One-Bit Algorithms

Amotz Bar-Noy * Joseph Naor † Moni Naor ‡

Abstract

Many algorithms in distributed systems assume that the size of a single message depends on the number of processors. In this paper, we assume in contrast that messages consist of a single bit. Our main goal is to explore how the one-bit translation of unbounded message algorithms can be sped up by pipelining. We consider two problems. The first is routing between two processors in an arbitrary network and in some special networks (ring, grid, hypercube). The second problem is coloring a synchronous ring with three colors. The routing problem is a very basic subroutine in many distributed algorithms; the three coloring problem demonstrates that pipelining is not always useful.

*Computer Science Department, Stanford University, Stanford, CA 94305. Supported in part by a Weizmann fellowship and by contract ONR N00014-85-C-0731.

†Computer Science Department, Stanford University, Stanford, CA 94305. Supported by contract ONR N00014-88-K-0166 and by a grant from Stanford's Center for Integrated Systems. This work was done while the author was a post-doctoral fellow at the University of Southern California, Los Angeles, CA.

‡IBM Almaden Research Center, San Jose, CA 95120. This work was done while the author was with the Computer Science Division, University of California at Berkeley, and Supported by NSF grant DCR 85-13926.

1 Introduction

A distributed system may be viewed as a network whose nodes represent the processors. Two nodes are adjacent if the two corresponding processors can communicate directly. An implicit assumption in most distributed algorithms is that the message length is unbounded ($O(\log n)$ bits where n is the number of processors in the network). This assumption does not take into account that the capacity of a communication line must increase with the size of the network. This is particularly critical in a dynamically growing network.

In this paper we go to the extreme and assume that only one-bit messages are permitted. We call algorithms that obey such restrictions *one-bit algorithms*. Note that our model restricts only the capacity of a communication line; in a single step, a processor can receive (send) bits from (to) different processors. (Unlike other bit-models, see e.g. [5]).

This development is similar to the recent importance that bounded-degree networks have acquired for practical reasons; in a real distributed system, the neighborhood size of a processor is independent of n, the network size.

An algorithm that takes time t and sends a total of M bits, where the maximum length of a single message is m, can be modified to a one-bit algorithm that takes at most time mt and sends $O(M)$ bits. The new algorithm sends the same messages as the old one, except it sends them bit by bit sequentially. A processor cannot start responding to a message before that message has completely arrived (m bits).

Our main goal is to find better one-bit algorithms by using pipelining. The intuition is that a processor may not need all the bits of a message in order to start sending the bits of its next message. We view one-bit algorithms as a first step in designing optimal algorithms in the case that message length is bounded by a parameter m which is independent of the network size. One should note that the concept of pipelining has appeared in the past in various areas of computer science.

The first problem that we address in this paper is routing between two processors in a network along one of the shortest paths between them, such that only processors that belong to the path take part in the routing. We give upper and lower bounds for an arbitrary network, and study the problem in the following networks: ring, torroidal k-ring (a mesh with a wraparound) and hypercube. This is a basic subroutine in many distributed algorithms.

The second problem is coloring a synchronous ring with three colors. Our lower bound implies that, for this problem, the translation of the unbounded-message solution into a one-bit algorithm is optimal up to a constant factor. This demonstrates that allowing pipelining does not always speed up the execution. We believe that this result has independent significance in proving lower bounds in distributed systems.

Further research directions are converting fundamental distributed algorithms into efficient one-bit algorithms for problems such as: minimum weighted spanning tree [3], choosing a

leader [6], fault tolerant systems [1], and more general routing problems.

A preliminary version of this paper appeared in the proceedings of the 7^{th} Annual ACM Symposium on Principles of Distributed Computing, Toronto (1988).

2 A routing problem

2.1 The problem

Let p and q be two distinguished processors in a distributed network. Processor p, the *sender*, wants to send one bit whose value is ε to q, the *destination*, along one of the shortest paths from p to q: $p = x_0 - x_1 - \cdots - x_d = q$. It is forbidden for any processor not on the path to participate in a solution. In addition, q does not have to know who the sender is. The network is asynchronous and the network topology is known to all the processors.

