Performance Analysis and Optimization of Search and Selection Algorithms for Highly Parallel Associative Memories

Behrooz Parhami

Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract

Several useful associative memory (AM) algorithms deal with identifying extreme values (max or min) in a specified field of a selected subset of words. Previously proposed algorithms for such extreme-value searches are bit-sequential in nature, even when implemented on fully parallel AMs. We show how the multiple-bit search capability of a fully parallel AM can be used to advantage in reducing the expected search time for finding extreme values. The idea is to search for the all-ones pattern within subfields of the specified search field in lieu of, or prior to, examining bit slices one at a time. Optimal subfield length is determined for fixed-size and variable-size bit groupings and the corresponding reduction in search time is quantified. The results are extended to rank-based selection where the jth largest or smallest value in a given field of a selected subset of words is to be identified. Analyses point to significant reduction in the average number of search cycles.

1. Introduction

Associative or content-addressable memories 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. An *associative memory* (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 AM words by n, where $n \leq N$.

When presented with a *search key* (*comparand*) and a *mask* specifying relevant field(s), the AM responds by *marking* all the words that satisfy the search requirements. Marking is done by (re)setting a *response bit* or *tag* in the AM cell. The N response bits together constitute the *response store*. A *response indicator* provides information on multiplicity (0, 1, several) or an exact count of the responders. When there is no responder, the search outcome is negative. With a single responder, the required data item has been located and can be appropriately dealt with by reading it out or modifying it in-place. With 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*.

Architecturally, associative memories are of four varieties: Fully parallel, bit-serial, word-serial, and block-oriented. Cost-effective bit-serial systems have been dominant in practical implementations, but fully parallel systems are also implemented, particularly where high performance for the basic masked exact-match search capability is required.

Starting with the pioneering works of Falkoff [3] and Estrin & Fuller [2], many algorithms have been developed for performing search, retrieval, and arithmetic/logic operations on data stored in AMs [1, 4, 7, 8]. Meanwhile, innovations in hardware acceleration methods and scalable AM architectures have made the implementation of larger systems, as an important subclass of high-performance SIMD architectures, attractive and practically feasible.

Even though bit-sequential arithmetic can be programmed on virtually all AMs, and special hardware features have been implemented/proposed for facilitating or speeding up numerical computations, the focus of AM implementation efforts and research studies has been searching and other non-numerical functions [5, 6]. Besides simple exact-match search, other types of searches can be implemented as primitives for data manipulation and retrieval [8]. These include approximate-match searches (similarity, adjacency, or threshold distance), relational searches (less than, greater than or equal to, etc.), and interval searches (combination of two relational searches).

Several useful AM search algorithms deal with identifying extreme values (max or min) in a specified field of a selected subset of words. In image processing, max/min pixel "intensities" are needed for normalization and contrast enhancement. Also, max or min pixel IDs are used for labeling an image's connected components, a frequently used operation in computer vision [9]. In other contexts, max or min finding is used for leader election (designating a processor as controller, arbiter, or coordinator) or in lieu of relational and between-limit searches to determine that a large collection of monitored objects/subsystems are all operating within safe margins. Selection (finding the jth largest/smallest value among n elements), and its special case of median finding where $j = n/2$, are similarly quite useful in image processing filters and data partitioning (e.g., in parallel divide-and-conquer algorithms for sorting).

Previously proposed algorithms for extreme-value search are bit-sequential in nature, and therefore relatively slow, even when implemented on high-performance fully parallel associative memories. In this paper, we show how the multiple-bit search capability of a fully parallel AM can be used to advantage in reducing the expected search time for finding extreme values. For brevity, we will discuss only maximum finding and selection of the jth largest value, but the proposed techniques are clearly applicable, with trivial adjustments, to minimum finding and selection of the jth smallest value as well.

2. Bit-Sequential Max Finding

2.1. Associative Memory Model

The AM model of interest here is the fully parallel model. Each AM cell compares the entire comparand, as masked by the content of a mask register, to its own content and sets a response tag if they match. The availability of a response indicator is assumed. A 3-valued indicator (0, 1, several) is adequate for the simpler algorithms while a response count is required for optimal adaptive versions. All N comparisons, setting of the AM tags, and response indication are performed within a single basic cycle which will be taken as unit time in the rest of the paper.

