the cell structure less regular. Any such parallel combining scheme is represented by the logical expressions:

\[
t_i = \prod_{j=0}^{W-1} (m_j + (c_j \oplus d_{ij}))
\]

\[
= \sum_{j=0}^{W-1} m_j(c_j \oplus d_{ij})
\]

In the special case where \( w = W \) (that is, the search field is as wide as an entire word), a dictionary lookup is performed whereby the presence or absence of a given word in the AP cells is determined. Design of VLSI-based dictionary machines, a subclass of APs, has received much attention since the early 1980s. For an overview and extensive references, see [23].

In bit-serial designs, exact-match search is implemented by sequentially reading out the unmasked bit slices, comparing each bit slice to the desired bit value, and keeping the logical AND of all bit-slice match indicators. Alternatively, mismatch signals may be produced for individual bit positions and ORed together to produce the word mismatch indicator. Either way, the scheme can implement both equality and inequality searches.

The worst-case and expected bit-serial exact-match search times are both \( O(w) \) bit-slice search steps, where \( w \) is the width of the unmasked field(s). Note that we do not claim an \( O(w) \) search time, as is commonly done in the literature, because the time needed for a bit-slice search increases with the AP size. Hence, a larger AP will take longer to carry out the same w-bit exact-match search (see subsection 7.3 on scalable APs).

3.2 Numerical proximity search

Numerical proximity searches are almost invariably programmed using a combination of other searches, since a direct implementation is not likely to be cost-effective. Let \( v \) be the value of the comparand as masked by \( m \). The objective is to mark words that have values in the interval \([v - \epsilon, v + \epsilon]\) in the corresponding field. This can be done through an interval search algorithm (see Subsection 4.3) or by subtraction, selective negation of negative differences (finding absolute values), and a relational “less-than” search (see subsection 4.1).

In either case, the search can be restricted in the intervals \([v, v + \epsilon]\) or \([v - \epsilon, v]\) to perform higher-within-threshold or lower-within-threshold search. When such one-sided numerical proximity searches are followed by min- and max-finding algorithms, respectively, the result is a next-higher-within-threshold or next-lower-within-threshold search. However,
transposing the order of the two operations—namely, finding the next higher or next lower (see subsection 5.2) and then checking if the difference is within the specified threshold—is more efficient since the proximity check can then be performed outside the AP system at higher speed.

3.3 Logical proximity search

The Hamming distance of two bit strings of equal length is defined as the number of bit positions in which they differ. If a fully parallel AP is implemented with only exact-match search capability, then determining all words that are a Hamming distance of 1 or less away from the masked comparand requires \( w + 1 \) searches. This technique quickly becomes impractical for higher Hamming distances. For example, to find all words that are Hamming distance of 2 or less away would require \( 1 + w(w + 1)/2 \) searches.

A fully parallel AP can be provided with a hardware mechanism to compute Hamming distances up to a maximum of \( h \). For this, an \((h + 2)\)-valued state must be passed between adjacent bit positions in a cell. This state would indicate partial Hamming distance from the left or the right end of the cell. The \( h + 2 \) states represent distances of 0, 1, 2, …, \( h \) and >\( h \). A masked-out or matching bit position would simply pass on the state as received from left (right) while a mismatch would update the state to the next higher one. For small values of \( h \), clever encoding of the state can be found to minimize the logic complexity.

In a bit-serial AP, the state of the search for each cell must be stored in memory or in bit-slice registers within the processing elements. Feng [8] provides an algorithm and associated encoding for \( h = 1 \).

Once the Hamming distances are found, identifying all the words with distances of \( h \) or less is straightforward, particularly if the encoding was selected with this step in mind. One may also wish to follow this algorithm with a min-finding algorithm (see section 5) in order to identify the words that are closest to the comparand.

3.4 Membership search

A membership search identifies the words that match any member of a given set of comparands. One way to perform a membership search is to simply do a sequence of searches with each member of the set used as the comparand in turn. This way, both exact- and inexact-match searches can be handled but execution time may become unacceptable for large sets. In the following paragraphs, algorithmic speedup techniques for the case of exact-match membership search are discussed.

