High-Performance Parallel Pipelined Voting Networks

Behrooz Parhami

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

Abstract

Synthesis methods for high-performance generalized bit-voting and word-voting networks are described and the resultant designs are evaluated with respect to speed and cost. Both ordinary and generalized m-out-of-n voting, with arbitrary votes assigned to the inputs, are considered.

0. Introduction

Voting is an important operation in the realization of ultrareliable systems that are based on the multi-channel computation paradigm. Voting is required whether the multiple computation channels consist of redundant hardware units, diverse program modules executed on the same basic hardware, identical hardware and software with diverse data, or any other combination of hardware/ program/data redundancy and/or diversity. Depending on the data volume and the frequency of voting, hardware or software voting schemes may be appropriate.

Hardware voters that have been described in the literature are essentially "bit-voters" that compute a majority function on n input bits. Hardware voting on words and higher-level data objects has traditionally been handled by using independent parallel bit-voters (potentially yielding incorrect results) or feeding the data sequentially through a single unit (suffering from low performance).

Hardware voters can be characterized by two performance indicators. The voting delay or latency is particularly important in real-time systems with hard deadlines and in high-performance systems. When data to be voted on is generated at a high rate, the voter must be able to keep up with the processing speed. In some such cases, the actual voting delay may not be critical but the voter throughput must match or exceed its input data rate.

Regardless of the level of voting (bits or words), we view a voting network as dealing with n input data objects x_i having the associated votes v_i, $i = 1, 2, \ldots, n$, (i.e., n input data-vote pairs $\langle x_i, v_i \rangle$) and producing the output data-vote pair $\langle y, w \rangle$. The input votes v_i may be explicitly represented (variable votes), stored in modifiable storage (adjustable votes), or wired into the voting network (either fixed-uniform or fixed-nonuniform). The output vote w may be implicit or explicit.

1. Bit-Level Voting Networks

An m-out-of-n bit-voter is a circuit with n bit-inputs x_i, $i = 1, 2, \ldots, n$, and a single bit-output y, such that $y = 1$ iff at least m of the n input bits have the value 1. Input votes are assumed equal and the output vote is implicit. Note that our definition is asymmetric with respect to 0 and 1 values in that an output of 0 does not imply and is not implied by the presence of at least m zeros at the inputs. Simple majority bit-voting corresponds to the special case of $m = \lfloor n/2 \rfloor + 1 = \lceil (n+1)/2 \rceil$. This special case has 0/1 symmetry for odd values of n.

An m-out-of-n bit-voter may be constructed as a two-level AND-OR or OR-AND logic circuit. It can be shown that the two-level AND-OR realization is "simpler" than the two-level OR-AND realization (both by gate count and gate-input count measures) iff $m > (n + 1)/2$. For large values of n, it is impractical to realize bit voters as two-level logic circuits. However, multi-level logic circuits can be developed based on these designs. We call such designs "gate-level" realizations.

Weighted bit-voters are more general t-out-of-Σv_i voters, where v_i, $i = 1, 2, \ldots, n$, are the input votes and t is the threshold. The input weights v_i may be fixed, adjustable, or variable. When the input votes are small integers, it is tempting to simply replicate (fan-out) each input by its corresponding weight and use an m-out-of-n bit voter with $m = t$ and $n = \Sigma v_i$. However, this method seldom results in an optimal voting network. In the following subsections, we offer several direct synthesis methods.

1.1. Arithmetic-Based Bit-Voters

In the "arithmetic" approach, the sign of $-t + \Sigma(x_i v_i)$ is computed. The products $x_i v_i$ can be computed by AND gates and then added by standard carry-save technique to yield the final result. If the v_is are fixed, the hardware realization can be optimized in each case by compressing the constant 0s in the binary numbers to be added.

