Distributed Algorithms

in Synchronous Broadcasting Networks

CUCS-180-85

Zvi Galil, Gad M. Landau, & Mordechai M. Yung

Zvi Galil[1]

Department of Computer Science, Tel-Aviv University, Tel-Aviv, 69978

and Department of Computer Science, Columbia University, New York, NY 10027

Gad M. Landau

Department of Computer Science, Tel-Aviv University, Tel-Aviv, 69978

Mordechai M. Yung[2]

Department of Computer Science, Columbia University, New York, NY 10027

[1]Supported in part by NSF grants MCS-8303139 and DCR-8511713

[2]Supported in part by NSF grant MCS-8303139 and an IBM fellowship

Abstract

In this paper we consider a synchronous broadcasting network, a distributed computation model which represents communication networks that are used extensively in practice. We consider a basic problem of information sharing: the computation of the multiple identification function. That is, given a network of p processors, each of which contains an n-bit string of information, how can every processor compute efficiently the subset of processors which have the same information as itself? The problem was suggested by Yao as a generalization of the two-processor case studied in his classic paper on distributed computing [18].

The naive way to solve this problem takes O(np) communication time, where a time unit is the time to transfer one bit. We present an algorithm which takes advantage of properties of strings and is $O(n \log^2 p + p)$ time. A simulation of sorting networks by the distributed model yields an O(n log p + p) (impractical) algorithm. By applying Yao's probabilistic implementation of the two-processor case to both algorithms we get probabilistic versions (with small error) where n is replaced by log n in the complexity expressions. We also present lower bounds for the problem: an $\Omega(n)$ bound and an $\Omega(p)$ bound are shown.

1. Introduction

The synchronous broadcasting distributed computation model presented here represents existing communication systems (for example, multiple-frequency-hopping radio networks or point-to-multipoint networks). Works in theoretical computing have considered either routing network models [10] [6] [14] or single channel broadcasting networks where either transmission order is prearranged [5], or a resource sharing (Ethernet-like) mechanism is used [7] [17]. A recent work [9] considers the multi-channel case when the network has few channels, but the order of transmission and channel allocation is prearranged as well.

Previous work unjustifiably neglected the network model, defined as follows: The processors in the network are $\{P_1,.....,P_p\}$, the network is fully connected and the communication is via the links. The operation mode is synchronous and the communication operations are *transmission*, in which the processor broadcasts its message, and *reception*, in which the processor chooses a processor to listen to and gets one bit during a single unit of time. This choice is made dynamically. This work is the first one known to the authors which considers this model in a theoretical context. Section 2 describes the model in detail.

One of the central class of problems in a distributed environment is information recognition and identification in a global context [3]. In such an environment each processor has its own local information, and the basic problem is how to let the processors recognize, share and process information which originally belongs to other processors. We find interesting versions of the problem in practice: in distributed sensor networks (abbreviated DSN) [19] and in distributed operating systems [16] [12]. This problem is also modeled in different theoretical contexts [1] [3] [18] [11] [6] [14].

To demonstrate problem solving capabilities of the synchronous broadcasting network, we investigate the following problem: Given a network of p processors, each with an n-bit string, each processor wants to know the subset (*class*) of processors which have the same information as itself. This problem appears in various situations. For example, the string can be information observed in a DSN where the system tries to compare signals received by different remote sensors and decide their credibility.

The problem is a generalization of the *identification function* computation: Two processors, one with a string x and the other with a string y, wish to compute the function $f(x,y)= \delta_{x,y}$.

($\delta_{x,y}$=1 iff x=y, and 0 otherwise.) Yao [18] defined the function, proved a lower bound on the communication complexity of this problem, and showed that at least n bits have to be transmitted when we allow deterministic two-way communication. He also gave a probabilistic protocol (with small error) in which only O(log n) bits are exchanged. Finally, he suggested a generalization to three processors in a very special case where two of them send information to the third one, and posed an open question: what is the complexity of the problem where more than two processors are involved? Here we examine this question in the model presented above.