Some membership searches can be converted to other searches. For example, to search for \( 0101, 0111, 1101, \) or \( 1111 \) in a 4-bit numerical field, one can perform exact-match search with the bit pattern \( \phi \), where \( \phi \) represents a masked bit. Searching for \( 1101, 1110, \) or \( 1111 \) is equivalent to a \( >1100 \) or \( \geq 1110 \) search (see section 4). In more general cases, special algorithms are needed as described below.

In a fully parallel AP, the problem of minimizing the number of search cycles for membership search can be solved in a way similar to two-level AND-OR logic circuit minimization: One needs to identify a minimal number of masked comparands or “prime implicants” that “cover” all input patterns. For example, to search for \( 0101, 0110, 0111, 1101, 1110, \) or \( 1111 \) in a 4-bit numerical field, two searches for the four-cubes \( \phi \) and \( \phi \) will do.

The problem is even more interesting for bit-serial APs. Let us first look at the case of a set of size 2 [8], which we call “double-search” here. If the two words in the set are Hamming distance \( d \) apart, then \( w + d \) bit-slice searches will identify the required words: \( w - d \) searches to check for the \( w - d \) bits that are equal and \( 2d \) searches to check for the complementary \( d \) bits. By taking one of the differing bit positions as a reference point and then comparing the other bits against this position using exclusive-OR operation between bit slices (XOR of any two bit slices among the complementary \( d \) bits must be the same), the number of bit-slice operations can be reduced to \( w + d - 3 \), which in the worst case of \( d = w \) becomes \( 2w - 3 \).

Feng [8] also suggests that for larger sets, the above procedure be applied to a number of pairs covering the entire set and having a minimal sum of pairwise Hamming distances. As an example, to search for members of the set \( \{0101, 0110, 1001\} \), with pairwise Hamming distances of 2, 2, and 4, the two double-searches \( \{0101, 0110\} \) and \( \{0101, 1001\} \) will be conducted as above, leading to 7 bit-slice operations, including the final logical OR to combine the results of the two double-searches. This is about twice as fast as searching for each pattern separately and then ORing the search results.

4 Relational and interval searches

There are four relational searches: \( <, \leq, >, \geq \). We consider the \( < \) and \( \leq \) algorithms for fully parallel APs in detail, discussing briefly how the other two relational searches and bit-serial versions of the algorithms can be derived.
4.1 Less-than search

The fully parallel "less-than" search can be implemented based on signaling from left to right or from right to left. In the left-to-right version (msb to lsb), one needs a pair of logical variables per cell to keep track of numbers that are less than the comparand thus far ($l_{ij}$) and those that are equal to $c$ thus far ($e_{ij}$). Updating rules and initial values for these variables are as follows:

$$ e_{ij} = e_{i, j+1} (m'_j + (c'_j \oplus d_{ij})), \text{ initially } 1 \quad (3) $$

$$ l_{ij} = l_{i, j+1} + m_j e_{i, j+1} c'_j d'_{ij}, \text{ initially } 0 \quad (4) $$

At the end, $l_{i0}$ indicates if the corresponding word $i$ is less than the comparand.

Note that masked bit positions, corresponding to $m_j = 0$, do not affect the two variables and are effectively skipped. Thus, bit-serial versions of this and other algorithms in this section can be obtained by simply setting $m_j$ to 1 in all equations and restricting the operations to unmasked bit-slices.

The right-to-left version of the "less-than" search is simpler in that only a single logical variable $l_{ij}$ needs to be propagated as follows:

$$ l_{ij} = m_j c_j d'_{ij} + l_{i, j-1} (m'_j + c_j + d'_{ij}), \text{ initially } 0 \quad (5) $$

In the bit-serial implementation of this algorithm, we have the added advantage of being able to skip over trailing 0s in the comparand, since such trailing 0s will force $l_{ij}$ to remain at its initial value of 0. For example, if the unmasked part of the comparand is 11001000, the search will skip the right-most three bit positions, leading to a reduction of 3 in the number of bit-slice searches and also in the associated logical operations of (5). Since the expected number of trailing 0s in a random string is about 1 (it is exactly 1 for an infinitely long string), the amount of savings is insignificant on the average.

4.2 Less-or-equal search

The left-to-right version of the "less-or-equal" ("not-greater-than") search can be performed by using (3) and (4) but setting the tag to $l_{i0} + e_{i0}$ instead of $l_{i0}$.