Consider as an example a voter with 6 bit-inputs having fixed associated votes of 2, 2, 2, 1, 1, 1 and the threshold of 5. The arithmetic expression to be evaluated is:

$$-5 + 2x_1 + 2x_2 + 2x_3 + x_4 + x_5 + x_6$$

The multiple-operand binary addition $(1011)_2 + (x_1x_4)_2 + (x_2x_5)_2 + (x_3x_6)_2$ can be performed by 4 full adders and 2 half adders organized in a 4-level circuit. The leftmost 1 can be ignored since it only causes a complementation that cancels the complementation needed for obtaining the resultant output from the sign bit.

Instead of using full adders and half adders, one can use larger building blocks known as parallel counters, and parallel compressors, which convert a number of input bits to a smaller number of output bits while maintaining the arithmetic value being represented.

1.2. Selection-Based Bit-Voters

The design of an m-out-of-n bit-voter is equivalent to selecting the mth largest bit value from among n input bits. The design of sorting and selection networks built from 2-sorter (comparator) cells has received much attention. For our problem, we can either use an n-sorter (descending order) and look at its mth output or take advantage of the generally simpler selection networks. Knuth [2] defines three types of selection networks that accomplish the following, when provided with n inputs:

1. The m largest inputs appear on m given output lines in no particular order.
2. The mth largest value appears on a given output line.
3. The m largest input values appear on m given output lines in sorted order.

Each network requires at least as many 2-sorters as the preceding one. Denoting the number of 2-sorter or comparator cells by $U(m, n)$, $V(m, n)$, and $W(m, n)$ for type-1, type-2, and type-3 selectors above, we have:

$$U(m, n) \leq V(m, n) \leq W(m, n)$$

Because we are dealing with bits, a two-sorter simply consists of a pair of 2-input gates: An OR gate to produce the larger and an AND gate to produce the smaller of the two input values. Clearly type-3 selectors do more than what is required here. Type-2 selectors do exactly what we want. However, for most practical values of m and n, particularly where m is neither too small nor too close to n, a type-1 selector augmented by an AND circuit (that indicates whether all of the m largest values are 1s) is both faster and more economical.

1.3. Decomposition-Based Bit-Voters

Hierarchical decomposition (divide-and-conquer strategy) can be used to facilitate the design of threshold bit-voters, although in general it does not guarantee an optimal network. The basic idea is to divide the inputs into disjoint subsets, enumerate the various combinations through which different subsets can contribute votes in such a way that the voting threshold is matched or exceeded, provide smaller bit-voters to realize these contributions, and finally, use a logic network for combining the results. The decomposition continues until voters with known designs are obtained.

When all input votes are equal, decompositions with almost equal-size subsets tend to work best. When the votes are unequal, the inputs can be divided into subsets each containing equal-vote inputs. Further decomposition will then proceed as before. This tends to simplify the design, although again there is no guarantee that it will yield the best network. There are two ways to proceed with the decomposition: (1) Picking a partitioning scheme and then designing the required combining network, and (2) Selecting a combining network first. In the remainder of this subsection, we outline a decomposition strategy with multiplexers as combining elements.

The design strategy is to start with the highest-vote inputs and take them as control inputs to a multiplexer and then repeat the process for each multiplexer input function until we reach simple residual functions that can be efficiently implemented (the extreme case being single-gate functions such as AND and OR). The reason for proceeding in descending order of votes is that when high-vote inputs are fixed at 1 or 0, relatively simpler residual logic functions of the remaining variables tend to result.

With this approach, an n-input t-out-of-Σv_i bit-voter is built from a multiplexer and two or more smaller voters. In particular if a 2-input multiplexer is used in the first decomposition stage, two (n–1)-input voters, one t-out-of-$(\Sigma v_i - v_{max})$ and the other $(t - v_{max})$-out-of-$(\Sigma v_i - v_{max})$, are required where v_{max} is the largest input vote. In the special case of equal votes, an m-out-of-n bit-voter is constructed from an m-out-of-(n–1) voter, an (m–1)-out-of-(n–1) voter, and a 2-input multiplexer.