Clearly, a larger AM needs a longer cycle time compared to a smaller AM in view of the additional time required for instruction/operand broadcasting and multiple response indication and/or resolution. These variations, however, are irrelevant in the context of this paper in that we assume the availability of a fully parallel AM of a given size and compare our algorithms, in terms of the number of cycles, to the ones proposed for the same architecture using exactly the same assumptions.

2.2. A Bit-Sequential Algorithm

The following algorithm is described assuming an unsigned integer field. Modification to signed and non-integer values is straightforward [8]. The bit-sequential max-finding algorithm scans the field of interest, starting from the most significant bit. At the start of a typical step, corresponding to bit position i, a set of AM words are candidates for being maximum. The candidate set is identified by setting of one of several tag bits or by the contents of a specific bit-slice in the AM cells. A search is performed for the value of 1 in bit position i. With no responder, the candidate set remains the same, while with several responders, the candidate set is replaced by the set of responders. For a k-bit field, a maximum of k AM cycles, as defined in Subsection 2.1, are required.

2.3. Approximate Average-Case Analysis

The bit-sequential max-finding algorithm described in Subsection 2.2 may terminate before all k bits in the given field have been examined. To analyze the average-case behavior of this algorithm, let us assume that the candidate set is initially of size m and that bit values are randomly distributed (i.e., in any given cell and bit position, the probability of having a 1 is 0.5, independent of all other cells and bit positions).

Let k (w) be the (remaining) number of bits in the search field and $T_{k,m}$ denote the expected number of search cycles. We write an approximate recursive formula for $T_{w,x}$ based on the observation that with probability $x/2^x$ the search is terminated after inspecting the most-significant bit of the w-bit field (the probability that of the x bits in one bit-slice, exactly one is 1). If the search continues, however, there will be an expected number of $x/2$ candidates with a remaining search field of length $w - 1$. Thus:

$$T_{w,x} \approx 1 + (1 - x2^{-x})T_{w-1,\,x/2} \text{ with } T_{0,x} = T_{w,1} = 0 \qquad (1)$$

If one plots the variations of $T_{w,x}$ with the candidate set size x for several field widths w, it becomes apparent that, consistent with intuition, when x is greater than 2^w, the number of search cycles is roughly w, while for $x < 2^w$, the expected search time is a logarithmic function of x.

Eq. (1) is approximate since on its right-hand side, a single term with the expected number $x/2$ of remaining candidates is used instead of x different terms, one for each possible size of the remaining candidate set, appropriately weighted with its occurrence probability. For example, from Eq. (1) we get $T_{w,2} = 1$ whereas a more precise formulation yields $T_{w,2} = 1 + (1/2)T_{w-1,2} = 2 - 2^{-w}$ based on the observation that both cells remain candidates with probability 1/2 (i.e., when they hold equal bits). However, this is a worst-case example and Eq. (1) usually gives fairly accurate results.

2.4. Exact Average-Case Analysis

The following is an exact probabilistic analysis. Let y be the number of responders after searching for 1 in a 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 $w - 1$ bits. For $y = 1$, the search ends. Finally, for $y \geq 2$ (probability $\binom{x}{y}2^{-x}$ for each y), the search continues with y candidates in the remaining $w - 1$ bits. Thus:

$$T_{w,x}=1+2^{-x}T_{w-1,x}+2^{-x}\sum_{y=2}^{x}\binom{x}{y}T_{w-1,y} \text{ with } T_{0,x}=T_{w,1}=0 \qquad (2)$$

Fig. 1 shows the variation of $T_{w,x}$ with the candidate set size x for several residual widths w. Eq. (2) is computationally much more intensive than Eq. (1), but the results of this exact analysis closely match those obtained through approximate analysis, particularly for large x.