Alternatively, the following recurrence equation for $g_{ij}$ ("not greater than" in cell $i$ up to bit position $j$) can be used along with (3) or its complementary version written in terms of $e'_{ij}$:

$$ g'_{ij} = g'_{i, j+1} (m'_j + e'_{i, j+1} + c_j + d'_{ij}), \text{ initially } 1 \quad (6) $$

Again, the right-to-left version is simpler and is defined by the single recurrence:

$$ g_{ij} = m_j c_j d'_{ij} + g_{i, j-1} (m'_j + c_j + d'_{ij}), \text{ initially } 1 \quad (7) $$

In the bit-serial implementation of this algorithm, trailing 1s in the comparand can be skipped in the same way as the skipping of trailing 0s discussed in connection with (5).

With more complex hardware in each cell of a fully parallel AP, the linear recurrences derived above can be computed by tree-structured parallel prefix circuits in much the same way as the carry-out signal of a carry-lookahead adder.

Clearly, when a string of $\gamma$ consecutive 1s or 0s is encountered in the comparand, a single $\gamma$-bit search can be used to refine the candidate set or to identify the subset of the candidates that must be marked as "greater" in a fully parallel implementation. The speedup obtained here is modest as the expected length of strings of consecutive 1s or 0s in a random binary sequence is no more than $1/2 + 2/4 + 3/8 + 4/16 + \ldots + n/2^n + \ldots = 2$.

4.3 Interval searches

An interval (between-limits or not-between-limits) search can be implemented as a pair of relational searches as discussed in the previous subsections. However, there are opportunities for optimization in hardware or search speed if both relational searches are considered together rather than as two independent searches.

Consider searching in the closed interval [$c$, $C$], where a responder $d_i$ must satisfy $c \leq d_i \leq C$. Since we are assuming integer or fixed-point values, searches with open ($c < d_i < C$) or half-open ($c \leq d_i < C$, $c < d_i \leq C$) intervals can be easily converted to closed-interval searches by adjusting the endpoint(s) involved or can be handled by simple modifications to the following algorithms.

The left-to-right (msb to lsb) search algorithm needs two logical variables to keep track of equality with $c$ and $C$ ($e_{ij}$ and $E_{ij}$) and one ($b_{ij}$) to indicate if $d_i$
is strictly between \( c \) and \( C \) up to this point. Recurrence (3) can be used for \( e_{ij} \), and by changing \( e \) to \( E \) and \( c \) to \( C \), for \( E_{ij} \). The third recurrence is:

\[
    b_{ij} = b_{i, j+1} + e_{i, j} + m \cdot j \cdot c_j \cdot d_{ij} + E_{i, j} + m \cdot j \cdot C_j \cdot d_{ij},
\]

initially 0 \hspace{2cm} (8)

In the bit-serial implementation of this procedure, an exact-match search can replace the above steps until the first occurrence of \( c_j - C_j \). This saves a great deal of time in some of the cases where the interval of interest is relatively small. For example, to search for stored values between \( 90 = (1011010)_2 \) and \( 92 = (10110100)_2 \), inclusive, one first identifies all cells holding \( 1011000 \) in the desired field and then uses the above algorithm to determine if the initially masked-out \( \phi \phi \phi \) part is between \( (010)_2 \) and \( (100)_2 \). The same optimization can be applied to fully parallel AP algorithms that are not based on built-in interval search hardware in order to reduce the search time.

The right-to-left (lsb to msb) version of the interval search algorithm is again simpler since it is based on only two recurrences for the two searches \( d_i \geq c \) and \( d_i \leq C \) instead of three for \( e_{ij} \), \( E_{ij} \), and \( b_{ij} \):

\[
l'_{ij} = m \cdot j \cdot c_j \cdot d_{ij} + l'_{i, j-1} (m \cdot j + c_j + d_{ij}),
\]

initially 1 \hspace{2cm} (9)

\[
g'_{ij} = m \cdot j \cdot C_j \cdot d_{ij} + g'_{i, j-1} (m \cdot j + C_j + d_{ij}),
\]

initially 1 \hspace{2cm} (10)

The final result is \( t_j = l'_{ij} \cdot g'_{ij} \). On the negative side, this algorithm does not lend itself to any optimization except ignoring or skipping of a number of right-most bit positions that simultaneously satisfy \( c_j = 0 \) and \( C_j = 1 \).

