Extreme-Value Search and General Selection Algorithms for Fully Parallel Associative Memories

BEHROOZ PARHAMI

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA
E-mail: parhami@ece.ucsb.edu

Several useful associative memory algorithms deal with identifying extreme values (maximum or minimum) 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 associative memories. We show how the multiple-bit search capability of a fully parallel associative memory 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 both 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 *j*th largest or smallest value in a given field of a selected subset of words is to be identified. We conclude that significant reduction in the number of search cycles is possible in most practical extreme-value search and selection problems.

Received May 18, 1993; revised February 26, 1996

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. Also, more functional variants of such systems known as associative or content-addressable processors and database machines (as complete systems, front-end data filters or back-end data servers) have been the subjects of extensive research. The origins of associative memories can be traced back to the mid-1940s when Konrad Zuse sketched an associative relay circuit [1] and Vannevar Bush published his visionary assessment of the need for associative access to information [2]. A decade later, cryogenic elements provided the first viable cell technology for large-scale implementation of associative memories [3]. Further developments in this area are amply documented in various books, article collections, and survey papers [4–12].

In its simplest form, 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. Figure 1 depicts a functional view and the main elements of an AM as used in this paper. We denote the number of nonempty AM words by n, where $n \leqslant N$.

When presented with a *search key* (also known as the '*comparand*'), a *mask* specifying the relevant field(s) of the stored words, and possibly an *instruction* containing the type of search, the AM responds by *marking* all the words that *match* the specified key or, more generally, satisfy the search requirements given in, or implied by, the instruction. Marking is done by setting or resetting a *response bit* or *tag (bit)* in the corresponding AM cell. The N response bits together form the AM's *response store*. A *response indicator* mechanism, which is a part of the global tag operations unit, may provide information on the multiplicity of responders (zero, one, several) or an exact count of the number of responders. If there is no responder, the search outcome is negative and we can proceed to the next step of our algorithm corresponding to such an outcome. 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*.

Architecturally, associative memories come in four basic varieties: fully parallel, bit-serial, word-serial and block-oriented [10]. Bit-serial systems have been dominant in practical implementations in view of their cost-effectiveness [5, 13, 14]. However, fully parallel systems have also been implemented, particularly where high performance for the basic masked exact-match search capability has been required.

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. Starting with the pioneering works of Falkoff [15] and Estrin and Fuller [16], research in associative memory algorithms has continued up to the present [17–23].

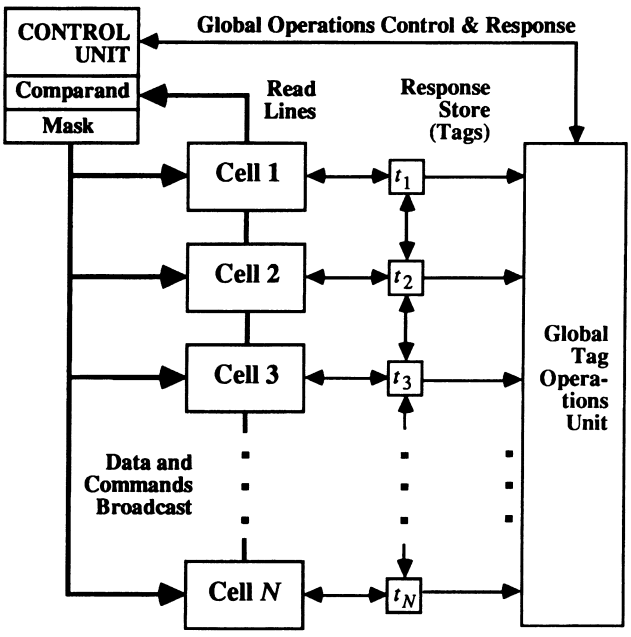

FIGURE 1. Functional view of an associative memory.

At the same time, innovations in hardware acceleration mechanisms and scalable AM architectures [24–28] have made the implementation of larger systems attractive and practically feasible. Such AM and AM-based systems constitute an important subclass of SIMD (single instruction stream, multiple data stream) architectures that over the years have provided some of the most cost-effective alternatives for high-performance computing [29, 30].

Even though bit-sequential arithmetic operations can be programmed on virtually all AMs, and special hardware features have been implemented or suggested in AM systems for facilitating or speeding up numerical computations, the primary focus of AM implementation efforts and research studies has been searching and other non-numerical functions. In addition to simple exact-match search, many other varieties of search types have been implemented as primitives for data manipulation and retrieval functions [31]. 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 (maximum or minimum) in a specified field of a selected subset of words. For example, in image processing, the maximum and minimum pixel 'intensities' are needed for normalization and contrast enhancement. Also, maximum or minimum pixel IDs may be used for labeling an image's connected components, a frequently used operation in computer vision [32]. In other contexts, max- and min-finding can be 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 or subsystems are all operating within safe margins. Selection (finding the jth largest or smallest value within a set of n elements), and its special case of median finding where $j = n/2$, are similarly quite useful in image processing filters, data partitioning (e.g. in parallel divide-and-conquer algorithms for sorting) and many other areas. In some such applications, extreme-value searches and rank-based selection are needed to an extent that the speed of these operations has a significant impact on the overall system performance. Hence our interest in making such algorithms more efficient.