This problem is a relaxation of a natural routing problem where p wants to send a long message to q and q should know who sent the message. Yet, it is general enough because p can send q in the fastest way the rest of the message with its name p, and an end-of-message symbol appended. Now, all the processors along the path know exactly where to forward the bits.

We are mainly interested in optimizing t, the time complexity. In an asynchronous network, t is measured by assuming that messages are transferred in one unit of time and the processors' internal computation is not taken into account. An obvious upper bound is $t \leq d(\lceil \log n \rceil + 1)$ where d is the distance between p and q. (We assume without loss of generality that the processors are labeled from $1, \ldots, n$). This is a direct translation of the simple unbounded-message-size algorithm, where each processor on the path sends the binary expansion of the number q. An obvious lower bound is $t \geq d$ because it takes at least d units of time for the bit to reach its destination.

Notations: Let $\alpha(k)$ be the number of bits needed to represent an integer k in a prefix-free code (a code in which no codeword is a prefix of some other codeword) and let $\beta(k)$ be that representation. Clearly,

$$\lceil \log k \rceil \leq \alpha(k) \leq 2\lceil \log k \rceil.$$

Let $\delta(x)$ denote the degree of x in the network and for every edge (x, y) let $\delta(x, y)$ denote the index of y among the neighbors of x.

2.2 General bounds

An algorithm for an arbitrary network: (folklore) The sender, p, forwards along the path "instructions" for $x_1, \ldots, x_d$ as to which edge x_i should use in communicating. Each x_i waits until it knows about x_{i+1} and then forwards the rest of the bits. The pipelining is

achieved by forwarding these instructions in a prefix free code. A processor concludes that it is the destination when it receives an instruction to use the edge from which it has just received a message. Formally,

- p sends to x_1:
  $\beta(\delta(x_1, x_2))\beta(\delta(x_2, x_3)) \cdots \beta(\delta(x_{d-1}, q))\beta(\delta(q, x_{d-1}))\varepsilon$.
- For $1 \leq i \leq d-1$, x_i sends to x_{i+1}:
  $\beta(\delta(x_{i+1}, x_{i+2})) \cdots \beta(\delta(q, x_{d-1}))\varepsilon$.
- q knows that it is the destination because it has received $\beta(\delta(q, x_{d-1}))$ from x_{d-1}.

(It seems that variations on the above algorithm have perviously appeared in the literature in other settings, and therefore we refer to it as folklore).

Theorem 2.1 *The routing problem can be solved on any network in*

$$d + \sum_{i=1}^{d} \alpha(\delta(x_i)) \leq d + 2\sum_{i=1}^{d} \lceil \log \delta(x_i) \rceil$$

time units.

Proof: The correctness of the algorithm is obvious and we prove the time complexity. The sender sends the $S = \sum_{i=1}^{d} \alpha(\delta(x_i))$ bits sequentially. Whenever x_i receives ε it forwards ε immediately. The sender sends ε after S units of time and it takes d units of time for ε to reach q. All together the process lasts $t = d + \sum_{i=1}^{d} \alpha(\delta(x_i))$. ∎

Theorem 2.2 *Denote by $\Gamma_i(x)$ the number of processors in the network at distance i from x. Then any solution for the routing problem takes at least $d + \log\left(\frac{\Gamma_d(p)}{\delta(p)}\right)$ units of time.*

Proof: The sender can choose one of its neighbors ($\delta(p)$ possibilities). This choice and the bits it sends must determine the identity of q (out of $\Gamma_i(p)$ possibilities). Hence, it must send $\log\left(\frac{\Gamma_d(p)}{\delta(p)}\right)$ bits. The extra d in the lower bound arises since the last bit arrives at q after d units of time. ∎

Corollary 2.1 *For a bounded-degree network with maximum degree Δ the results are tight up to a constant:*

$$\log\left(\frac{\Delta(\Delta-1)^{d-1}}{\Delta}\right) = (d-1)\log(\Delta-1)$$
$$\leq t \leq d(1 + 2\lceil \log \Delta \rceil).$$

Proof: The lower bound is achieved by applying the general lower bound to a Δ-regular tree of height at least d. ∎

2.3 The ring and the torroidal k-grid

Applying the general algorithm to a ring yields an algorithm with $t = 2d$. In this subsection we improve this result to $t = 1 + \lceil 3d/2 \rceil$ even if the processors are indistinguishable. (In this case, the sender wants to send a bit to the processor at distance d from it.) In this setting we present an almost matching lower bound of $t \geq \lceil 3d/2 \rceil$.