5 Extrema searches

There are four extrema searches: max, min, next-higher (minimum among greater values), and next-lower (maximum among lesser values). We will discuss only max-finding and next-higher search algorithms here as the others are similar. The analyses and optimizations presented in this section are original [29], [31].

5.1 Max-finding search

The max-finding algorithm to be described next is essentially bit-sequential even when executed on a fully parallel AP with special max-finding hardware features such as those in [16]. Initially, at the msb position, all cells are candidates for holding the maximum value. At the start of a typical bit-slice step, a search is performed for the value of 1 in bit position \( j \). Depending on the multiplicity of responders, the candidate set is modified as follows:

None: Candidate set does not change.
None: Stop; maximum already identified.
Several: Candidate set is replaced by the set of responders.

At most, \( w \) bit-slice searches are required in the worst case. One can find the expected number of bit-slice searches based on the assumption of randomness of the stored bit patterns. Let \( T_{wx} \) denote the expected number of bit-slice searches for a field of width \( w \) when there are \( x \) candidate cells initially. Then:

\[
    T_{wx} = 1 + 2^{-x} T_{w-1, x} + 2^{-x} \sum_{y=2}^{x} \left( \begin{array}{c} x \\ y \end{array} \right) T_{w-1, y},
\]

with \( T_{0x} = T_{wx} = 0 \) \hspace{2cm} (11)

To justify the above recurrence, let \( y \) be the number of responders after searching for 1 in a particular bit slice and thus spending one search cycle. If \( y = 0 \) (no responder, probability of event \( = 2^{-x} \)), the search continues with the same number \( x \) of candidates in the remaining \( w - 1 \) bit positions. For \( y = 1 \), no further search is necessary. Finally, if there are \( y \geq 2 \) responders, an event having the probability \( \left( \frac{x}{y} \right) 2^{-x} \) for each value of \( y \), the search continues with \( y \) candidates in the remaining \( w - 1 \) bit positions. Figure 2 shows the variation of \( T_{wx} \) with the candidate set size \( x \) for several values of the field width \( w \).

In a fully parallel AP, one can use the available multiple-bit parallel search capability to advantage for reducing the expected max-finding search time. Detailed analyses and algorithms are beyond the scope of this paper, but a simple example and a few charts can convey the extent of savings that can be achieved [29], [31].

Intuitively, when the number of candidates \( x \) is much larger than \( 2^w \), it is very likely that the maxi-
maximum value is $2^w - 1$, represented by the all-ones bit pattern. Thus, in such a case, it makes sense to first search for the all-ones pattern and resort to a bit-serial max-finding algorithm only if the first search yields no match. Each word contains the all-ones pattern with probability $2^{-w}$. Therefore, the probability that none of the given $x$ words contains the all-ones pattern is $(1 - 2^{-w})^x$. Denoting the number of search steps with this strategy as $T_{wx}^{(w)}$, where the superscript designates the group length in the initial search, we have

$$
T_{wx}^{(w)} = 1 + (1 - 2^{-w})^x T_{wx}
$$

where $T_{wx}$ is the bit-serial search complexity given by (11). Plots of the value of $T_{wx}^{(w)}$ as a function of $x$ for different values of $w$ are given in Figure 3. Comparing Figures 2 and 3 reveals the advantages of this new strategy when $x$ is larger than $2^w$. For example, given $x = 1,000$ and $w = 8$, the initial search for the all-ones pattern reduces the expected search time from 8 (see Figure 2) to about 1.15 (see Figure 3), implying an average speedup of about 7.

Instead of searching the entire $w$-bit field for the all-ones pattern at once, one can proceed in smaller $\gamma$-bit groups. In other words, the $w$-bit search is divided into $\lceil w/\gamma \rceil$ phases, each involving a search for the $\gamma$-bit all-ones pattern followed by $\gamma$ single-bit searches if the former is unsuccessful. The resulting search complexity $T_{wx}^{(\gamma)}$ is plotted in Figure 4 for $w = 12$. Figure 4 clearly shows how the optimal group size increases, from 2 to 3 to 4 for the data points shown, as the size $x$ of the candidate set grows.

