Invited Paper: On the Cost-Optimal Parallel Solution of the Majority Problem

Jie Wu
Center for Networked Computing, Temple University, USA

***Abstract*—The majority problem can be stated as follows: there is a collection of n bank cards that belong to different bank accounts. The only way to "read" these cards is through one or more two-input "equivalence testers". Each tester accepts two cards and outputs "yes" if two input cards correspond to the same bank account or "no" if they do not. The objective of the majority problem is to design a solution that uses the minimum number of tests to determine if more than $n/2$ cards belong to the same bank account and then find these cards. We first review two sequential solutions, including the Boyer-Moore optimal linear solution (i.e., with a time complexity of $O(n)$). Then, we introduce an optimal elimination solution together with a special structure to support a cost-optimal parallel solution that solves the problem in $O(\log n)$ using $O(n/\log n)$ processors (i.e., testers). That is, this solution matches the optimal sequential solution with $O(n)$ operations.**

***Index Terms*—Cost-optimal solutions, data structure, parallel algorithms, speedup.**

I. Introduction

Data processing is increasingly important in the era of big data. New parallel processing paradigms, such as MapReduce [1], can perform fast processing of data in parallel for various problems, including word count and data sorting. However, there are still many data processing problems that cannot easily be mapped to the MapReduce paradigm for an efficient solution without selecting an appropriate data structure together with a carefully-crafted algorithm.

The *majority problem* is such a problem that can be stated as follows: there is a collection of n bank cards (or simply *cards*) that belong to different bank accounts (or *accounts*). The only way to "read" these cards is through one or more two-input "equivalence testers" (or *testers*). Each tester accepts two input cards and outputs "yes" if they correspond to the same account or "no" if they do not. The objective of the majority problem is to design a solution that uses the minimum number of tests to determine if more than $n/2$ cards belong to the same bank account and then find these cards.

The majority problem appears in several sources. In [2], the problem was given as an exercise in seeking a solution using the divide-and-conquer approach with a cost (i.e., the number of operations) of $O(n \log n)$. In the Boyer–Moore solution [3], the problem is defined as a data stream problem that finds the majority of a sequence of elements using linear time (i.e., $O(n)$, the optimal sequential solution) and constant space based on a counting method.

The work was inspired and motivated in part by the recent election process for the house speaker of the US Congress, where the strict majority is needed for the winner among multiple candidates.

This research was supported in part by NSF grants CNS 2214940, CPS 2128378, CNS 2107014, CNS 2150152, CNS 1824440, and CNS 1828363.

In this paper, we focus on the cost-optimal parallel solution that matches the optimal sequential solution in terms of the total number of operations used for the majority problem. We first show that neither the divide-and-conquer solution nor the Boyer-Moore solution can easily be extended for the cost-optimal parallel solution. We then propose a linear elimination solution and extend the solution to a cost-optimal parallel solution using a special structure.

The remainder of the paper is organized as follows: Section 2 reviews two known sequential solutions and proposes a new linear sequential solution based on elimination. Section 3 presents a special structure that supports the cost-optimal parallel extension of the elimination solution. Section 4 concludes the paper with two open problems.

II. Sequential Solutions

We first discuss three sequential solutions (i.e., solutions that uses one processor or tester). All solutions are based on the following approach: a candidate card and its corresponding account are first identified such that if a majority exists, this account is the one. A false positive case will occur if a majority does not exist. In all solutions, two phases are applied. The candidate account (or simply *candidate* or *seed*) is first identified in phase 1, followed by validation of the candidate with all other cards in phase 2. Note that a linear solution is the best possible solution, as cards can correspond to distinct accounts; therefore, these cards need to be examined individually at least once.

A. Divide-and-Conquer Solution

The divide-and-conquer solution recursively divides the given sequence of cards into two equal halves: a left subsequence and a right subsequence. Suppose one seed l is found in the left subsequence and another one r is found in the right subsequence. During the conquer phase, compare l (r) with each element of the right (left) subsequence. A seed will survive if it is the majority of the combined left and right subsequences. Note that the combined subsequence may or may not have a seed anymore. Finally, if a majority exists in the original sequence, it must be the last top-level survivor. The accumulative complexity is $O(n)$ at each level. With $O(\log n)$ levels in the recursion, the overall complexity is $O(n \log n)$.