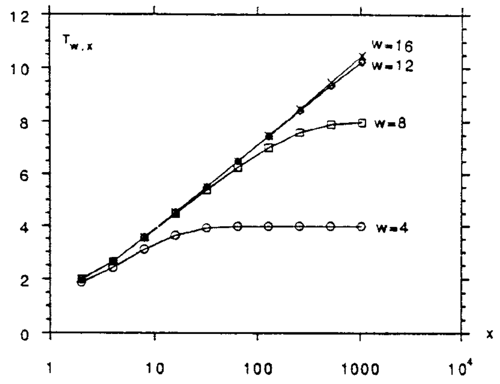

Fig. 1. Expected number of cycles in bit-serial max search.

3. Max Finding with Uniform Groups

3.1. Benefit of Multi-Bit Search

Intuitively when m is much larger than 2^k, it is very likely that the maximum value is $2^k - 1$, 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. Let the probability that a match is not found in the first step be $p_{k,m}$. Each word contains the all-ones pattern with probability 2^{-k}. Thus, the probability that none of the given m words holds the all-ones pattern is $p_{k,m} = (1 - 2^{-k})^m$. The search time is thus

$$T'_{w,x} = 1 + p_{w,x}\,T_{w,x} = 1 + (1 - 2^{-w})^x\,T_{w,x} \qquad (3)$$

where $T_{w,x}$ is the bit-serial search time of Eq. (2). Plots of $T'_{w,x}$ as a function of x for different values of w appear in Fig. 2. Comparing Figs. 1 and 2 reveals advantages of this new strategy when x is larger than 2^w.

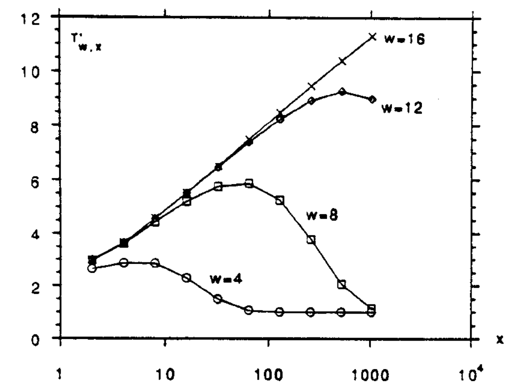

Fig. 2. Expected number of cycles in bit-serial max search with initial search for the w-bit all-ones pattern.

In order to achieve better performance for cases where x is smaller than or comparable to 2^w, one can perform g-bit block searches for the all-ones pattern rather than using the entire w-bit field. Intuitively, if the candidate set size is larger than 2^g, g bits may be relaxed in a single cycle.

3.2. Approximate Average-Case Analysis

We write a recursive formula for $T^{(g)}_{w,x}$ (the expected time for max finding using uniform groups of size g) based on the observation that with probability

$$x/2^g \sum_{v=1}^{2^g-1} (v/2^g)^{x-1}$$

the search is terminated after inspecting the most-significant g bits of the w-bit field (this is the probability that of the x g-bit fields, exactly one has the value v and all others are less than v for some v in the range $1 \le v < 2^g$). If the search continues, however, there will be an expected number of $x/2^g$ candidates with a remaining search field of length $w - g$. If the candidate set size is x, the probability that a match is not found when searching for the all-ones pattern in the current g-bit block is $(1-2^{-g})^x$. In this case, one can resort to bit-sequential search for the current block and treat the subsequent blocks in the same way. Thus:

$$T^{(g)}_{w,x} \approx 1+(1-2^{-g})^x T_{g,x}+(1-x/2^g \sum_{v=1}^{2^g-1}(v/2^g)^{x-1})T^{(g)}_{w-g,x/2^g} \quad (4)$$

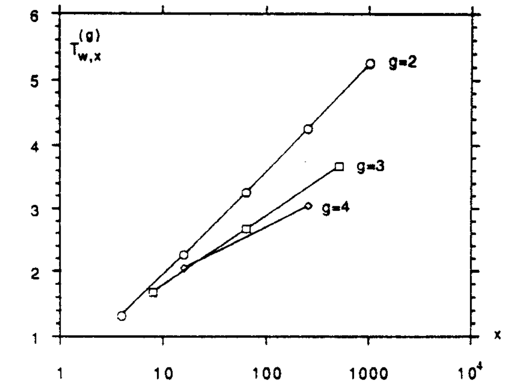

Fig. 3. Expected number of cycles in max search in a field of width w = 12 with fixed group size g.