Finally, instead of using fixed-size groups, the length of the search subfield can be dynamically adjusted with the size $x$ of the candidate set. Figure 4 also represents information on the optimal group size. Numerical experimentation has shown that the optimal length $\gamma^*_{opt}$ of the search subfield grows roughly as $\log_2 x$.

5.2 Next-higher search

To find the next higher value relative to a masked comparand $c$, one can follow a relational $>$ or $\geq$ search (depending on how “next higher” is defined) with a min-finding algorithm. However, direct implementation can be more efficient.

Consider the next-higher ($>$) search algorithm as a bit-serial search proceeding from left to right (msb to lsb). Fully parallel implementation of the next-higher search is infeasible in view of the requirement for counting the set of candidates in intermediate bit-
Figure 3. Expected number of cycles in a bit-serial max search with initial search for the all-ones pattern.

Figure 4. Expected number of cycles in max search in a field of width $w = 12$ with fixed group size $\gamma$. 

19
slices, as will become evident shortly. Before inspecting bit-slice \( j \), the present state of the search consists of two disjoint sets of candidates (maintained in two bit-slice registers).

Primary candidates: Equal to the comparand thus far \( e_{ij} \)

Secondary candidates: Minimally greater than the comparand thus far \( g_{ij}^{(m)} \).

The required “next-higher” element(s) will be among the secondary candidates only if all primary candidates turn out to be \( \leq c \) (the primary set becomes empty at some stage). If both sets are empty, then the search fails.

Updating of \( e_{ij} \) values is straightforward and can be done either bit-serially or in fully parallel fashion based on Equation (3). As for \( g_{ij}^{(m)} \), it is passed to the next lower bit position unmodified if \( c_j = 1 \) and is changed according to the following rule if \( c_j = 0 \):

\[
\text{if } d_{ij} = 1 \text{ for one or more primary candidates} \quad (13)
\text{then } \text{they become the set of secondary candidates}
\text{else if } d_{ij} = 0 \text{ for one or more secondary candidates} \quad (13)
\text{then } \text{they become the set of secondary candidates}
\text{else there is no change}
\text{endif}
\]

Since the determination of \( g_{i,j-1}^{(m)} \) from (13) requires two global responder operations, the algorithm is impractical for fully parallel implementation.

It is interesting to note that, in most references on associative processing, next-higher and next-lower searches are commonly grouped with inexact match searches. Here, we have chosen to regard them as extrema searches since the algorithms resemble those for max- and min-finding. More important, the next higher (lower) value need not be proximate to the comparand in the usual sense of the term. For example, the next higher value for the comparand 24 may turn out to be 106.

6 Rank-based selection and sorting

Rank-based selection is a generalization of max- and min-finding that lets us identify the \( k \)th largest (smallest) element in a set of \( n \) candidates. When \( k = \left\lfloor n/2 \right\rfloor \) or \( \left\lceil n/2 \right\rceil \), the median value is obtained, which is useful in many application areas. The algorithms presented in this section are original [31].

6.1 \( k \)-th largest search

Like max-finding, the determination of the \( k \)th largest element is bit-sequential in nature. To execute the following algorithm, the AP must be equipped with a multiple response counter capable of providing the exact number of responders.

There are \( w \) steps in the algorithm. Consider step \( i \) \((0 \leq i < w)\) that starts with \( x \) candidates among which the \( r \)th largest must be identified \((r \leq x; \text{ initially, } x = n, \ r = k)\). A search for 1 is conducted in the \( i \)th most significant bit of the field. Let there be \( y \) responders. If \( y \geq r \), then the search continues among the responders for the \( r \)th largest value starting at the next bit slice. If \( y < r \), then the required value must have a 0 in the current bit slice. The search continues among the non-responding subset of the original set of candidates for the \((r - y)\)th largest value, again starting at the next bit slice.