The immediate solution to the network of p processors gives an n(p-1) time algorithm. We design an O(n $\log^2 p$ + p) time distributed algorithm. We use string properties, propose a structured organization of the communication, and design an algorithm which uses only communication operations and comparisons of bits. A second algorithm is given where the distributed system simulates a sorting network. In this algorithm the processors use arithmetic operations. Using the reduction to sorting and incorporating recent results [2] [13], we are able to design an O(n log p + p) time algorithm (with a very large constant multiplying the n log p term). Both algorithms can be transformed into probabilistic ones if a small error can be tolerated. In the probabilistic versions the n term becomes log n in both complexity expressions. Sections 3 and 4 present the algorithms while section 5 describes the probabilistic implementations.

Lower bounds of Ω(n) and Ω(p) to the problem are given in section 6. Our algorithm allows the processors to send messages which include information (i.e. addresses) about other processors. We show that restricting messages to be functions only of the processor's input weakens the model, since any such restricted algorithm requires Ω(np). We then suggest some open problems, the most challenging of which is developing lower bound techniques for broadcasting networks. The synchronous broadcasting model seems to defy all known lower-bound techniques. These bounds are usually based on weaknesses of the model in information transfer, while our model seems to have no such weaknesses.

2. The Synchronous Broadcasting Network Model

The processors in the model (P_1,.....,P_p) are random access memory machines (RAM's) with local memory: without loss of generality we can assume that $p=2^k$. A processor is identified by its name (its index) by all the processors. The network is fully connected, the communication is via the links, and there is no central common memory.

The operation mode is synchronous: in each time unit each processor can perform either a local computation or one of the following communication operations:

1. *transmission*- the processor broadcasts (sends) its bit to all its outgoing links.
2. *reception*- the processor chooses a processor to listen to and gets one bit from it.

Two sub-models are possible, according to communication concurrency. A full-duplex communication in which concurrent transmission and reception are allowed and a half-duplex communication in which each processor can either transmit or receive in a given time unit. In the first sub-model there are no problems of synchronization. In the second one, however, we might need to synchronize operations; if a processor has to receive a bit of information from another processor while the second one is not transmitting, but receiving information itself instead, then the first processor will have to try again. It is apparent that (2 log p) time units of half-duplex communication are sufficient to simulate one step of the full-duplex communication. At time unit 2i-1 the processors whose numbers contain 0 in the i-th position broadcast their bits, while in time unit 2i those with 1 in the i-th position broadcast.

In the rest of the section we show that the model is not sensitive to the inexistence of concurrency in the communication operation. We present a simulation of the full-duplex model by the half-duplex one in which each time unit of the former model is achieved in only six units of the latter one. We call this simulation the Echo Algorithm. The idea is that the parity of the processor partitions the processor set in a manageable way.

The Echo Algorithm:
Each processor P_i has a bit b_i.

begin
1. odd processor : sends its bit.
   even processor 2i : gets b_{2i-1} from P_{2i-1}.

2. odd processor 2i-1 : gets b_{2i} from P_{2i}.
   even processor : sends its bit.

{ *at this point, in each pair of processors* (P_{2i-1}, P_{2i}), *each member knows both* b_{2i-1} *and* b_{2i} }

3. odd processor 2i-1 : sends b_{2i-1}.
   even processor : if it needs b_{2i-1} it gets it from P_{2i-1}.

4. odd processor 2i-1 : sends b_{2i}. { P_{2i-1} *serves as echo for* P_{2i} }
   even processor : if it needs b_{2i} it gets it from P_{2i-1}.

5. odd processor : if it needs b_{2i-1} it gets it from P_{2i-1}.
   even processor 2i : sends b_{2i-1}. { P_{2i} *serves as echo for* P_{2i-1} }

6. odd processor : if it needs b_{2i} it gets it from P_{2i}.
   even processor 2i : sends b_{2i}.

end; { Echo-Algorithm }

The Echo algorithm is a universal compiler which takes care of the synchronization problem and translates algorithms in the full-duplex sub-model into the half-duplex one. This implies that one can design algorithms for the half-duplex sub-model using the stronger full-duplex one.