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 or equivalent network-based architectures such as broadcasting [33] and reconfigurable [34] mesh-connected array processors. 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 and clarity of exposition, 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.

The remainder of this paper is organized as follows. In Section 2, we specify a fully parallel AM model that will be used for developing several extreme-value search algorithms and describe the standard bit-sequential algorithms for finding maximums and minimums. Next, we show how the search algorithms can be speeded up by inspecting several bits at a time and suggest how an algorithm with fixed-size bit groups might work (Section 3). We then examine variable-length grouping of bits in optimal adaptive algorithms (Section 4) and present some experimental results on the attainable speedup with fixed-group and adaptive-group search strategies (Section 5). Section 6 deals with extension of the methods to general selection and other search problems. Section 7 concludes the paper with a review of the relative advantages of our proposed approach and directions for further research.

2. BIT-SEQUENTIAL MAX FINDING

2.1. AM model

The AM model of interest in this paper is the fully parallel model. Each AM cell compares the entire comparand, as masked by the contents of a mask register, with its own content and sets a response tag if they match. We assume the availability of a response indicator. A three-valued indication (zero, one, several) is adequate for some of the simpler algorithms while an actual response count is required for the 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 will need a larger cycle time compared with a smaller AM in view of the additional time required for instruction/operand broadcasting and multiple

response indication and/or resolution [11, 26, 27]. However, these variations are not relevant 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 search cycles needed, 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. The bit-sequential maximum 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. Depending on the number of responders, the candidate set is modified as follows:

None:	Candidate set does not change.
One:	Stop; maximum already identified.
Several:	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. Before presenting an exact probabilistic analysis for the average-case behavior of this algorithm (Subsection 2.4), in the next subsection we deal with a simpler approximate analysis. Such approximate analyses appear in subsequent sections of the paper as well, since they facilitate the understanding of the more elaborate, and less intuitive, exact versions while also providing surprisingly accurate results on their own.

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 are 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 be the number of bits in the field of interest and $T_{k,m}$ denote the expected number of search cycles for maximum finding. We can 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 (this is the probability that of the x bits in one bit slice, exactly one is 1 and all others are 0). 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:

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

If one plots the variation of $T_{w,x}$ with the candidate set size x for several values of the field width w, it becomes apparent that when x is greater than 2^w, the number of search cycles is approximately w. On the other hand for $x < 2^w$, the expected search time is a logarithmic function of x. These results are consistent with intuition.

Equation (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 the probability of its occurrence. For example, from equation (1) we find $T_{w,2} = 1$ 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 values in the bit slice). However, this is a worst-case example and in most other situations, equation (1) 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 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 $w-1$ bits. For $y = 1$, no further search is necessary. Finally, if there are $y \geqslant 2$ responders (probability $\binom{x}{y}2^{-x}$ for each value of y), the search continues with y candidates in the remaining $w-1$ bits. Thus:

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

Figure 2 shows the variation of $T_{w,x}$ with the candidate set size x for several values of the field width w. Equation (2) is computationally much more intensive than Equation (1), but the results of this exact analysis closely match those obtained through approximate analysis in Subsection 2.3, particularly for relatively larger values of x.

3. MAX FINDING WITH UNIFORM GROUPS

3.1. Taking advantage 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}. Therefore, the probability that none of the given m words contains the all-ones pattern is $p_{k,m} = (1 - 2^{-k})^m$. Denoting the search time with this strategy as $T'_{w,x}$, we have

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

where $T_{w,x}$ is the bit-serial search time derived in Subsection

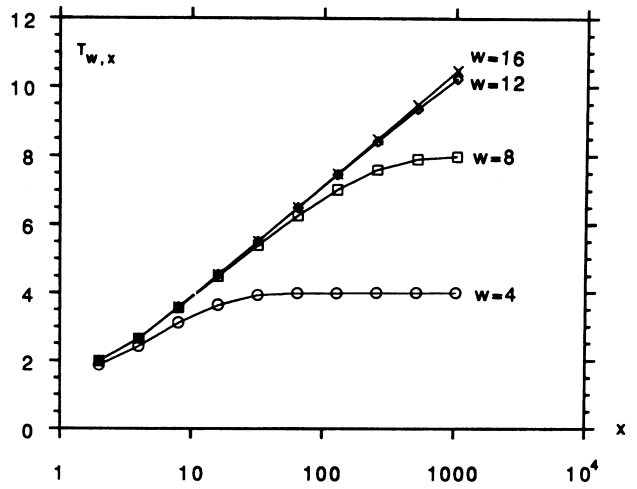

FIGURE 2. Expected number of cycles in bit-serial max search.