The average-case complexity of this algorithm can be analyzed in a manner similar to that of bit-sequential max-finding. The following recursive formulation of the expected number of steps for finding the \( r \)th largest among \( x \) values is directly derivable from the above description:

\[
S_{wx} = 1 + \sum_{y=r}^{x} \binom{x}{y} 2^{-x} S_{w-1}, \quad r, \ y = \sum_{y=r}^{r-\lfloor \log_2 x \rfloor} \binom{x}{y} 2^{-x} 
\]

Boundary conditions for Equation (14) are \( S_{w1} = S_{01} = 0 \).

Here is one way to speed up the algorithm on a fully parallel AP. Consider a \( \gamma \)-bit subfield starting at position \( i \) and let there be \( x \) candidates at this point. Approximately \( x/2^\gamma \) of the candidates have the all-ones pattern in the next \( \gamma \) bit positions. Thus, if \( r < x/2^\gamma \), it is likely that \( \gamma \) bits can be relaxed with a single search for the \( \gamma \) bit-all-ones pattern since the \( r \)th largest value is likely to have the all-ones pattern in these \( \gamma \) bits. If the search does in fact produce at least \( r \) responders, then we simply continue from bit position \( i + \gamma \) looking for the \( r \)th largest value among the responders. If, on the other hand, fewer than \( r \) responders are produced, then we repeat from bit position \( i \), using single-bit searches within the current \( \gamma \)-bit group or, more generally, with a suitably chosen smaller group size. We can also se-
lect the search subfield size optimally in a manner similar to that of Subsection 5.1.

6.2 Finding the k largest elements

Another form of rank-based selection is to identify the set of k largest values (rather than just the kth largest value). This algorithm is quite similar to the previous one, except that if \( y < r \), then all responders are marked as having been selected and the search continues for the \( r - y \) largest values among the non-responders.

6.3 Ordered retrieval and sorting

Ordered retrieval is the sequential reading out of a selected subset of AP words in nondecreasing or non-increasing order of keys contained in a given field. Ordered retrieval lets us pass on the results of various search and data selection functions to a conventional host computer in sorted order for further processing, archiving, or output.

The main component of most ordered retrieval schemes is a max- or min-finding algorithm. Simultaneous max- and min-finding can be used to effectively double the retrieval rate at the cost of some increase in hardware complexity [32]. It is also possible to mechanize the max-selection hardware in such a way that once the overall maximal element is identified, each successive maximum is identified and retrieved in one cycle [20], although this cycle is likely to be longer than a normal search cycle in view of the need for global communication (see subsection 7.3 on scalable APs). Ignoring these speedup techniques, the fundamental concept to be noted in ordered retrieval algorithms is a way to avoid spending the full w search cycles for identifying the next max or min element.

Consider ordered retrieval in nondecreasing order, requiring max-finding at each step. Once the overall maximum element(s) is (are) identified and retrieved, using the multiple response resolution mechanism if needed, the next largest value is likely to have many bits in common with the value just retrieved. Thus, it may be possible to avoid stepping through all w bitslices for max-finding among the remaining elements.

This can be accomplished by a backtracking scheme. For example, if 1110 is identified as the overall maximum, we backtrack from the 0 to the preceding 1 and search for another 0 in that bit-slice, identifying all entries containing 1100. We then move forward from the backtracking point, proceeding to search for 1101 and 1100 in turn.

The above algorithm is quite efficient when the number of elements to be retrieved is large (comparable in magnitude to \( 2^w \)), since in such a case, the average amount of backtracking, in bit slices, is fairly small.

7 Extensions and generalizations

In this section we deal with several extensions to the algorithms presented thus far. Modifications required for searching with signed integers and floating-point numbers are discussed in subsections 7.1 and 7.2, respectively. Some ideas for extending these algorithms for use with certain scalable AP organizations and more general massively parallel architectures are presented in subsections 7.3 and 7.4.

7.1 Searching with signed numbers

Numerical fields were assumed to be unsigned in the main body of the paper to simplify the exposition of algorithms. Let us consider briefly the modifications that would be required to deal with signed integers represented in sign-and-magnitude or in two’s-complement form for each search type.

Exact match. No change.

Inexact match. Numerical proximity search needs subtraction, computing of absolute values (through selective negation of negative differences), and min-finding. Alternatively, a more sophisticated min-absolute-value-finding algorithm can be developed directly that pursues separate search paths for nonnegative and negative values, but such an algorithm will not be more efficient than the simpler scheme already discussed due to overhead.