The algorithm: Let $d' = \lceil d/2 \rceil$.

- If d is odd then the sender, p, sends the following sequence of bits: $\{1\}^{d'}0\varepsilon$. Otherwise p sends $0\{1\}^{d'-1}0\varepsilon$.
- Let $\varepsilon_1, \ldots, \varepsilon_k$ be the bits that x_i receives from x_{i-1}.
  - If $\varepsilon_1 = 0$ then x_i forwards $1\varepsilon_2 \cdots \varepsilon_k$.
  - If $\varepsilon_1 = \varepsilon_2 = 1$ then x_i forwards $0\varepsilon_3 \cdots \varepsilon_k$. ($x_i$ *absorbs* ε_1.)
  - Otherwise, $\varepsilon_1 = 1$ and $\varepsilon_2 = 0$ then x_i realizes that it is the destination and that ε_3 is the information bit.

Theorem 2.3 *The algorithm for the ring is correct and takes at most* $1 + \lceil 3d/2 \rceil$ *units of time.*

Proof: Assume that d is odd; the case where d is even is similar. The sender sends "1", d' times followed by "0" and ε. Each processor at odd distance from p along the path absorbs one of the "1"s. Hence, the first two bits that q receives are "1" and "0". According to the rules of the algorithm q concludes that it is the destination.

Each bit is either absorbed or forwarded and ε runs along the path without delay. Therefore, $t \leq d' + 1 + d \leq 1 + \lceil 3d/2 \rceil$. ∎

Theorem 2.4 *Any solution for the routing problem on a ring with identical processors must take at least* $\lceil 3d/2 \rceil$ *units of time.*

Proof: We show that for every pair of consecutive processors along the path, at least one must absorb the first bit it receives. In any optimal algorithm, every processor must have a rule for the first bit it receives: either it absorbs "0" or "1" but it cannot forward both. Otherwise, the message would never stop advancing. Without loss of generality, assume that every processor absorbs "1".

Let x and y be a consecutive pair of processors. If x does not absorb the first bit, it must send "1"; otherwise, this "0" makes a full round of the ring. Consequently, y absorbs the first bit it receives.

Thus, at least $d' = \lceil d/2 \rceil$ processors absorb the first bit (q must absorb the first bit) and this causes a delay of d' units of time for ε. ∎

It is possible to give bounds on variants of this problem where the processors are distinguishable. For instance, if $d = O(n)$ the bound is $t = d + \Theta(\sqrt{d})$, as opposed to the latter bound (Theorems 2.3 and 2.4) which is $t = d + \Theta(d)$.

Now, we generalize the ring algorithm to the torroidal k-grid. For clarity, assume that there is a fixed order of the k dimensions and the path follows that order. Moreover, assume that p and q differ in all k dimensions. (If not, our algorithm can be easily generalized). We use the following notation: $x(i, j)$ indicates the j-th processor of the i-th dimension along the path.

The algorithm: Let the path from p to q be:

$$p = x(1,0) - \cdots - x(1,d_1) = x(2,0) - \cdots$$

$$-x(2,d_2) = \cdots = x(k-1,0) - \cdots - x(k-1,d_{k-1})$$

$$= x(k,0) - \cdots - x(k,d_k) = q,$$

where $\sum_{i=1}^{k} d_i = d$. The processor $x(i, d_i) = x(i+1, 0)$, an *intersection* processor, is the processor on the path that switches the route to the $i+1$st dimension.

For every dimension i, p sends $1 + \lceil d_i/2 \rceil$ bits according to the ring algorithm. In each dimension ε_i indicates one of two possible changes of direction. These k sequences are followed by (the real) ε. By receiving bits from a processor adjacent to q in the kth dimension, q concludes that it is the destination.