B. Boyer-Moore Counting Solution

The Boyer-Moore solution is based on a special counting method with one counter and one data storage unit. Initially,

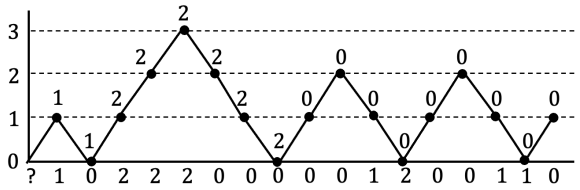


Fig. 1: The Boyer-Moore counting method.

the counter is set to zero without any card in the storage. Cards are examined in sequence in phase 1 to identify a seed. When a new card is examined with the counter set to zero, the card is stored and the counter is set to 1. When a new card is examined with the counter set to a non-zero number, the counter is incremented by 1 if there is a match; otherwise, the counter is decremented by 1. The final stored card after the examination is the seed that will be compared to all other cards once more in phase 2 to validate the seed.

Fig. 1 shows an example of card string 10222000001200110 with three distinct accounts: 0 (majority), 1, and 2. The x-axis shows the card sequence from left to right while the y-axis lists the value of the counter after examining each card. The numbers associated with small dots are the stored elements. Initially, the storage is empty (represented by ?). The last stored card is 0, a seed, which is the majority after the validation process with all other cards in phase 2. Suppose the string is 121212200; the last stored card is 0, which is not the majority after validation in phase 2. The complexity of this solution is $O(n)$, as both phases need a linear scan of cards.

C. Proposed Eliminating Solution

The proposed eliminating method also scans the given card string from left to right as shown in Fig. 2. However, a special bookkeeping process is introduced. Elements in the original card string are *pair-wise* examined from left to right using the equivalence tester. If they are the same, the pair as a whole is "promoted" (slanted lines in Fig. 2) to the next level; otherwise, the pair is considered "eliminated" (horizontal lines). When n is odd, the last card at each level is not paired with any other, and hence, it has "survived". This process repeats level by level until there is no further promotion or elimination.

Fig. 2 shows the process of promotion and elimination. For $i > 0$, a *compound* element at level i includes 2^i *basic* elements at level 0 that have the same account. In Fig. 2, the y-axis shows the element size at each level. Level 1 (labelled as 2^1) in the y-axis has five compound elements (from left to right): 2, 0, 0, 0, 1, each containing two basic elements. The first two elements are eliminated, and the next two elements, 0 and 0, are further promoted to a level 2 element 0, which corresponds to four 0's. The last element, 1, at level 1 has survived. We argue that the *top-level survivor* is the seed. That is, if there is a majority, this seed is the one. The following properties ensure the correctness and linearity of the solution.

Theorem 1: *If the given set of cards has a majority account, that account must be the top-level survivor after elimination.*

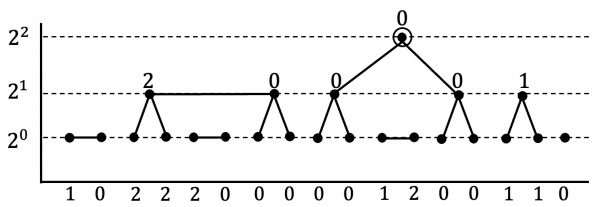


Fig. 2: The proposed elimination method.

Proof: By the nature of the elimination process, two equal subsets of cards that correspond to two different accounts are eliminated. Therefore, at least one copy of the majority will survive. Based on the definition, each level has at most one survivor. Since the size of the survivor at level i is 2^i, the size of the top-level survivor is greater than the size of all other survivors combined; because $2^i > 2^{i-1} + ... + 2^1 + 2^0$. Therefore, the top-level survivor must be the card that corresponds to the majority account if one exists. □

In Fig. 2, the survivors at levels 2, 1, and 0 are 0, 1, and 0, respectively, with double-circled survivor 0 at level 2 being the top. In Fig. 3, double-circled survivor 0 at level 1 is the top, as all elements at level 2 are eliminated.