Relational searches. Values with one sign or the other can be ignored (the cells disabled) in relational searches. For example, when searching for values that are greater than a positive number, negative values can be removed from the candidate set in an initialization step.

Interval searches. Similar to relational searches.

Extrema searches. Consider max-finding (min-finding is similar). First, determine if there are nonnegative numbers in the initial candidate set. If so, remove all negative values from the set and perform unsigned max-finding on all bits excluding the sign bit. If not, then all the candidate numbers are negative. For sign-and-magnitude representation, perform unsigned min-finding. For two’s complement representation, perform unsigned max-finding. This last statement is justified [18] by noting that the two’s-complement number \( x_{n-1}x_{n-2} \ldots x_0 \) represents the
value \( -x_{1-1} \cdot 2^{-n} + (x_{n-2} \ldots x_0 1_0)_2 \). Next-higher and next-lower numerical searches are programmed through in-place subtraction followed by min-finding.

**General selection.** Consider the problem of determining the \( k \)th largest value in a candidate set containing \( p \) nonnegative values. If \( k \leq p \), then restrict the search to nonnegative values. On the other hand if \( k > p \), then the \((k - p)\)th largest value among the negative numbers must be found. Again with sign-and-magnitude representation, we must determine the \((k - p)\)th smallest and with two's-complement representation, the \((k - p)\)th largest value in the remaining bits.

**Ordered retrieval.** Handle nonnegative and negative values separately and in turn.

### 7.2 Searching with floating-point numbers

A floating-point number consists of a sign, an exponent (usually in biased format), and a significand. IEEE floating-point standard representation is assumed in the following [18], but the algorithms can easily be modified for other representations. For clarity, we will deal only with positive floating-point numbers. The sign can be handled by modifications similar to those discussed in subsection 7.1.

The relative numerical order and equality of two bit strings is the same whether they are viewed as binary fixed-point binary integers or floating-point representations with the exponent (in biased format) to the left of the significand. Thus, virtually all of the search and selection algorithms discussed in the previous sections of this paper apply, with no modification, to floating-point values. Note, however, that exact-match comparisons are usually avoided in floating-point arithmetic.

The only algorithm in need of special attention is that of **Inexact match**: In a vast majority of cases, numerical proximity search can be performed by simply limiting the search to values having the same exponent as the comparand. Exceptions occur when the comparand has a significand that is "very close,‖ as determined by the comparison threshold \( \varepsilon \) and the comparand’s exponent, to the lower and upper limit of its range (1 or 2 for the IEEE standard format). In such cases, the next higher or next lower exponent value must also be considered.

### 7.3 Searching on scalable APs

Recent advances in VLSI technology have made large APs and other massively parallel architectures practically realizable. Tens of thousands of processors or cells already appear in some commercially available systems and the use of one million processors is being contemplated. With such a large number of elements, optimization of processor design and scalability of processor interconnection mechanisms become major issues.

The above concerns have led to the formulation of the concept of systolic APs [24], which are built by interconnecting smaller building-block APs (BB-APs) and using them in a pipelined fashion [28]. Figure 5 shows a tree-structured systolic AP built of seven BB-APs, each containing 5 cells. Note that this small example is only for exposition. In practice, the number of cells per BB-AP would be much larger. The cells are numbered to facilitate communications between adjacent cells using the tree links. Instructions and associated data are given to the root and then move, in broadcast mode, to lower levels. Partial search results are combined in an upward sweep. Multiple instructions can be pipelined, provided they are scheduled to avoid data and control dependencies in a manner similar to the scheduling of instructions in a CPU instruction pipeline.

Some search algorithms work on such systolic architectures unmodified. Examples include less-than search (responding set is the union of the local sets) and max-finding (overall maximum is the largest of the local maxima). Others must be specifically designed with the particular systolic architecture in mind. For example, suppose that we want to search for the \( k \)th largest value in the AP of Figure 5 and that the BB-APs are bit-serial in architecture. The algorithm described in subsection 6.1 needs to know the number of responders in each bit slice to determine the set of candidates for the next step. Thus, the number of local responders must be summed in each step. Possible strategies for effective use of the hardware include interleaving of multiple independent searches (akin to exploiting multiple threads in some superpipelined microprocessors) and speculative execution of the more likely path or both branch paths by BB-APs pending the outcome of the tested condition.