3. The Multiple Identification Algorithm

First we describe some properties of binary strings used by the algorithm, then we describe the algorithm and prove its properties.

3.1. Relations on Strings

Let $\Sigma = \{0,1\}$ and $x,y \in \Sigma^n$. We denote the bits of the string x: x(1)....x(n).

We use the following notation to describe properties of strings and their prefixes.

Definition 1: $\forall i, 0 \leq i \leq n, x E_i y \Leftrightarrow \forall j, 0 < j \leq i, x(j)=y(j)$.

Notice that trivially $\forall x,y \in \Sigma^n$: $x E_0 y$.

Definition 2: $\forall i, 1 \leq i \leq n, x F_i y \Leftrightarrow (x E_{i-1} y)$ and $(\neg (x E_i y))$. (Notice that $x F_{n+1} y$ means $x=y$).

E_i simply means that the prefix of length i is equal, while F_i means that the prefix of length i-1 is equal and the i-th bit is different, the notation is introduced to simplify the following discussion.

Two simple facts about binary strings are used by the algorithm:

Fact 1: E_i is an equivalence relation and E_{i+1} is a subset of E_i .

Fact 2: $\forall i, 1 \leq i \leq n : ((x_1 F_i y_1)$ and $(x_2 F_i y_2)$ and $(x_1 E_i x_2)) \Rightarrow (y_1 E_i y_2)$.

3.2. The Information Structures in the Processor

In each processor P_v we have the following data structures:

1. The input string which is the array $x_v = x_v(1),.....,x_v(n)$, where $x_v(i) \in \{0,1\}$.
2. An address array $R_v = R_v(1).....R_v(n)$ to store processor addresses. $R_v(i) \in \{0,1\}^{\log p}$. The algorithm will satisfy the property that if $R_v(i)=w$ then $x_v F_i x_w$.
3. An output array $N_v = N_v(1),.....,N_v(p)$. $N_v(i) \in \{0,1\}$, $N_v(i)$ corresponds to processor number i. The array is the result of the computation. It will be shown that at termination $N_v(u)=1$ iff $x_v=x_u$.

3.3. Organization of Communication

Our algorithm is divided into steps. There are k=log p steps, in each one of them we partition the $p=2^k$ processors into clusters. A cluster is a group of r consecutive processors $P_{i+1}, P_{i+2},... P_{i+r}$. In a step processors communicate only with processors in their cluster.

Definition 3: a 2^m-cluster: In step m the clusters have size 2^m and are called 2^m-clusters. They are defined in the obvious way. For $0 \leq m \leq k$ there are 2^{k-m} 2^m-clusters. The j-th 2^m-cluster is $\{P_{(j-1)2^m+1}, ..., P_{j2^m}\}$.

Obviously, each processor is a 2^0-cluster, and there is one p-cluster which contains all the processors. Clusters can be represented by a *cluster tree*. A 2^m-cluster is the father of the two 2^{m-1}-clusters contained in it.

In step m let S be a cluster. Its left and right sons which are now sub-clusters are denoted by S_l and S_r. During the step there is a cluster conference: Each processor P_v in the cluster is aware of its cluster number, its own number within the cluster and its sub-cluster. The goal of the conference is to let $P_v \in S$ collect the information about strings of processors in S. If $P_v \in S_l$ then it knows the information about strings in this sub-cluster from previous steps and it has to get information from S_r. We will describe the information provided by the algorithm and will prove its sufficiency. The arrays R_v and N_v represent the information known to P_v about its cluster. We will show that at the end of step m if P_v and P_w are in the same cluster then $N_v(w)=1$ iff $x_v=x_w$.

3.4. The Algorithm

The algorithm has log p steps. Before the algorithm starts, each processor P_v assigns the following values: For all j, $R_v(j):=\Phi$ where Φ denotes the null processor, and $N_v(v):=1$ while for all j: $j \neq v$, $N_v(j):=0$.