Theorem 2: *The elimination solution is linear, i.e.,* $O(n)$.

Proof: Because the frequency of elements reduces by at least half while cards are promoted to a higher level, the total number of newly generated elements is less than n. Therefore, there are no more than n card pairs to be examined. □

Combining Theorems 1 and 2 with the linear majority validation process in phase 2, we have the following result.

Corollary: *The elimination solution solves the majority problem in linear time, i.e.,* $O(n)$.

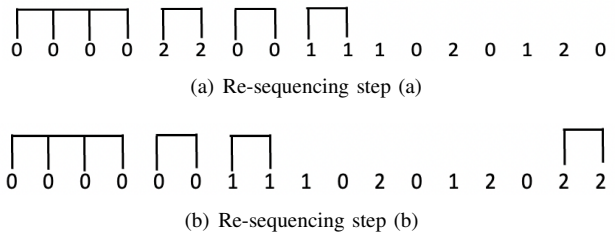
III. Parallel Solutions

The divide-and-conquer solution is parallel in nature with a run time of $O(\log n)$ using $O(n)$ processors (i.e., testers). It is not cost-optimal (see below), as the best known sequential algorithm in terms of cost is $O(n)$. In fact, there are no obvious cost-optimal parallel extensions for divide-and-conquer and Boyer-Moore solutions. The elimination solution can be naturally parallelized, while maintaining cost-optimality.

A. Basic Metrics

The cost of a parallel algorithm is the product of its run time and the number of processors used p. A parallel algorithm is *cost-optimal* when its cost matches the run time of the best known sequential algorithm. The speedup S offered by a parallel algorithm is the ratio of the run time of the best known sequential algorithm to that of the parallel one. Its efficiency E is the ratio of the speedup to the number of processors used. Clearly, the cost-optimal parallel algorithm has speedup p and efficiency 1. A parallel algorithm is cost-optimal in a strong sense if it is both cost-optimal and the fastest.

The Parallel RAM (PRAM) [4] is a natural generalization of the RAM (Random-Access Machine) model for sequential algorithms. We assume the model of EREW (exclusive read



(a) Re-sequencing step (a)

(b) Re-sequencing step (b)

Fig. 3: Re-sequencing of Fig. 2 example.

and exclusive write), which is the basic one for PRAM, without additional assumptions on read/write.

B. Challenges

Compared to sequential solutions, the challenge of parallel solutions lies in keeping all processors "busy" to maintain high efficiency. In the elimination solution, only one seed is identified after phase 1. Multiple seeds need to be quickly identified using multiple testers before a parallel validation can start. Parallel solutions are classified into two types: one in which the seed can be copied and the other one in which the seed cannot be copied.

C. Parallel Elimination Solution with Copying

The parallel elimination solution with copying uses $O(\log n)$ time (see below). The maximum number of processors that can be deployed while keeping the efficiency $E = 1$ to ensure a linear cost of $O(n)$ is $p = O(n/\log n)$.

The candidate selection can easily be parallelized for the elimination solution as we move from lower levels to higher levels using $p = O(n/\log n)$ processors by equally dividing elements at each level into p subsequences. The top-level survivor can be elected among $\log n$ levels in $O(\log n)$[1]. The time complexity for this stage is $O(\log n)$.

We can emulate CREW (concurrent read and exclusive write) using EREW with a $O(\log p)$-round of seed replication to generate p seeds with $p \leq n$, assuming each processor now has a new function that copies the account of the seed to another (new) card. In this way, seeds can be doubled after each round. Each processor will use a distinct seed, i.e., different cards but the same account, to sequentially validate a subsequence of length $O(\log n)$. Overall, the time complexity is $O(\log n)$. We also conjecture that $O(\log n)$ is the fastest cost-optimal algorithm.

D. Parallel Elimination Solution without Copying

We consider here a harder problem where a seed cannot be copied. In our case, cards cannot be replicated to take the advantage of EREW emulation of CREW. The main difficulty lies in quickly (no longer than $O(\log n)$ time) finding p copies of the seed; in order for p processors to perform parallel validations later. This process is called *parallel exploration*.