The initial conditions for Eq. (4) are the same as those for Eq. (1). Fig. 3 shows the variation of $T^{(g)}_{w,x}$ with the candidate set size x for several values of group size g assuming w=12. The reader should not be misled by the straight lines drawn through the points in Fig. 3 in order to show the trends. A detailed plot with more points would reveal significant jitter (resulting from divisibility and rounding effects) which is typical of situations when continuous analysis is used for integer-valued parameters. As an example, for x = 64, the group size g = 3 is better than g = 4, whereas Fig. 3 seems to indicate otherwise.

3.3. Exact Average-Case Analysis

We write the following recurrence for expected search time:

$$T^{(g)}_{w,x} = 1 + (1 - 2^{-g})^x\, T_{g,x} + \sum_{y=2}^{x} q_{g,x,y}\, T^{(g)}_{w-g,y} \quad (5)$$

where $q_{g,x,y}$ is the probability of candidate set size being reduced from x to y after relaxing a g-bit block. At the end of this recursion, when $w<g$, we set $T^{(g)}_{w,x}$ to $T_{w,x}$. Note that the first two terms on the right-hand side of (5) are identical to those in (4). The probability $q_{g,x,y}$ is:

$$q_{g,x,y} = 2^{-gx}\, d(x-y) + \sum_{v=1}^{2^g-1} \binom{x}{y} 2^{-gy}\, (v/2^g)^{x-y} \quad (6)$$

The first term in Eq. (6) corresponds to the case of no 1 in any of the g bit slices of the current group, with $d(x - y)$ being the step function which is 1 if x=y and 0 otherwise. In other words, the maximum value in the g-bit field is 0 with probability 2^{-gx} and in this case, y will be equal to x. Each term in the summation corresponds to the y responders having the value v ($1 \le v \le 2^g-1$) and all the x–y non-responders having values less than v. Again, results from the exact analysis closely match the approximate ones for large x. Observe that in the limiting case of g=w, the sum term in (5) becomes zero and the scheme leading to Eq. (3) results. Thus $T^{(w)}_{w,x} = T'_{w,x}$, as expected.

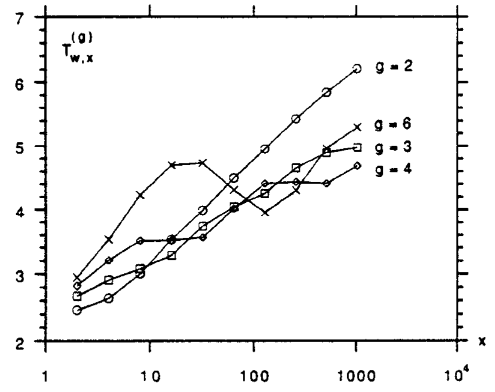

Fig. 4. Expected number of cycles in max search in a field of width w = 12 with fixed g (exact analysis).

Fig. 4 shows the variation of $T^{(g)}_{w,x}$ with x and g for w=12. The jitter and periodic variations due to divisibility and rounding effects are evident here. However the overall trend of larger group sizes doing better as x gets larger is also clearly seen. The periodic variation is due to the tail end of the recursion, as discussed earlier. As an example, g = 6 does better than g = 2, 3 , or 4 for x = 128 = 2^7 and again becomes better as x is increased beyond 2^{14}.

3.4. On Optimal Group Size

Intuitively, an optimal value for g exists since for g too large, block searches are unlikely to be successful, while if g is too small, we won't do much better than bit-sequential search. Even though the optimal value for g cannot be derived in closed form, the above analyses help in selecting a good value for g. Figs. 3 and 4 show that the optimal group size g^{opt} for w = 12 (giving the lowest value of $T^{(g)}_{w,x}$) shifts from 2 to 3 and from 3 to 4 as the candidate set size x grows beyond 8 and 32, respectively.