We describe the algorithm for a general processor P_v in cluster $S=(S_l \cup S_r)$ where (without lost of generality) $P_v \in S_l$. In each step the processor chooses a processor belonging to the other sub-cluster (S_r in our case) from which it gets the information about this sub-cluster. We call this processor the partner of P_v, denoted by P_w. During the step the processor may change its partners. Sometimes during the step the processor stops working for the rest of the step. Each processor P_v has a local Boolean variable named $Work_v$ which is true at a beginning of a step and stays true as long as the processor is working in the step.

ALGORITHM 1;

Each step m (m=1,..,log p) has three parts:

Part 1. initialization:

P_v chooses a partner P_w (without loss of generality w is $2^{m-1}+v$) and $Work_v:=true$.

Part 2. scanning:

During the step, P_v scans the string x_v from left to right in n time units. Let P_w be its partner in time unit i. An invariant property of partners' strings is $x_v(1),...,x_v(i-1)=x_w(1),....,x_w(i-1)$ i.e. $x_v E_{i-1} x_w$. P_v

receives $x_w(i)$ and $R_w(i)=u$ from its partner P_w. If $u \neq \Phi$ then since $(R_w(i)=u) \Rightarrow (x_w F_i x_u)$, P_v concludes that $x_u(1),....,x_u(i-1)=x_w(1),....,x_w(i-1)$ i.e. $x_u E_{i-1} x_v$. If $x_v(i) \neq x_w(i)$ then $x_w F_i x_v$ and since $R_w(i)=u$ it knows (by fact 2) that $x_u(1),...,x_u(i)=x_v(1),...,x_v(i)$ i.e. $x_u E_i x_v$. Therefore P_v sets $R_v(i):=w$ and in the next time unit u becomes the partner of v (w:=u). By changing partners P_v can always scan the next bit in the string during the next time unit. If, on the other hand, $x_v(i)= x_w(i)$, P_v does not change partner and copies $R_w(i)=u$ to $R_v(i)$. (This copying is actually needed only if $R_v(i)=\Phi$). If there is a mismatch (i.e. $x_v(i) \neq x_w(i)$) and $u=\Phi$ then P_v can stop working in the current step since there are no members of its class in the other sub-cluster ($Work_v$:=false). The following procedure describes the processor's task in this part.

The Procedure 'Scan':

```
1.  for j:= 1 to n do
    begin { time unit j }
2.      If Work_v then
                    begin
2.a                     Send ( x_v(j) , R_v(j) ) ;
2.b                     Get  from P_w ( x_w(j) , R_w(j) ) ;
3.                      Call 'Check'; {whether there is a match; see below}
                    end
4.                  else { Work_v=false } wait a time unit;
    end; { time unit j }
```

The procedure 'Check' summarizes the local operations in a time unit:

The procedure 'Check':

```
 begin { time unit j  ( P_v got x_w(j) and R_w(j) ) }
1.    If x_v(j) = x_w(j) { match }
2.      then   If (R_w(j) ≠ Φ) then R_v(j):= R_w(j)
        else
          { mismatch x_v ≠ x_w }
3.            If R_w ≠ Φ
4.              then        { change partner, update partner index }
                      begin   R_v(j) := w ;   w := R_w(j) ;  end
5.              else  Work_v := false ;
 end
```

Part 3. union class:

After scanning the string, the processor has to identify processors in the other sub-cluster which belong to its class: A processor which is still working knows that its current partner belongs to its class. It gets a sequence of zeros and ones from its partner. The i-th element of this sequence indicates whether the i-th element of the partner's sub-cluster belongs to the class or not. Here is the procedure which describes the

operation in short:

The procedure 'Union-class':
{ Let LA be the smallest address in the processor's cluster
Let RA be the smallest address in the partner's cluster }

```
 begin { step m }
1.  for i := 1 to 2^(m-1) do
    if Work_v then
                begin
2.a               Send ( N_v(LA-1 + i) )
2.b               Receive from partner ( N_v(RA-1 + i) )
                end
 end
```

3.5. Correctness and Complexity of the Algorithm

Consider the beginning of time unit j of step m and any $P_v \in S_l$ whose partner is P_w. The claims below are proved by induction on time (i.e., on the step number and within a step on the unit of time). For convenience we denote by R^m the array R at the end of step m.

