1. counting for each bit. $O(nw)$.
2. for each bit, count the number of 0s and 1s in the array, which needs $O(\log n)$ bits of space to store if the array has length n. assume word operation is $O(1)$ after preprocessing (the word operations we want: (1) add two w_1-bit numbers, (2) padding, i.e. pad w_1 bits with leading 0 to get w_2 bits), the counting for each bit can be performed in parallel by word operations.
Use divide and conquer to perform counting, let $f(n)$ denote the running time to get the count for each bit when the array has length n. merge the results for two arrays with length $\frac{n}{2}$ needs $O(w \cdot \log n)$ bits, i.e. $O(\log n)$ word operations.
$f(n) = 2f(n/2) + O(\log n)$, i.e. $f(n) = \sum_{i=0}^{\log n} 2^{\log n - i} \cdot i = O(n)$.
total time $O(n)$.

actually if we represent the counters by $O(\log n)$ words, where the j-th bit of the i-th word represent the i-th bit of the j-th counter, we can construct the word operations we want by $O(1)$ basic word operations. The following C++ implementation uses the logarithmic method [1] to reduce the space complexity to $O(\log^2 n)$.

References

[1] Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.