2.4. Plots of the value of $T'_{w,x}$ 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 for the case where x is larger than 2^w.

Based on the above observations, and in order to achieve better performance for cases where x is smaller or not much larger than 2^w, one can perform block searches using blocks of g bits for the all-ones search instead of using the entire w-bit wide field. Again, intuition suggests that when the candidate set size is larger than 2^g, one may be able to relax g bits in one cycle by searching for the g-bit all-ones pattern.

3.2. Approximate average-case analysis

Let $T^{(g)}_{k,m}$ denote the expected number of search cycles for max finding using uniform groups of size g. We can write an approximate recursive formula for $T^{(g)}_{w,x}$ 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 $1 \leqslant v < 2^g$). 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 in a given step 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 with the same two-step process. Thus:

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

The initial conditions for equation (4) are the same as those for equation (1), i.e. $T^{(g)}_{0,x} = T^{(g)}_{w,1} = 0$. Figure 4 shows the variation of $T^{(g)}_{w,x}$ with the candidate set size x for several values of group size g assuming a field width of $w = 12$ bits. The reader should not be misled by the straight lines drawn through the points in Figure 4 in order to show the trends. A more detailed plot with more points (not included here) reveals significant jitter (resulting from divisibility and rounding problems) 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 Figure 4 seems to indicate otherwise.

Equation (4) is approximate since on its right-hand side, a single term with the expected number $x/2^g$ of remaining candidates is used instead of x different terms, one for each possible size of the remaining candidate set, appropriately weighted with the probability of its occurrence.

3.3. Exact average-case analysis

With an argument similar to that of Subsection 3.2, we can

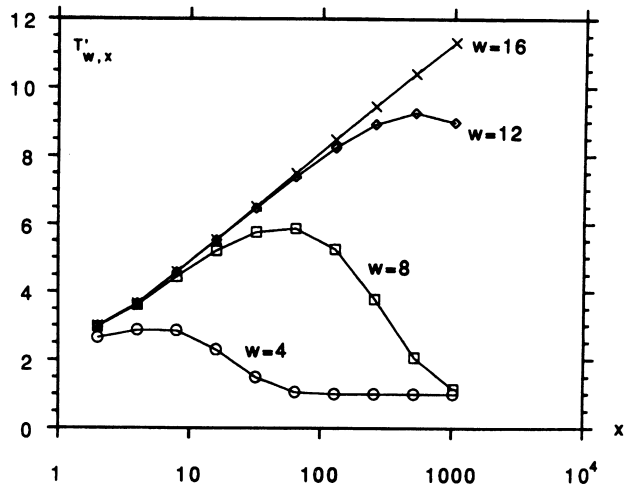

FIGURE 3. Expected number of cycles in bit-serial max search with initial search for the w-bit all-ones pattern.

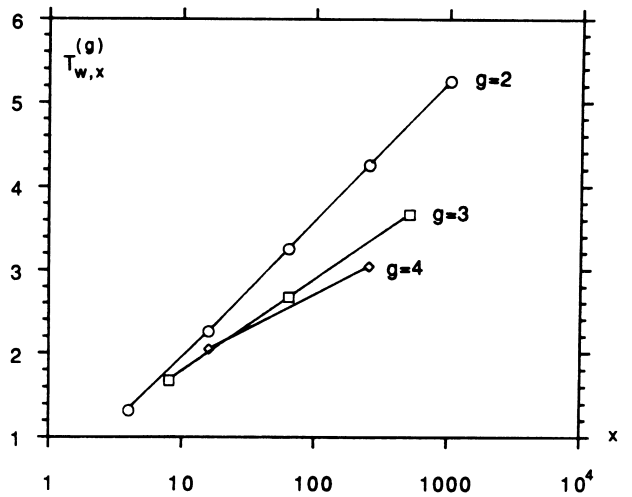

FIGURE 4. Expected number of cycles in max search in a field of width $w = 12$ with fixed group size g (approximate analysis).

write the following exact equation for the expected search time with uniform groups of size g:

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

where $q_{g,x,y}$ is the probability of the number of candidates being reduced from x to y after relaxing the current g-bit block. Near the end of this recursive process, we will have $w < g$. Then, $T^{(g)}_{w,x}$ is set equal to $T_{w,x}$. Notice 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 computed as follows:

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

The first term in equation (6) corresponds to the case of no 1 in any of the g bit slices in the current group, with $\delta(x - y)$ being the step function whose value is equal to 1 only 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 $v (1 \leqslant v \leqslant 2^g - 1)$ and all the $x - y$ non-responders having values less than v. Again, results from the exact analysis closely match the approximate ones for larger values of x. Observe that in the limiting case of $g = w$, the summation term in (5) becomes zero and the scheme discussed at the beginning of Subsection 3.1 results. Thus $T^{(w)}_{w,x} = T'_{w,x}$, as expected.