Claim 1: (1) $P_w \in S_r$; (2) If $R_v^m(j) = u \neq \Phi$ then $P_u \in S_l \cup S_r$.

Proof:

By induction on time, (1) holds since partner changes maintain it; (2) then is true since R is updated by information from partners and by changes of partners. *QED*

Lemma 1: (1) $x_w E_{j-1}^m x_v$. (2) $\forall i \leq j$: $\{(R_v(i)=u \neq \Phi) \Rightarrow (x_u F_i^m x_v)\}$

Proof:

By induction on time. All decisions in a time unit related to the choice of the next partner, as well as the update of R maintain (1) and (2). For example, assume by induction that at the beginning of time unit j $x_v E_{j-1} x_w$, and consider the case of a mismatch ($x_v(j) \neq x_w(j)$). $R_v^m(j)$ is set to w, and indeed $x_v F_j x_w$, so (2) holds. In the case that $R_w^{m-1}(j)=u$, P_u becomes the new partner in time unit j+1. By fact 2 and the induction $x_v E_j x_u$, so (1) is maintained. *QED*

Lemma 2: If for $P_{v'} \in S_l \cup S_r$ $x_v F_j x_{v'}$ then $R_v^m(j) \neq \Phi$; if in addition $P_{v'} \in S_r$ then P_v is still working at time unit j.

Proof:

Again by induction on time. If $P_{v'} \in S_l$, the lemma follows from the induction assumption, so we assume $P_{v'} \in S_r$.

We first show that P_v is still working at time unit j. Let $i < j$ be the time unit when P_v stopped working, and $P_{w'}$ be the last partner of P_v. By lemma 1, $x_{w'} E_{i-1} x_v$ and by the algorithm $x_{w'}(i) \neq x_v(i)$. Hence $x_{w'} F_i x_v$ and by induction $R_{w'}^{m-1}(i) \neq \Phi$ and P_v will get a new partner -- contradiction. So P_v is still working at step j.

Now recall that P_w is the partner of P_v at time unit j. In case of a match, we have by lemma 1 $x_w E_j x_v$ and (by fact 2) $x_w F_j x_{v'}$. By induction $R_w^{m-1}(j) \neq \Phi$ and as a result $R_v^m(j) \neq \Phi$. In case of a mismatch $R_v(j)$ is set to w. So in both cases $R_v^m(j) \neq \Phi$. *QED*

From lemma 2 if for any $P_v \in S_l$ there is $P_{v'} \in S_r$ with $x_v = x_{v'}$, then P_v is still working at the end of time unit n, i.e., it still has a partner P_w and by lemma 1 $x_w = x_v$. Hence, an induction on the number of step shows the correctness of Union-Class, P_v will get from P_w the processors from his class which are in S_r and at the end of the step will know its class in his cluster.

Theorem 1: The algorithm is correct and its complexity is $O(n \log^2 p + p)$.

Proof: Since at the end of the algorithm all the processors are in the same cluster we conclude:

$\forall v,w: ((x_v = x_w) \Leftrightarrow (N_v(w) = 1))$.

The algorithm consists of $k = \log p$ steps. In each 'Scan' sub-step, there are n time units: each costs $(\log p + 1)$ communication bits. Therefore the total time spent scanning is $O(n \log^2 p)$. The length of 'Union-Class' in a step is the length of the sub-cluster, hence the total time of the unions is $\sum_{i=0}^{k-1} 2^i = O(p)$. *QED*

4. An Algorithm Using Reduction to Sorting Networks

4.1. Simulation of Sorting Network

Constructing sorting networks is one of the most widely studied problems in parallel computation. For a long time the best network to sort N numbers was Batcher's $O(\log^2 N)$-level construction [4]. A recent breakthrough by Ajtai, Komlos and Szemeredi [2] achieved an O(log N)-depth parallel network that sorts N numbers. Their work also provided an O(N)-node, O(log N)-degree network which sorts N inputs in O(log N) time (which is the depth of the network). Leighton [13] further reduced the degree of such a network to a constant. (These new networks are not practical since either the depth or the number of processors has a huge constant factor.) A sorting network is composed of comparison boxes, each with two inputs and two outputs. In our model a box can be simulated by four processors, two of which contain the input strings of the box; the other two processor receive these strings (as output of the comparison). The broadcasting network can simulate a sorting network and sort p k-bit words in O(k log p) time. Each processor can quickly compute the addresses of the other processors involved in each comparison, due to the recursive structure of the sorting network.