In the ideal case, it takes $\log p$ phases to find p seeds that correspond to the same account (if one exists), starting from one seed, similar to the copying method. However, if the test is a mismatch, no new seed will be generated. The challenge is how to "select" cards that have high probability of generating matching results with the initial seed. The proposed approach uses a special structure that rearranges the card sequence to ensure a high success probability of matching quickly if the seed is indeed the majority.

[1]Assume each tester can select the min/max address among $O(\log n)$ local elements in $O(\log n)$. Also, a comm. step is no more than a comp. step.

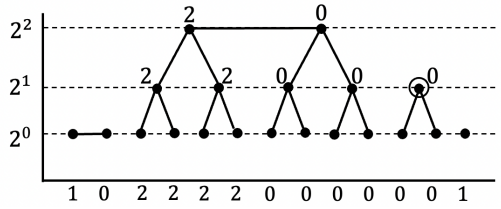


Fig. 4: An example of the top-level survivor not at the top-level of the elimination method.

Card re-sequencing:

(a) Rearrange the sequence from the top level down to the lowest level, and within a level, going from left to right.
(b) Move the first non-seed (compound) element of the card sequence after step (a) to the end of the sequence.

When a compound element at a particular level is selected, all its constituents (i.e., basic elements) at level 0 are selected. Figs. 3 (a) and (b) show the new card sequences of the original card sequence in Fig. 2 after card re-sequencing step (a) and step (b), respectively. In Fig. 4, the new sequence after step (a) is 2222000000101, compared to the original sequence 1022220000001. Step (b) of card re-sequencing moves compound element 2 to the end to generate: 0000001012222.

The re-sequencing, i.e., address remapping, can be done in parallel in $O(\log n)$ through two-level pipelines among inter- and intra-level indexing using $p = O(n/\log n)$ processors, assuming each processor has the basic counting/adding capability. The first non-seed element can be identified through parallel exploration, also in $O(\log n)$. Details will not be elaborated here due to the space limitation.

The parallel exploration of seeds follows the prefix of the new card sequence, called *prefix subsequence.* That is, the exploration starts from the left to the right of the new card sequence in parallel, but in a non-overlapped fashion.

Theorem 3: *In the new card sequence, the majority has at least half in any prefix subsequence.*

The proof of Theorem 3 is shown in the appendix. Based on Theorem 3, the seed always maintains the majority in any prefix subsequence if it is indeed the majority. That is, less than half of the testers will generate a mismatch with the seed. After $\log p + 1$ rounds of parallel exploration, once p or more seeds are identified, we can start the parallel validation process. Note that this seed may still be invalidated later if it is not the majority. Also, if there are fewer than p cards that match the seed, there will be no majority, avoiding the validation phase. Each tester needs to know its relative position with the help of an *atomic counter* that issues sequence numbers to ensure non-overlapped exploration regions among p testers.

Parallel exploration: We use existing seeds (initially, only one seed) to perform parallel searching for new seeds in rounds.

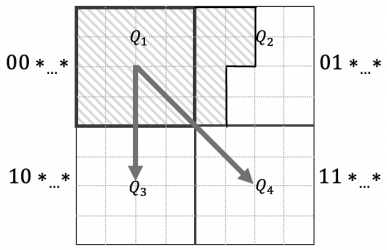


Fig. 5: Parallel exploration of seeds.

Each round will double the search space. When a majority exists, the first element (with address 0) after re-sequencing must match the seed; otherwise, the seed will have two more than all others at the top level – causing a further promotion, which is a contradiction. A mismatch with the first element will indicate a no-majority case. For simplicity, we assume that address 0 is the seed. Also, seed addresses that are based on the atomic counter and card addresses in the new sequence are both represented by binary numbers starting from 0.

- Round 1: seed 00 (i.e., 0) checks cards 01, 10, and 11.
- Round 2: seed 000 (same as seed 00) checks cards 100 and 110; seed 001 (same as seed 01) checks 101 and 111.
- Round 3: seed 0000 (same as seed 000 and seed 00) checks cards 1000 and 1100; 0001 (same as seed 001 and seed 01) checks 1001 and 1101; 0010 checks 1010 and 1110; 0011 checks 1011 and 1111.
- ...
- Round i: For a seed from 0 to $2^{i-1}-1$, it is represented by an $(i+1)$-bit binary 00 *...*, where *, 0 or 1, is a wild card. This seed checks cards 01*...* and 11*...*.