Even though the worst-case complexity of such multiplexer-based voter designs appears to be exponential in n, pruning and circuit-sharing is possible in practice. Thus the complexity may be acceptable for particular values of n and m with specific vote assignments. Numerous worked-out examples suggest that the size of the multiplexers used may be just as important as the ordering of the decomposition variables in obtaining efficient designs. Clearly there is ample room for further research in this area.

1.4. Comparison of Approaches

Figure 1 shows the cost of simple majority voter designs based on the gate-level approach, the arithmetic-based approach, selection networks, and 2-input multiplexer decomposition, assuming maximum allowable gate fan-in of 4. Figure 1 indicates that the gate-level or the multiplexer-based approach is best for small values of n whereas selection networks offer the most economical solution for larger values of n. The crossover points will of course change if the assumptions (such as gate fan-in and fan-out limitations) are altered.

Although the arithmetic-based approach appears to be inferior to the one based on selection, it can become quite competitive if the resulting arithmetic circuit is fully optimized to eliminate all redundant circuits (only a small amount of optimization has been done to obtain the data for Figure 1). Note that Figure 1 relates only to simple majority voters. The strength of the arithmetic-based approach shows up in the case of unequal input votes where the selection-based approach is not directly applicable. Similarly, the multiplexer-based approach is not particularly efficient for simple majority voting but may yield good designs for specific values of the parameters in weighted t-out-of-Σv_i voting.

If the voter designs are used with pipelining, the differences in latencies (number of gate levels) are not significant as far as throughput is concerned. However, the number of gate levels does affect the cost due to the requirement for latches between pipeline stages. A general analysis is impractical because the number of logic signals going from one pipeline stage to the next cannot be expressed as a simple function of the relevant parameters. For these reasons, it is advisable to proceed with and check out several designs for any given set of application parameters before a final selection is made.

2. Word-Level Voting Networks

An m-out-of-n word-voter can be defined as a circuit with n word-inputs x_i, $i = 1, 2, \ldots, n$, and a single word-output y, where either at least m of the inputs have the value y or else a separate "lack of quorum" signal is raised. An alternative that we will pursue in the following paragraphs is to produce an output vote w along with a value y having maximum vote, so that comparing w to m yields the desired quorum information and indicates the validity of y if threshold voting is desired. When two or more values have equal votes, any one of them is an acceptable output. Again, a simple majority word-voter corresponds to the case of $m = \lceil (n + 1)/2 \rceil$. We proceed directly to the general case where a vote v_i is associated with the input x_i.

A straightforward approach for t-out-of-Σv_i voting on a set of n k-bit words is to use independent t-out-of-Σv_i bit-voting for each of the k bit positions. However, there are at least two problems with this approach. First, for $t \leq (\Sigma v_i)/2$, independent bit-voting may produce incorrect outputs. A second, less obvious, problem is that the independent bit-voters may indicate a quorum when no quorum actually exists. Analytical results obtained with reasonable assumptions about errors show that problems may arise even with fairly short word lengths (16 bits).

2.1. Three-Phase Word-Voters

Assume that k-bit data values and their associated b-bit votes are provided in parallel on $k + b$ lines per input. The requirements for digit-serial computation, which is more practical due to the need for fewer interconnections and less complex circuits, have been investigated but will not be discussed here due to space limitation. Word-voting networks can be constructed on the basis of the following algorithm in which each phase corresponds to a subnetwork of the voting network.

Algorithm: *Three-Phase Word Voting*

```
Phase 1. Sort the input pairs $\langle x_i, v_i \rangle$ according to $x_i$s.
Phase 2. Combine all pairs with equal data values into a
         single pair by summing their votes.
Phase 3. Select the pair with the largest vote. ■
```

The design of sorting networks (Phase 1) has been studied extensively [1], [2]. Sorting networks can be constructed from simple 2-input, 2-output comparator cells according to several schemes. Each comparator either passes the two inputs directly to the two outputs or switches the input values to make their order consistent with the desired sorted output order. The only modification required for our application is to add logic to the cells for passing or switching the associated votes in tandem with the data values. Although the design of optimal sorting networks is still an open problem (we mean truely optimal and not asymptotically optimal), for small values of n likely to be of practical interest in our voting schemes, optimal or near-optimal n-sorters are known.