Figure 5 shows the variation of $T^{(g)}_{w,x}$ with x and g, assuming a field width of $w = 12$ bits. The jitter and periodic variations due to the effects of divisibility and rounding are evident here. However, the overall trend of larger group sizes performing better as x gets larger is also clearly seen. The periodic variation is due to the tail end of the recursion as discussed above. For example, $g = 6$ performs 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 if g is too large, block searches are less likely to be successful (particularly as the candidate set size shrinks) while if g is too small, we approach the performance of a slow bit-sequential search. Even though the optimal value for g cannot be derived in closed form, the above analyses can help in selecting a good value for g. For example, both Figures 4 and 5 show that the optimal group size g^{opt} for $w = 12$, corresponding to the lowest value of $T^{(g)}_{w,x}$ in the figures, 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 the block size g can be picked arbitrarily, then it can be adjusted at each step to minimize the expected remaining search time. As a first approximation to this global optimization problem, one may try a greedy strategy of minimizing the per-bit search time for the next few bits. This approach is called greedy since it is based on maximizing the immediate gain as opposed to following a globally optimal course.

To be more precise, consider the following algorithm. In Phase i of the algorithm, 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. Using a greedy strategy, the per-bit search time for the current phase is taken as the quantity to be minimized. To enable the selection of an optimal group size, an exact or approximate count of the number of responders is required after the search 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 contribution of the initial g-bit block search and $(1 - 2^{-g})^x$ is the probability of needing an extra search per bit position.

Figure 6 shows the values of g^{opt} that minimize equation (7) for different values of x. It is seen that g^{opt} grows roughly as $\log_2 x$. The least-squares straight line through the data points shown in Figure 6 is actually $g = (67/84)\log_2 x - 5/28$. Ignoring the first couple of points corresponding to $\log_2 x \leqslant 4$ in view of possible boundary effects, the least-squares straight line becomes $g = (61/70)\log_2 x - 38/35$. This line (dotted in Figure 6) matches the data points for $\log_2 x \geqslant 6$ perfectly, to within the jitter expected for an integer-valued function.

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

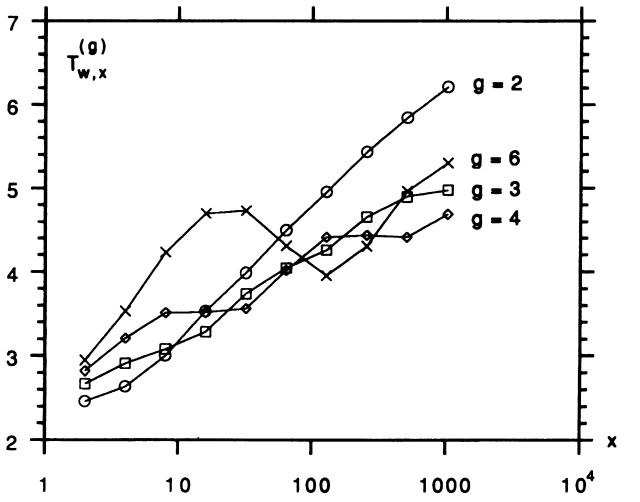

FIGURE 5. Expected number of cycles in max search in a field of width $w = 12$ with fixed group size g (exact analysis).

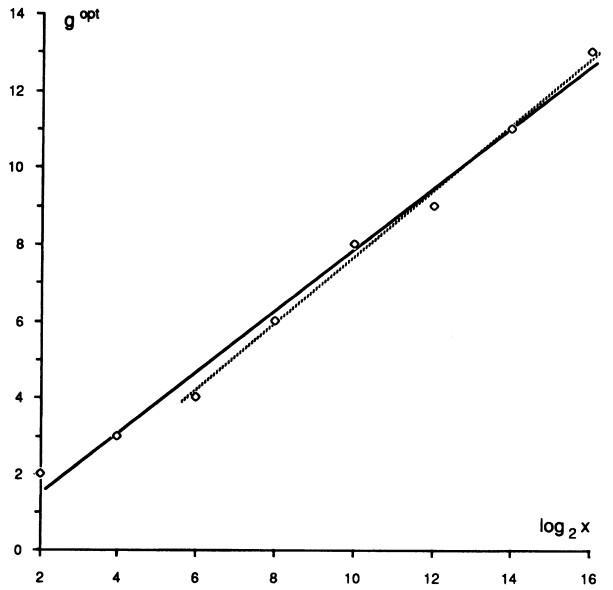

FIGURE 6. The optimal group size g^{opt} as a function of the number of candidates x assuming the use of bit-sequential search after an unsuccessful block search.

bits in the event of finding such matching patterns among the candidate words.