4.2. The Algorithm Using Sorting Network Simulation

ALGORITHM 2:

The algorithm has the following 4 parts:

Part 1. sorting:

Each processor concatenates its address v to its input x_v. The processors sort the strings $<x_v,v>$, v=1,..p (notice that v is less significant in the concatenated string). After the sorting, processor P_v receives the string $<x'v'>$. Call all processors which receive the same x' a *group*. As a result of the sorting, the group is a set of consecutive processors, the first group of processors, starting with P_1, contains the addresses (v'-s) of the first class (the one with the smallest x) in increasing order, and so on. The idea is that now the group can compute the result of the computation by communicating only with a local consecutive block of processors.

Part 2. group boundaries:

Each processor performs a search to find the boundaries of its group, that is, the smallest and largest processors which got the same string x'. This can be done using one of the following methods:

- 1. The doubling technique: Each processor P_v broadcasts its string (4 log p) times. Simultaneously, P_v compares its string to the one of the processor P_{v+1}, P_{v+2}, P_{v+4},... and so on until its x' is different from the string x' of P_{v+2^i}, and then by binary search it finds the largest processor with the same string. The smallest processor is found symmetrically.
- 2. The method of searching for boundary indicators: The processor P_v has two Boolean variables called Left and Right. Each processor compares its string x' with those of its immediate left and right neighbors (P_{v-1} and P_{v+1}) and updates Left and Right according to the result of the comparison. Then the processor broadcasts its variables <Left, Right> p times and simultaneously listens to its neighbors which are in its group first to its left neighbor P_{v-1}, then to P_{v-2},.... until it gets an indication that some P_{v-j} is not in the group; then it does the same with its right neighbors.

Part 3. output calculation:

The goal of this part is to enable each processor P_v to calculate the output N' of processor $P_{v'}$ whose input string and address (x'v') were received by P_v in part 1. Therefore, P_v needs to know the addresses received by all processors in its group. First, P_v (except if it is the smallest in its group) gets from P_{v-1} the address (v-1)' received by it in part 1. Then P_v computes the difference of the addresses v'-(v-1)'. Now the i-th processor in the group knows the difference between the addresses of the i-th and the (i-1)-th processors of the class which forms the group. Then one by one and in order the members of the group (except the smallest one) broadcast the differences (using a special symbol to denote end-of-message). Each processor, knowing all the differences, can calculate the addresses of the processors in the class. P_v computes a vector N' by assigning 1 to indices corresponding to addresses of members of the class. (N' is the output vector of processor number v').

Part 4. output distribution:

The goal of this part is for $P_{v'}$ to receive its output from P_v (which calculated N'). Processor P_v concatenates v' and its own address v. The system sorts <v',v> (v is less significant). As a result, processor $P_{v'}$ gets <v',v>, where v is the name of the processor that computed its output. Next $P_{v'}$ receives the output N' from P_v and the algorithm ends.

The algorithm correctness is directly implied by the sorting processes of the network and the searching processes within the groups. The time analysis of the algorithm is as follows: Part 1 takes O((n+log p)log

p) sorting time and part 2 takes O(n log p) using the first method or O(n+p) using the second one. Part 4 takes O(p) time for broadcasting of N', dominating the sorting of the addresses, which takes only $O(\log^2 p)$. The address difference transmissions in part 3 cost O(p), which dominates the time of this part. The total time of the algorithm is therefore O(n log p +p).

The time analysis of part 3 is based on the observation that the sum of the differences transmitted by a group in this part is bounded by p, which is implied by the following simple claim:

Lemma 3: If $a_1, a_2, \ldots, a_k$ are non-negative integers that satisfy $\sum_{i=1}^{k} (a_i) \leq p$, then $\sum_{i=1}^{k} (\lfloor \log a_i \rfloor + 1) = O(p)$.