### 7.4 APs as massively parallel processors

Associative processors constitute a prominent subclass of fine-grained, massively parallel SIMD (single-instruction-stream, multiple-data-stream) architectures [1], [14]. The validity and cost-effectiveness of the massively parallel processor (MPP) approach to high-performance computing and the relative merits of SIMD versus MIMD architectures have been the subjects of heated debates in recent years [30]. It is thus interesting to see where APs fit in the MPP world and how they are affected by these arguments.
A bit-serial AP, with a shift capability for its response store as the only communication mechanism between the cells, can be viewed as a linear array architecture with a broadcast bus. Each processor or cell in this equivalent linear array receives instructions and data via the common bus and has adequate storage to keep an entire word of the original AP and to randomly access its bits.

Fully parallel APs in which comparand and mask are broadcast to each cell and the cell match signal propagates sequentially through all the cell bits before affecting the tag, are architecturally equivalent to 2D meshes with column buses, as shown in Figure 6(a). Obtaining the cell mismatch signal through wire-ORing the individual bit mismatch signals within a cell would be akin to providing row buses as well.

If tree-structured circuits are used for operand broadcasting within columns and combining of match signals within rows, then the architecture can be viewed as an augmented 2D mesh of trees [21] depicted with a few of its row/column trees in Figure 6(b).
However, both 2D meshes with row/column buses and mesh of trees architectures are usually envisaged to have considerably more general message-passing capabilities than APs. Whereas all the algorithms presented in this paper can be run, with minor adjustments, on these more flexible architectures, they would suffer from significant overhead compared to their execution on an AP with its highly specialized communications structure.

A fundamental trend in the design of massively parallel APs over the past decade has been the provision of more general capabilities for communication and collective computation [13]. Even though the entire spectrum of systems from bare-bone APs to general-purpose MPPs must be examined for any given application, no architectural "feature" comes for free, and simpler AP architectures remain the most cost-effective for certain application areas.

8 Conclusions

Search and data selection algorithms for both fully parallel and bit-serial APs were reviewed and analyzed. Search and data selection functions covered included exact-match, inexact-match (numerical or logical proximity), membership (multiple match), relational (<, ≤, >, ≥), interval (between-limits), extrema (max, min, next higher, next lower), rank-based selection (the kth or k largest/smallest), and ordered retrieval (sorting). Various hardware features that facilitate or speed up these algorithms were also described. In addition to several algorithms that were presented for the first time, discussions touched on two topics that have been inadequately covered in previous treatments: (1) Explicit analysis of search algorithm complexity, and (2) Effects of hardware features and implementation details on the performance of search and data selection algorithms.

Both worst-case and average-case complexities of the algorithms were considered. For example, average-case-optimal extrema search strategies for fully parallel APs were discussed. It was shown that significant speedup can be obtained when searching in a small field with a large number of initial candidates. Such optimizations are not only relevant to fully parallel systems but may find applications in bit-serial systems as well. Typically in such systems, multiple-bit searches are provided as part of the basic instruction set. An instruction for a γ-bit search is likely to execute much faster than a sequence of γ single-bit search instructions. In such cases, similar optimality results can be obtained, although optimal group sizes are likely to be much smaller compared to those of the fully parallel case.

This paper does not exhaust the discussion of AP search algorithms. Many interesting variations were left out in the interest of brevity. Other search algorithms can be devised and existing ones can be analyzed in greater detail (for example, modeling bit-slice reads/writes and logical operations separately, with allowance for various degrees of overlap between them in bit-serial APs) or with more realistic assumptions about the distribution of bit patterns in the stored data. Cost-effective hardware aids for speeding up fully parallel or bit-serial searches are also needed. There are numerous opportunities for further research.

Our search algorithms were described assuming conventional number representations. Other representations or codes such as the residue number systems [27], constant-weight codes [2], and order-preserving codes [26], proposed in connection with special applications, as well as set-based computations [35], would require their own search and selection algorithms. However, given the framework of this paper, such algorithms can be readily derived.

Acknowledgment

The research was supported in part by the US National Science Foundation, under Grant MIP-9001618.

References