4.2. Alternate adaptive max finding 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 the reduced group size are $g-1$, $g/2$, etc. This approach should lead to better overall performance by increasing the initial group size g^{opt} (the time penalty for not finding any all-ones responder is not so great now). The analyses of these schemes are quite complicated and computationally intensive.

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. The group size reduction function $f(g)$ can be any integer-valued function satisfying $f(g) < g$. Let us again use a greedy strategy to minimize the expected search time for the immediately following bit. This can be written as

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

where $1/g$ is the per-bit contribution of the initial g-bit block search and $(1-2^{-g})^x$ is the probability of stepping down to the next lower block of size $f(g)$ due to failure to find any all-ones pattern of length g.

The algorithm discussed in Subsection 4.1 can be viewed as a special case of the above with $f(g) = 1$. Figure 7 shows the values of g^{opt} that minimize equation (8) for different values of x using the function $f(g) = \lfloor g/2 \rfloor$. Again g^{opt} grows roughly as $\log_2 x$. The least-squares straight line through the data points shown in Figure 7 is $g = (6/7)\log_2 x - 3/14$. The first couple of data points corresponding to $x \leqslant 4$ show the boundary effect more drastically here. Ignoring these points, the least-squares straight line becomes $g = \log_2 x - 2$ (dotted line in Figure 7). As expected, the optimal group size is slightly larger than that obtained in Subsection 4.1.

4.3. On exact average-case analyses

We have developed recursive formulae for the exact average-case analysis of the above and various other adaptive schemes. Most of these schemes do not lend themselves to 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 summations, with each summation 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 $f(g)$ functions need to be determined.

Despite this difficulty of analyses, however, application of such methods remains practical. The time-consuming computations are performed only once, perhaps in an approximate manner, with the results converted to simple heuristics or pre-stored tables in order to make the run-time overhead acceptable (see also the final paragraph in Subsection 5.2).

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

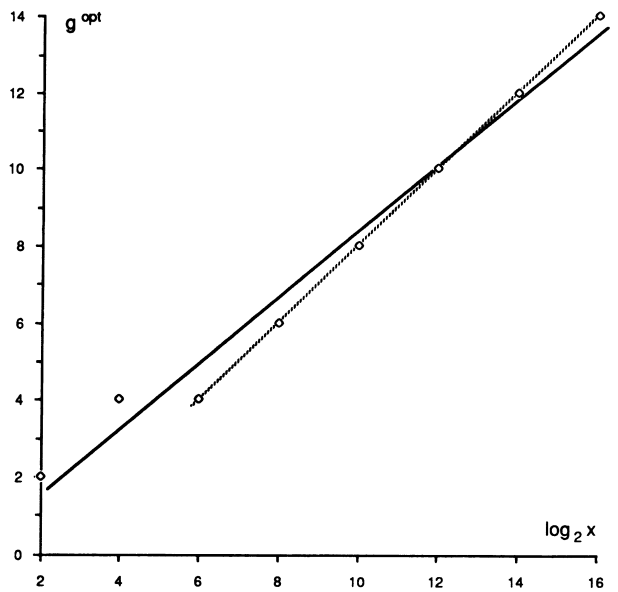

FIGURE 7. The optimal group size g^{opt} as a function of the number of candidates x assuming the group size is halved after an unsuccessful block search.

for boundary cases involving small values of x. It is, therefore, quite possible to begin the search with one strategy that exhibits better performance for larger values of 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. PERFORMANCE OF MAX-FINDING

5.1. Experimental assessment of speedup

The expected speedups offered by our extreme-value search algorithms depend 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 with realistic parameter values. Adaptive groups, which imply somewhat more complicated control computations to set up the required comparands and masks in successive steps of the search algorithm, offer speedups that are typically two to three times larger than those with fixed-size groups.

As an example, we have used simulation to estimate the speedup attained with various methods when searching for extreme values in a field of width $w = 12$ bits. Each data point represents the average performance observed over thousands of trials. In each trial, the $x \times w$ AM matrix was filled with randomly generated data, the desired search algorithms were run to completion, and the number of search cycles was noted. The average speedup was then computed by dividing the bit-sequential search time by the average number of search cycles obtained from the simulation.