4. Max Finding with Adaptive Groups

4.1. A Greedy Max-Finding Algorithm

If g can be picked arbitrarily, then it can be adjusted at each step to minimize the expected residual search time. As an approximation to this globally optimal approach, one may try a greedy strategy of minimizing the per-bit search time for the next few bits. This is called greedy since it is based on maximizing the immediate gain as opposed to following a globally optimal course.

Consider the following algorithm. In Phase i, a block size g is selected and a search for the all-ones pattern within that block is conducted. If matches are found, Phase $i + 1$ is initiated. Otherwise, Phase i is completed by performing g bit-sequential searches. Our greedy strategy is based on minimizing the per-bit search time for the current phase. To execute this algorithm, an exact or approximate count of the number of responders is required in each phase.

The probability of not finding an all-ones pattern in a g-bit field of x words is $(1 - 2^{-g})^x$. Thus, the expected per-bit search time within the current g-bit block is

$$t_{g,x} = 1/g + (1 - 2^{-g})^x \qquad (7)$$

where $1/g$ is the per-bit overhead of the initial g-bit search. Fig. 5 shows the logarithmic growth of g^{opt} with x. The least-squares straight line through the data points is really $g = (67/84)\log_2 x - 5/28$. Ignoring the first two points for $\log_2 x \leq 4$ in view of boundary effects, the least-squares fit becomes $g = (61/70)\log_2 x - 38/35$. This line (dotted in Fig. 5) matches the data points for $\log_2 x \geq 6$ perfectly, to within the jitter expected for an integer-valued function.

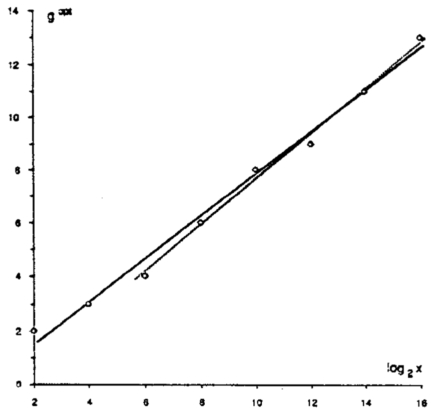

Fig. 5. The optimal group size when bit-sequential search is used after an unsuccessful block search.

These results are consistent with intuition. A block size of $\log_2 x$ yields an average of one responder. Therefore, in some cases there will be no responder and the block search cycle goes to waste. A block size slightly less than $\log_2 x$ increases the likelihood of finding some all-ones patterns while at the same time it relaxes a relatively large number of bits in the event of finding such matching patterns.

4.2. Alternate Adaptive Strategies

Instead of resorting to bit-sequential search within a group of size g when the search for the all-ones pattern produces no responder, one can try again with a smaller group size. Possible choices for reduced group size are g–1, $g/2$, etc. Following is an approximate analysis for a class of algorithms in which the group size is reduced from g to $f(g)$ after any unsuccessful block search. Using a greedy strategy to minimize the expected search time for the immediately following bit, we get:

$$t_{g,x} = 1/g + (1 - 2^{-g})^x \, t_{f(g),x} \text{ with } t_{1,x} = 1 \qquad (8)$$

Fig. 6 shows g^{opt} for $f(g) = \lfloor g/2 \rfloor$. The least-squares straight line through the points is $g = (6/7)\log_2 x - 3/14$. The first two data points corresponding to $x \leq 4$ exhibit the boundary effect more drastically here. Ignoring these points, the least-squares fit becomes $g = \log_2 x - 2$ (dotted line). The optimal group size is slightly larger here, as expected.