The combining subnetwork (Phase 2) essentially performs a partitioned semigroup computation. The operation of addition must be performed in partitions that are delimited by pairs of unequal data values. This can be done by means of a circuit in which n overlapping binary trees are embedded such that the tree rooted on Line i accumulates data from line i and all subsequent lines that carry equal values (all the way to line n in the worst case). Just as sorting networks can be synthesized from 2-sorters, one can construct the required n-combiner from 2-combiners. It is easy to show that the cell count $C_c(n)$ and the delay $D_c(n)$ for an n-combiner are as follows, where $\lg = \log_2$:

$$\begin{aligned} C_c(n) &= (n-1) + (n-2) + (n-4) + \ldots + (n-2^{\lfloor \lg n \rfloor}) \\ &= n\lfloor \lg n \rfloor - (2^{\lfloor \lg n \rfloor + 1} - n - 1) < n \lg n \\ D_c(n) &= 2\lceil \lg n \rceil - 1 \end{aligned}$$

The max-selector subnetwork (Phase 3) is essentially an $\lceil n/2 \rceil$-leaf binary tree of 2-selector cells. The cell count and delay of an n-max-selector are thus:

$$C_m(n) = n - 1 \quad \text{and} \quad D_m(n) = \lceil \lg n \rceil$$

The complexity and delay of the 3-phase voting network is easily obtained by adding the respective parameters of the three phases with suitable weights to account for the unequal costs and delays of the three cell types.

2.2. Two-Phase Word-Voters

Having obtained a design for word-voting networks, it is natural to ask whether the design can be simplified. In this subsection we show that the first two phases of the three-phase voter can be combined by merging the functions of 2-sorter and 2-combiner cells into a slightly more complex 2-voter cell that acts as a 2-sorter when input data values are unequal and as a 2-combiner when they are equal. This will reduce the total number of cells but since the cells are now more complex, an obvious tradeoff is available (see Subsection 2.3). The merging of Phase 1 and Phase 2 is made possible by the following.

Theorem: If in any sorting network, the 2-sorter cells are replaced by 2-voters, the outputs will be identical to those produced by a 2-phase sorter/combiner network. ■

The complexity and delay of the 2-phase voting network is easily obtained by adding the respective parameters of the two phases (see Subsection 2.3).

2.3. Comparison of Approaches

we now proceed to quantify the relative advantages of three-phase versus two-phase word-voter designs. Although we do not claim that either design is universally optimal, the following informal argument suggests that two-phase cellular voters are very close to optimal in terms of the number of cells. Any word-voting network as defined here can be used as a sorter of the v_i values on the input lines if distinct values are provided for the x_i inputs (so that no combining is possible). Thus a voting network is at least as complex as a sorter in terms of the number of cells required. Our two-phase voter design only contains $n-1$ simple 2-selector cells beyond those required for the sort/combine part. Thus, simplification, if possible, will likely occur in the internal design rather than the multiplicity or interconnection structure of cells.

In what follows, we will ignore the complexity and delay associated with the selection phase which is common to three-phase and two-phase designs. Thus the figures presented will provide an indication of the difference rather than the ratio of complexities and delays. This will be sufficient for judging when the three-phase or two-phase design should be preferred over the other. We will take the complexity and delay of a 2-sorter as our units and denote by γ' and γ'' (respectively δ' and δ'') the relative complexities (respectively delays) of the 2-combiner and the 2-voter cells. We further denote by $C_S(n)$ and $D_S(n)$ the number of 2-sorter cells and the number of cell levels needed in an n-sorter network. Then the complexities C' and C'' and delays D' and D'' of three-phase and two-phase designs can be written as:

$$\begin{aligned}
C'(n) &= C_S(n) + \gamma' C_C(n) \\
&= C_S(n) + \gamma'(n\lfloor \lg n\rfloor - 2^{\lfloor \lg n\rfloor+1} + n + 1) \\
C''(n) &= \gamma'' C_S(n) \\
D'(n) &= D_S(n) + \delta' D_C(n) = D_S(n) + \delta'(2\lceil \lg n\rceil - 1) \\
D''(n) &= \delta'' D_S(n)
\end{aligned}$$