Figure 8 shows the variation of the resulting speedup as a function of the initial number of candidates x for three search schemes:

1. Fixed group size of $g = 12$ bits, i.e. preceding the bit-sequential search with a single search for the 12-bit all-ones pattern.
2. Fixed group size of $g = 6$ bits, thus leading to at least two searches and a maximum attainable speedup of 6.
3. Using adaptive groups based on the greedy algorithm discussed in Section 4 and halving of group size after each unsuccessful search.

When the number x of candidates is relatively small, say up to a few tens, it is often faster to read out the candidate words and perform the required manipulations within the sequential host computer as opposed to within the AM cells. The crossover point is likely to be even higher if the required manipulations are more extensive than a simple extreme-value search. We see from Figure 8 that beyond this initial region, the speedup attained by the adaptive method is quite respectable, ranging from about 5 to the maximum of 12.

5.2. Search time variability and overhead

One issue about the performance is that the proposed algorithms may change the deterministic number of cycles in a worst-case bit-sequential search algorithm (e.g. 12 cycles with $w = 12$) into a non-deterministic one that may even change from one run to the next. However, this presents no problem as most algorithms already contain conditional computations and other constructs that make their running times data-dependent. Furthermore, even the bit-sequential search may be provided with an early termination test that stops the search if there is only one responder. Thus, the non-determinism in running time is not a serious issue.

In real-time applications with hard deadlines, scheduling of tasks must be based on their worst-case running times. Our proposed algorithms have worst-case time complexities that are slightly higher than that of the naive bit-sequential search. Thus, one may wish to avoid using these algorithms in the context of hard real-time systems that lack the needed slack or to make the group size g an adjustable parameter that can be reduced to 1 when the worst-case running time is to be minimized. Having said this, we note that the slight additional investment in hardware resources (larger number of processors or faster technology) to accommodate the slightly higher worst-case running times of our algorithms is more than offset by the substantial processing power that is saved on the average. This freed processing power can be applied to the handling of background tasks, more extensive error detection, diagnostic tests, fault tolerance schemes based on task replication and the like, leading to higher overall cost-effectiveness and reliability.

The process by which the optimal group size is selected in each step deserves some elaboration. As discussed in section 4, a greedy or some other suitable analytic/experimental strategy can be used to determine the optimal group size. This strategy provides the optimal group size as a function of x, the number of remaining candidates. Since the group size is always a fairly small integer, the staircase-like function $g^{opt}(x)$ can be precomputed and stored in a small table for use by the adaptive search algorithm. With this approach, the control overhead associated with the adaptive algorithm becomes negligible.

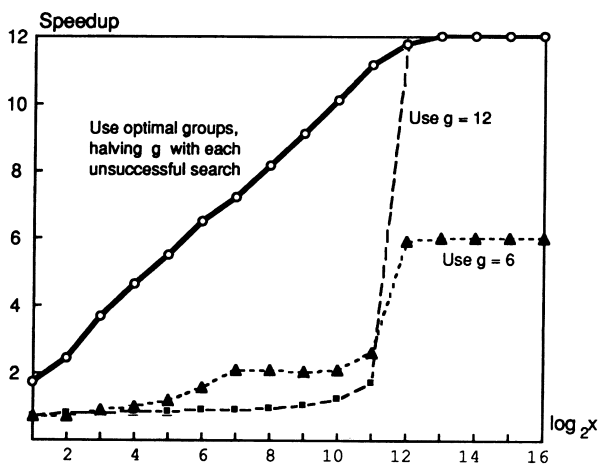

FIGURE 8. Speedup attained by three max-finding algorithms with block searches relative to a 12-cycle bit-sequential search for a field width of $w = 12$ bits.

6. GENERAL SELECTION AND OTHER PROBLEMS

6.1. 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 ($0 \leqslant i < k$) starts with x candidates among which the rth largest must be identified ($r \leqslant x$; initially, $x = m$, $r = j$). A search for 1 is conducted in the ith most significant bit of the field. Let there be y responders. If $y \geqslant r$, then the search continues among the responders for the rth 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 time 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 is directly derivable from the above description:

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

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

6.2. Selection with block searches

Here is one way to speed up the algorithm. 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 bit 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 since the rth largest value is likely to have the all-ones pattern in these g bits. 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, fewer than r responders are produced, then we repeat from bit position i, using single-bit searches within the current g-bit group or, more generally, with a smaller group size $h(g)$, in a manner similar to that discussed in Subsections 4.1 and 4.2.