We remark that a practical implementation of the algorithm, which uses Batcher's network, takes $O(n \log^2 p + p)$ time. This is the same complexity as our first algorithm. Notice that the algorithm using simulation of sorting networks requires that processors perform additions and subtractions, while the first algorithm does not.

5. The Probabilistic Algorithms

Karp and Rabin [8] introduced the idea of *fingerprint function*, which is to choose a random hash function ϕ such that $\phi(x) << x$, and for every collection of strings of a given size there is only a small probability that $x \neq y$ when $\phi(x) = \phi(y)$. Given our set of strings (regarded as a set of binary numbers) we can choose the family of functions to be { x mod q : q prime }, namely, the fingerprints are the residues. The analysis given in [8] shows that the probability of an error is very small even for small q, $q \leq 5 (\log n + \log p)$. Yao used this idea to design a probabilistic two-processor algorithm: the same can be done in the multi-processor case. Notice that we require the processors to perform modular arithmetic operations when they compute the fingerprint.

The Probabilistic scheme is as follows:

- 1. P_1 chooses (probabilistically) a random prime q [8], q of length 20 + log(5(log n + log p)) bits, and broadcasts it.
- 2. Each P_i computes $\phi(x_i) = x_i \bmod q$.
- 3. The processors execute the algorithm (any of the algorithm presented) using $\phi(x_i)$ as the information string instead of the original input.

The complexity of the probabilistic version of algorithm 1 is $O((\log n \log^2 p) + p)$ while the complexity of the probabilistic algorithm which is based on algorithm 2 is $O((\log n \log p) + p)$.

6. Lower Bounds

We introduce here two lower bounds. The two cases are extreme cases where either the number of processors or the length of the strings is constant.

Lemma 4: The multiple identification problem is $\Omega(n)$.

This is proven for the case $p=2$. The proof is Yao's theorem in [18] since, when $p=2$, our model is not stronger than the model in [18].

Lemma 5: The multiple identification problem is $\Omega(p)$.

Proof:

Consider the case where $n=1$: that is, one bit x_i is stored in each processor p_i. Let x be $x_1x_2...x_p$. The address information of a bit is actually its location in x. For a processor P_i, if $x_i=1$ then the output $N_i=x$. Otherwise N_i is the complement string of x. In any algorithm the processor receives a certain number of bits and computes N_i. The transmissions must define the initially unknown part of x, that is a string of length p-1. Call the length of the transmissions LT, and call the Kolomogorov Complexity of x (or its complement string, since they are the same) KC. We claim that $LT \geq KC$, since otherwise LT is a shorter description of the string. Most of the strings of length p-1 have $KC = \Omega(p)$ so $LT = \Omega(p)$. The length of the transmissions received by the processor is $\Omega(p)$ and in each time unit the processor gets one bit; therefore the time of the algorithm is $\Omega(p)$. *QED*

We comment that known techniques used for proving lower bounds, namely information transfer, crossing sequences, fooling sets and arguments involving a network's diameter or a transmission's history (see for example: [15], [18], [11], [14], [10], [9], [5]) do not help us in the broadcasting model. This is because after n units of time the input strings of all the processors can be transmitted while each processor can get only part of this information.

Different models restrict the message space differently. In our algorithms processors send data

information and address information. We trade address transmissions for the necessity of exchanging information with all the processors. This address-data transmission trade-off is the idea that makes this protocol superior to any protocol which allows only transmission of input data. Using [18] it is easy to show that any such restricted protocol forces the processor to get information about each input string in the system directly from the processor holding that string. Thus about p^2 problems are solved, and about $n\ p^2$ bits are exchanged in total. In each time unit only p bits can be received by all processors, and therefore the algorithm must take $\Omega(np)$ time. This demonstrates the differences between the two-processor case, which is the case considered by communication complexity which tries to capture information transfer of input on a VLSI chip, and multi-processor models of communication networks, where more information about the computation environment is known.

7. Conclusions

In this paper we have introduced the synchronous broadcasting model. A problem of information sharing, the multiple identification problem was posed and solved using the model.