Figure 2 depicts γ'' as a function of γ' for different values of n such that $C'(n) - C''(n) = 0$. In the area above each of the lines shown in Figure 2, the three-phase design is better for the corresponding value of n, while the two-phase design is advantageous below the lines. One may observe that as n increases, the lines generally move downward (eventually tendig to $\gamma'' = 1$ for very large n).

The value of $D'(n) - D''(n)$ is positive for $n \le 16$ provided that $\delta'' < 16/9 + 7(\delta' - 1)/9$. Since it is highly unlikely that $\delta'' \ge 16/9 \approx 1.78$, it is safe to say that the 2-phase design always has the edge in terms of delay. However if $\delta' < \delta''$, the 3-phase design will support a higher throughput, given the fact that a 2-selector is simpler than a 2-sorter, a 2-combiner, or a 2-voter.

3. Future Research

Research is now in progress on refining the concepts presented in this paper and on more detailed quantitative comparison of the various bit-voter and word-voter designs. A unified view of various voting schemes (hardware and software) is also being developed in order to systematize research in this area.

References

Due to lack of space, only the main references are listed here. An extensive list may be found in Reference [3].

[1] Cormen, T.H., C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990, Chapter 28, pp. 634-653.

[2] Knuth, D.E., *The Art of Computer Programming -- Vol. 3: Sorting and Searching*, Addison-Wesley, 1973, Section 5.3.4, pp. 220-246.

[3] Parhami, B., "Voting Networks", Being revised for publication in *IEEE Transactions on Reliability*.

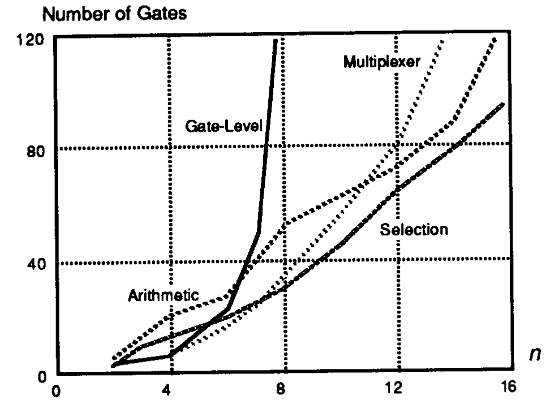

Figure 1. Cost (4-input gate count) of simple majority bit-voters of different designs as a function of input size n.

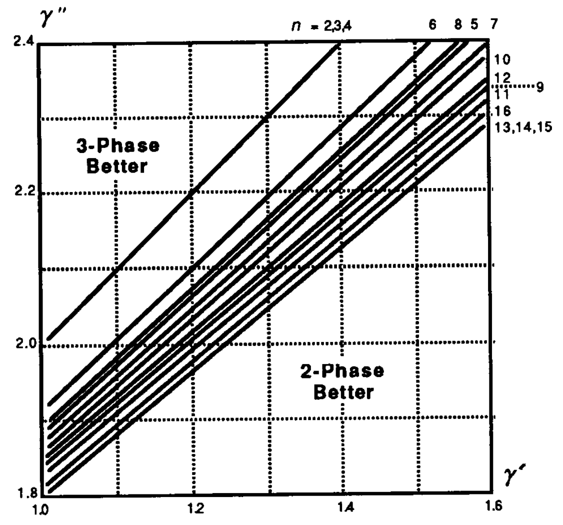

Figure 2. Given efficient realizations for the three cell types involved (i.e., 2-sorters, 2-combiners, and 2-voters), one can determine γ' and γ'' from various technology-dependent indicators of complexity and use this diagram to determine if 3-phase or 2-phase voter design is better.