6.3. 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 the expected search time for the immediately following bit, the quantity to be minimized can be written as

$$s_{g,x} = 1/g + \left[\sum_{y=0}^{r-1} \binom{x}{y} 2^{-gy} (1 - 2^{-g})^{x-y} \right] s_{h(g),x}$$

$$\text{with } s_{1,x} = 1 \tag{10}$$

where $1/g$ is the per-bit contribution of the initial g-bit block search and the summation term is the probability of having fewer than r responders, thus necessitating stepping down to the next lower block of size $h(g)$.

The algorithm discussed in Subsection 4.2 is a special case of the above algorithm with $r = 1$. Minimum finding can be viewed as another special case with $r = x$. It is instructive to analyze the behavior of the algorithm 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 $y \geqslant \lfloor x/2 \rfloor$, then the rth largest value among the responders is sought starting at the next bit position. 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 conducted to relax several bits at once. If, on the other hand, $y < \lfloor x/2 \rfloor$, then the $(r-y)$th largest value among the non-responding candidates must be identified. Again, $r - y$ is likely to be fairly small, leading to a large optimal group size as in the case of maximum finding.

6.4. Application to other problems

Speedup techniques similar to those discussed previously are applicable to other searches on fully parallel AMs. Consider, for example, a relational search used to identify values in a specified field that are greater than a given comparand. The bit-sequential algorithm for this function consists of scanning the field from the most significant to the least significant end. The set of active words, or candidates, at a particular stage consists of all the words that are equal to the comparand up to that bit position. A comparand bit of 1 refines the candidate set by removing those candidates that have a 0 in the corresponding bit position. A comparand bit of 0 causes words that have a 1 in the corresponding bit position to be marked as 'greater' and to become inactive for subsequent steps. Clearly, when a string of g consecutive 1s or 0s is encountered in the comparand, a single g-bit search can be used to refine the candidate set or to identify the subset of the candidates that must be marked as 'greater'. The speedup obtained here is modest since the expected length of strings of consecutive 1s or 0s in a random binary sequence is $1/2 + 2/4 + 3/8 + 4/16 + \ldots + i/2^i + \ldots \approx 2$.

Other searches for which these speedup techniques are applicable include identifying the j largest values (rather than the jth largest, as discussed in the preceding subsections) 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. In the latter case, the speedup is likely to be significant when the range (interval) of interest if fairly narrow.

7. 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 (maximum and minimum finding) and rank-based selection algorithms. The improvement results from the ability to relax several bits of the field under consideration at once. Analyses were offered for both fixed-size and fully adaptive groups and the derivation of optimal group size was discussed in each case. Results for fixed-size group length are of particular interest for byte-serial AMs as well. We have also pointed out the applicability of such speedup techniques to other search functions of interest in several AM applications.

The results presented in this paper can be enhanced and extended in several different directions. These include more extensive experimental verification of the derived time complexities and optimality results through simulation. Also of interest are refinements of some of the analytical results in order 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 and results presented in this paper are not only relevant to fully parallel systems but may find applications in bit- and byte-serial systems as well. For example, designers of bit-serial AM systems typically provide multiple-bit searches as part of the basic instruction set, with built-in hardware or microprogrammed 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 strategies and optimality results can be derived, although optimal group sizes and the resulting speedups are likely to be much smaller compared with our results here.

ACKNOWLEDGEMENTS

The research reported in this paper was supported in part by the US National Science Foundation, grant MIP-9001618. The assistance of Dr Ching Yu Hung in performing the required numerical computations and production of Figures 2–5 is gratefully acknowledged. The author is also indebted to Mr Ding-Ming Kwai for designing and running the simulations needed to produce the speedup results shown in Figure 8.

REFERENCES

[1] Zuse, K. (1986) *Der Computer-Mein Lebenswerk.* Springer-Verlag, Berlin, p. 77.

[2] Bush, V. (1945) As we may think. *Atlantic Monthly*, **176,** July; reprinted in (1988) *Computer Bulletin*, **4**, No. 7, 35–40.

[3] Slade, A. and McMahon, H. O. (1956) A cryotron catalog memory system. *Proc. Eastern Joint Computer Conf.*, pp. 115–120. American Institute of Electrical Engineers, NY.

[4] Chisvin, L. and Duckworth, R. J. (1989) Content-addressable and associative memory: alternatives to the ubiquitous RAM. *Computer*, **22**, No. 7, 51–64

[5] Foster, C. (1976) *Content Addressable Parallel Processors.* Van Nostrand Reinhold, New York.

[6] Grosspietsch, K. E. (guest ed.) (1992) Special Issue on Associative Processors and Memories (in 2 parts). *IEEE Micro*, **12**, nos 6 and 12.