We demonstrated the power of broadcasting in distributed models. The cluster tree and simulating sorting networks used in the solutions are efficient schemes for communication organization. Developing methods of communication organization for different communication schemes and network topologies is a crucial step in distributed-algorithm design.

The main open problem related to this work is developing techniques for proving lower bounds for multi-processor problems when we allow broadcasting and transmission of information which is not restricted only to the input strings. This interesting topic requires further extension of the approach used here and those of the field of communication complexity. Developing efficient algorithms which use broadcasting effectively is a challenge as well.

8. Acknowledgement

We thank Manfred Warmuth for helpful discussions and Bruce Abramson, Stuart Haber, Othar Hansson and Andrew Mayer for their comments on earlier versions of the paper.

References

1. Abelson, H. Lower Bounds on Information Transfer in Distributed Computation. Symposium on Foundation of Computer Science, IEEE, October, 1978, pp. 151-158.

2. Ajtai M., J. Komlos and E. Szemeredi. An O(n logn) Sorting Network. Symposium on Theory of Computing, ACM, April, 1983, pp. 1-9.

3. Angluin D. Local and Global Properties in Network of Processors. Symposium on Theory of Computing, ACM, May, 1980, pp. 82-93.

4. Batcher K. Sorting Networks and their Applications. AFIPS Spring Joint Computer Conference, 32, 1968, pp. 307-314.

5. Chandra A. K., M. L. Furst and R. J. Lipton. Multi-Party Protocols. Symposium on Theory of Computing, ACM, April, 1983, pp. 94-99.

6. Gafni E., M.C. Loui, P. Tiwari, D.B. West and S. Zaks. Lower Bounds on Common Knowledge in Distributed Algorithms, with Applications. University of Illinois at Urbana-Champaign, March, 1984.

7. Greenberg A.G. On the Time Complexity of Broadcast Communication Schemes. Symposium on Theory of Computing, ACM, May, 1982, pp. 354-364.

8. Karp R.M. and M.O. Rabin. Efficient Randomized Pattern-Matching Algorithms. Harvard University, Center for Research in Computing Technology, 1981.

9. Marberg J.M. and E. Gafni. Sorting and Selection in Multi-Channel Broadcast Networks. Computer Science Department, Univ. of California, Los Angeles, 1985.

10. Pachl J., E. Korach, and D. Rotem. A Technique for Proving Lower Bounds for Distributed Maximum-Finding Algorithms. Symposium on Theory of Computing, ACM, May, 1982, pp. 378-382.

11. Papadimitriou C. H., and M. Sipser. Communication Complexity. Symposium on Theory of Computing, ACM, May, 1982, pp. 196-200.

12. Saltzer, J.H. Naming and Binding of Objects. In *Operating Systems: an advanced course*, Bayer, R., R. M. Graham, and G. Seegmuller, Ed., Springer-Verlag, 1978, pp. 99-208.

13. T. Leighton. Tight Bounds on the Complexity of Parallel Sorting. Symposium on Theory of Computing, ACM, May, 1984, pp. 71-80.

14. Tiwari P. Lower Bounds on Communication Complexity in Distributed Computer Networks. Symposium on Foundation of Computer Science, IEEE, October, 1984, pp. 109-117.

15. Ullman J. D.. *Computational Aspects of VLSI.* Computr Science Press, Rockville, Maryland, 1984.

16. Watson, R.W. Identifiers (Naming) in Distributed Systems. In *Distributed Systems- Architecture and Implementation: an advanced course*, Lampson B.W., M. Paul, and H.J. Siegert, Ed., Springer-Verlag, 1983, pp. 191-210.

17. Willard D. Log-logarithmic Protocol for Resolving Ethernet and Semaphore Conflicts. Symposium on Theory of Computing, ACM, May, 1984, pp. 512-521.

18. Yao A.C. Some Complexity Questions Related to Distributive Computing. Symposium on Theory of Computing, ACM, May, 1979, pp. 209-213.

19. Yemini Y. and A. Lazar. Towards Distributed Sensor Networks. Conference on Information Science and Systems, Princton University, March, 1982.