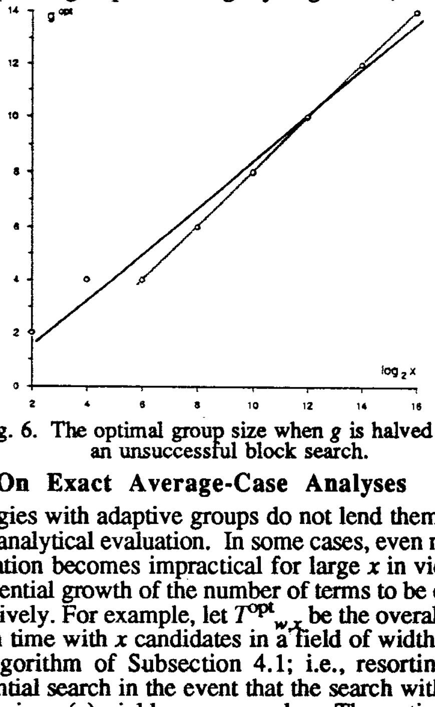

Fig. 6. The optimal group size when g is halved after an unsuccessful block search.

4.3. On Exact Average-Case Analyses

Strategies with adaptive groups do not lend themselves to exact analytical evaluation. In some cases, even numerical evaluation becomes impractical for large x in view of the exponential growth of the number of terms to be evaluated recursively. For example, let $T^{opt}_{w,x}$ be the overall optimal search time with x candidates in a field of width w, using the algorithm of Subsection 4.1; i.e., resorting to bit-sequential search in the event that the search with optimal group size $g(x)$ yields no responder. The optimal group size function $g(x)$ must be determined from a recursive equation involving two x-term sums, with each term being the product of an event probability and a residual optimal time expression of the form $T^{opt}_{w-g(x),y}$. The situation is much worse for the algorithms of Subsection 4.2 since both the $g(x)$ and $f(g)$ must be determined.

In all cases where we have been able to perform numerical experimentation, the logarithmic growth rate of the optimal group size g^{opt} is observed asymptotically. The differences between various strategies are significant only for boundary cases involving small values of x. It is, therefore, quite possible to begin the search with one strategy that exhibits good performance for large x and to then switch to a different strategy as x becomes smaller. Ways of combining such strategies and the optimal switching point need further study.

5. General Rank-Based Selection

5.1. A Bit-Sequential Selection Algorithm

Assuming that an exact count for the number of responders is available after each search, the jth largest value held within a set of m candidate fields of length k can be identified through bit-sequential searching as follows. There are k steps in the algorithm. Step i ($0 \le i < k$) starts with x candidates among which the rth largest must be identified ($r \le x$; initially, $x = m$, $r = j$). Let there be y responders to the search for 1 in the ith most significant bit of the field. If $y \ge r$, then the search continues among the responders for the rth largest value. If $y < r$, then the required value must have a 0 in the current bit slice. Thus, the search continues among the non-responding subset of the original set of candidates for the (r–y)th largest value.

Average-case time complexity of this algorithm is found in a manner similar to that of bit-sequential max finding:

$$S_{w,r,x} = 1 + \sum_{y=r}^{x} \binom{x}{y} 2^{-x} S_{w-1,r,y} + \sum_{y=0}^{r-1} \binom{x}{y} 2^{-x} S_{w-1,r-y,x-y} \quad (9)$$

Boundary conditions for Eq. (9) are $S_{w,1,1} = S_{0,r,x} = 0$.

5.2. Selection with Block Searches

Consider a g-bit subfield starting at position i and let there be x candidates at this point. Approximately $x/2^g$ of the candidates have the all-ones pattern in the next g positions. Thus if $r < x/2^g$, it is likely that g bits can be relaxed with a single search for the g-bit all-ones pattern. If the search does in fact produce at least r responders, then we simply continue from bit position i+g, looking for the rth largest value among the responders. If, on the other hand, there are fewer than r responders, we repeat from bit position i, using single-bit searches within the current g-bit group or, more generally, with a smaller group size $h(g)$.