[7] Hanlon, A. G. (1966) Content-addressable and associative memory systems: a survey. *IEEE Transactions on Electronic Computers*, **15**, 509–521.

[8] Kohonen, T. (1987) *Content-Addressable Memories* (2nd edn). Springer-Verlag, Berlin.

[9] Krikelis, A. and Weems, C. C. (eds) (1994) Special Issue on Associative Processing and Processors. *Computer*, **27**, 12–72.

[10] Parhami, B. (1973) Associative memories and processors: an overview and selected bibliography. *Proc. IEEE*, **61**, 722–730.

[11] Parhami, B. (1990) Massively parallel search processors: history and modern trends. *Proc. 4th Int. Parallel Processing Symp.*, Fullerton, CA, pp. 91–104.

[12] Yau, S. S. and Fung, H. S. (1977) Associative processor architecture—a survey. *Computing Surveys*, **9**, 3–27.

[13] Batcher, K. E. (1980) Design of a massively parallel processor. *IEEE Trans. Comp.*, **29**, 836–840.

[14] Hillis, W. D. (1985) *The Connection Machine.* The MIT Press, Cambridge, MA.

[15] Falkoff, A. D. (1962) Algorithms for parallel search memories. *J. ACM*, **9**, 488–511.

[16] Estrin, G. and Fuller, R. H. (1963) Algorithms for content-addressable memories. *Proc. IEEE Pacific Computer Conf.*, March, pp. 118–130. IEEE.

[17] Davis, W. A. and Lee, D. -L. (1986) Fast search algorithms for associative memories. *IEEE Trans. Comp.*, **35**, 456–461.

[18] Feng, T. -Y. (1970) Search algorithms for associative memories. *Proc. Princeton Conf. on Information Sciences and Systems*, pp. 422–426. Princeton University Press, Princeton, NJ.

[19] Feng, T.-Y. (1982) On the development of bit-sequential searches. *Proc. Int. Computing Symp.*, pp. 760–769.

[20] Feng, T.-Y. (1990) Search algorithms for bis-sequential machines. *J. Parallel Distrib. Comput.*, **8**, 1–9, 1990.

[21] Lee, D. -L. and Davis, W. A. (1988) An $O(n+k)$ algorithm for ordered retrieval from an associative memory. *IEEE Trans. Comp.*, **37**, 368–371.

[22] Scherson, I. D. and Ruhman, S. (1988) Multi-operand arithmetic in a partitioned associative architecture. *J. Parallel Distrib. Comp.*, **5**, 655–668.

[23] Sips, H. J. (1984) Bit-sequential arithmetic for parallel processors. *IEEE Trans. Comp.*, **33**, 7–20.

[24] Jalaleddine, S. M. S. and Johnson, L. G. (1992) Associative IC memories with relational search and nearest match capabilities. *IEEE J. Solid-State Circuits*, **27**, 892–900.

[25] Kapralski, A. (1989) The maximum and minimum selector SELRAM and its application for developing fast sorting machines. *IEEE Trans. Comp.*, **38**, 1572–1577.

[26] Parhami, B. (1990) Systolic associative memories. *Proc. Int. Conf. on Parallel Processing*, St Charles, IL, Vol. I, pp. 545–548.

[27] Parhami, B. (1992) Architectural tradeoffs in the design of VLSI-based associative memories. *Microproc. Micro-prog.*, **36**, 27–41.

[28] Ramamoorthy, C. V., Turner, J. L. and Wah, B. W. (1978) A design of a fast cellular associative memory for ordered retrieval. *IEEE Trans. Comp.*, **27**, 800–815.

[29] Hord, R. M. (1990) *Parallel Supercomputing in SIMD Architectures.* CRC Press, Boca Raton, FL.

[30] Parhami, B. (1995) Panel assesses SIMD's future. *Computer*, **28**, No. 6, 89–91.

[31] Parhami, B. (1996) Search and data selection algorithms for associative processors. *Associative Processing and Processors.* IEEE Computer Society Press monograph.

[32] Weems, C. C. and Rana, D. (1991) Reconfiguration in the low and intermediate levels of the image understanding

architecture. In Li, H. and Stout, Q. F. (eds), *Reconfigurable Massively Parallel Computers*, pp. 88–105. Prentice-Hall, Engelwood Cliffs, NJ.

[33] Serrano, M. J. and Parhami, B. (1993) Optimal architectures and algorithms for mesh-connected parallel computers with separable row/column buses. *IEEE Trans. Parallel Distrib. Systems*, **4**, 1073–1080.

[34] Kao, T. -W., Horng, S. -J., Wang, Y. -L. and Tsai, H. -R. (1995) Designing efficient parallel algorithms on CRAP. *IEEE Trans. Parallel Distrib. Systems*, **6**, 554–560.