In general, starting round 2, we use the first quarter Q_1: 00*...* seeds to check the corresponding third quarter Q_3: 10*...* and fourth quarter Q_4: 11*...* as shown in Fig. 5. Note that the second quarter Q_2: 01*...* has been checked in the previous round. Seeds in Q_2 (in the shaded area of Fig. 5) are not used to ensure that all seeds examine non-overlapped regions. Each newly detected seed will obtain its seed ID via the atomic counter. Based on Theorem 3, the number of seeds must have at least two first quarters of the current space so that we can double the search space in the next round.

Parallel validation: Once p seeds are identified, validation can easily be carried out by evenly dividing the sequence into $p = O(n/\log n)$ subsequences of length $O(\log n)$.

Overall, the parallel elimination solution without copying is cost-optimal that uses $p = O(n/\log n)$ processors (i.e., testers) and solves the majority problem in $O(\log n)$ time.

IV. Conclusion

The paper re-examines a classic problem that can be solved in a cost-optimal way. The proposed elimination method together with a special structure for parallelism is an addition to two classic sequential solutions that cannot easily be parallelized in a cost-optimal way. The readers may find the card reading method through equivalence testers to be overly restrictive. This restrictive validation method may find potential applications where privacy is a concern. Our cost-optimal solution assumes that the atomic counter runs much faster than the equivalence tester. One open question is the existence of a cost-optimal $O(\log n)$-speed parallel solution without copying and without using an atomic counter. We also need to validate our conjecture that the proposed parallel solution is cost-optimal in a strong sense, i.e., it is the fastest possible solution with a linear cost.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[2] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education, 2006.
[3] R. S. Boyer and J. S. Moore, "Mjrty—a fast majority vote algorithm," in *Automated Reasoning*. Springer, 1991, pp. 105–117.
[4] J. JaJa, *An introduction to parallel algorithms*. Addison-Wesley, 1992.

Appendix

To Theorem 3, we first introduce two lemmas:

Lemma 3a: *If a given card sequence has a majority element, the majority element has at least half the number of elements at each level of the elimination solution.*

Proof: We prove by induction. Level 0 clearly holds. Assume that the lemma holds for level i; pairs that are not promoted to level $i+1$ are pair-wise eliminated. When the number is odd, the last element is not paired. Clearly, at least half of the pairs promoted to level $i+1$ correspond to the majority. □

The next lemma shows a way of expressing a prefix subsequence:

Lemma 3b: *All elements in a prefix subsequence (after step (a) of card re-sequencing) that contains elements from the top level down to level i can be expressed by elements at level i.*

Proof: By the elimination method, all elements at level $i+1$ are "constructed" from level i through promotions. Through iterations, all elements above level i are constructed from level i by treating all elements in level i as basic elements. □

Using Fig. 2 and Fig. 3 (a) as an example, Lemma 3b clearly holds for level 0 for the whole sequence as a prefix subsequence. From the top-level to level 1, the prefix subsequence 0000220011 is instructed by all elements at level 1 in Fig. 2: 2 (for 22), 0 (for 00), 0 (for 00), 0 (for 00), and 1 (for 11).

Proof of Theorem 3: We can prove by contradiction. Suppose the last element of a prefix subsequence is at level i. If the majority account (i.e., seed) does not have half of the prefix subsequence, based on step (b) of the new sequencing, at least one non-seed element at level i is removed and placed last in the new sequence. That is, among all elements at level i in the prefix subsequence, the number of non-seed elements is at least two more than the number of seeds. On the other hand, based on Lemmas 3a and 3b, the seed has at least half of the elements at level i in the elimination method. Among elements at level i that are not selected in the prefix subsequence, seeds will have at least two more elements than non-seeds. In this case, at least two seeds will be adjacent – those will be promoted to a higher level. This promoted element should appear in the prefix subsequence, which is a contradiction. □