5.3. A Greedy Adaptive Selection Algorithm

The greedy adaptive strategy for selection is quite similar to that used for max finding in Subsection 4.2. Using an integer-valued group size reduction function $h(g)$ and a greedy strategy to minimize expected search time for the immediately following bit, the quantity to be minimized is

$$s_{g,x} = 1/g + \Big[\sum_{y=0}^{r-1} \binom{x}{y} 2^{-gy} (1-2^{-g})^{x-y}\Big] s_{h(g),x} \text{ with } s_{1,x} = 1 \quad (10)$$

It is instructive to analyze the algorithm's behavior in the important special case of median finding; i.e., for $r = \lfloor x/2 \rfloor$. In the initial step, the optimal group size is g=1, leading to $y \approx x/2$ responders. If $y \ge \lfloor x/2 \rfloor$, then the rth largest value among the responders is sought. The optimal group size in this second step is likely to be large in view of the fact that r is likely to be very close to y and a search for the all-zeros pattern can be done to relax several bits at once. For $y < \lfloor x/2 \rfloor$, the (r–y)th largest value among the non-responding candidates must be identified. Again, r–y is likely to be small, leading to a large optimal group size.

6. Conclusions

We have shown how the multiple-bit search capability of fully parallel AMs can be used to improve the average-case performance of extreme-value searches and rank-based selection algorithms. Other searches for which these speedup techniques are applicable include identifying the j largest values (rather than the jth largest) in a given field of a set of x words, finding the next larger/smaller value (greater-than or less-than search combined with min or max finding), and range searches. The improvement results from the ability to relax several bits of the field under consideration at once. Analyses were offered for fixed-size and adaptive groups and optimal group size was discussed.

The expected speedup offered by our extreme-value search algorithms depends on the width of the search field and the initial size of the candidate set. With optimal fixed group length, speedups in the range 3-5 have been observed. Adaptive groups, implying somewhat more complicated control computations to set up the required comparands and masks in successive steps, offer speedups that are typically 2-3 times larger than those with fixed-size groups.

The results presented in this paper can be extended in many directions. These include experimental verification of the derived complexity and optimality results via simulation. Also of interest are refinements of some of the analytical results to obtain good approximations or computationally more tractable exact formulae. Finally, the entire class of useful AM search functions, including more complicated approximate and multidimensional searches, could be examined in order to determine the applicability of these and similar speedup and optimization techniques.

The techniques presented here are not only relevant to fully parallel systems but may also find applications in bit- and byte-serial systems. Designers of bit-serial AMs, e.g., typically provide multiple-bit searches as part of the basic instruction set with built-in control of bit-sequential steps. On such systems, an instruction for a g-bit search is likely to execute much faster than a sequence of g single-bit search instructions. In these cases, similar optimality results can be obtained, although optimal group sizes are likely to be much smaller compared to our results.

References

[1] Davis, W.A. and D.-L. Lee, "Fast search algorithms for associative memories", *IEEE Trans. Computers*, Vol. 35, No. 5, pp. 456-461, May 1986.

[2] Estrin, G. and R.H. Fuller, "Algorithms for content-addressable memories", *Proc. IEEE Pacific Computer Conf.*, 1963, pp. 118-130.

[3] Falkoff, A.D., "Algorithms for parallel search memories", *J. ACM*, Vol. 9, pp. 488-511, Oct. 1962.

[4] Feng, T.-Y., "Search algorithms for bis-sequential machines", *J. Parallel & Distributed Computing*, Vol. 8, No. 1, pp. 1-9, Jan. 1990.

[5] Jalaleddine, S.M.S. and L.G. Johnson, "Associative IC memories with relational search and nearest match capabilities", *IEEE J. Solid-State Circuits*, Vol. 27, No. 6, pp. 892-900, June 1992.

[6] Kapralski, A., "The maximum and minimum selector SELRAM and its application for developing fast sorting machines", *IEEE Trans. Computers*, Vol. 38, No. 11, pp. 1572-1577, Nov. 1989.

[7] Lee, D.-L. and W.A. Davis, "An $O(n+k)$ algorithm for ordered retrieval from an associative memory", *IEEE Trans. Computers*, Vol. 37, pp. 368-371, Mar. 1988.

[8] Parhami, B., "Search and data selection algorithms for associative processors", In *Associative Processing and Processors*, A. Krikelis and C. Weems (Eds.), IEEE Computer Society Press, to appear.

[9] Weems, C.C. and D. Rana, "Reconfiguration in the low and intermediate levels of the image understanding architecture", *Reconfigurable Massively Parallel Computers*, H. Li & Q. Stout (Eds.), Prentice-Hall, 1991, pp. 88-105.