Theorem 2.5 *For the torroidal k-grid there is an algorithm for the routing problem that requires*

$$d + \sum_{i=1}^{k-1} \left(1 + \left\lceil \frac{d_i}{2} \right\rceil\right) = \left\lceil \frac{3d}{2} \right\rceil + 2(k-1)$$

time units.

2.4 The hypercube

Corollary 2.2 *Every algorithm for solving the routing problem in the k-dimensional hypercube requires at least*

$$\log \frac{\binom{k}{d}}{k} \geq d \log \frac{k}{d} - \log k$$

time units.

Proof: Each processor in the hypercube is represented by a binary vector of length k. The processors at distance d from p are those whose representation differs from that of p in exactly d indices. Therefore, $\Gamma_d(p) = \binom{k}{d}$. ∎

The general upper bound yields a bound of $d(1 + \lceil \log k \rceil)$. We can improve this result to $d + \alpha(a_1) + \ldots + \alpha(a_d)$ where $\sum_{i=1}^{d-1} a_i \leq k$. We assume that a total order is defined among the hypercube dimensions, and all the processors are familiar with it. Moreover, each message is sent according to this order.

We modify the general algorithm as follows. Instead of instructing x_i which dimension to use, p sends the difference between the dimension on which x_i received messages and the one on which x_i should send messages. For example, suppose p and q differ on dimensions $8, 12, 20$ and 25. Then, p sends $\beta(4)\beta(8)\beta(5)$ to its neighboring processor on dimension 8. The algorithm proceeds similar to the general one.

When x_i receives $\beta(a_i)$ from dimension j, it forwards the rest of the bits on dimension $j + a_i$. The end of the process is indicated by $a_i = 0$.

Theorem 2.6 *There is an algorithm for the routing problem on the k-dimension hypercube which takes at most*

$$d + d\alpha(k/(d-1)) \leq d(1 + 2\lceil \log(k/(d-1)) \rceil)$$

units of time.

Proof: The proof is implied by the fact that the maximum is achieved when $a_i = \frac{k}{d-1}$ for $1 \leq i \leq d-1$ (a_d is always 0). ∎

3 The ring 3-coloring problem

A fundamental application of symmetry breaking is that of coloring a ring of n processors with 3 colors. We assume a completely synchronous unidirectional ring of n processors, where at each round every processor sends its neighbor one bit. We are interested in the number of rounds it takes until each processor is assigned one of three colors so that no two successive processors are colored the same. At the beginning, each processor has a unique identification number whose size is polynomial in n.

The complexity of the case where messages are unbounded has been completely characterized: Cole and Vishkin [2] gave an algorithm that required $O(\log^* n)$ messages (see also [4]). Linial [7] showed that $\Omega(\log^* n)$ messages are necessary. We prove tight bounds on the number of rounds of one-bit algorithms, and actually note that the [2] algorithm is optimal in the one-bit model as well.

Corollary 3.1 (upper bound) *There is an $O(\log n)$ solution for the ring 3-coloring problem.*

Proof: Converting the [2] algorithm into a one-bit algorithm yields the upper bound. In that algorithm $r = O(\log^* n)$ messages are sent, where the ith message is of size $c \log^{(i)} n$ for some constant c. The number of bits that are sent is $\sum_{i=1}^{r} c \log^{(i)} n$ which is $O(\log n)$. ∎

Theorem 3.1 (lower bound) *Every one-bit algorithm for the ring* 3*-coloring problem takes* $\Omega(\log n)$ *units of time.*

Proof:

Proposition 3.1 *If four consecutive processors in the ring send and receive exactly the same bit sequence, then they cannot color themselves.*

Proof: Among the four consecutive processors, at least two will be assigned the same color. Since for each processor the other processors are indistinguishable, the two that get the same color might be adjacent. ∎

Each processor has a protocol P and denote by P_i the sub-protocol of P at the i-th round. P_i can be viewed as a $\{0,1\}$-labeling of the leaves of a full binary tree of depth $i-1$. A path from the root to a leaf in this tree corresponds to the messages received so far (with 0 represented by a left turn and 1 represented by a right turn), and the label corresponds to the message that the processor sends in the kth round.

Proposition 3.2 *if there are* $k+4$ *consecutive processors, all having the same sub-protocols* $P_1, P_2, \ldots, P_k$*, then they cannot color themselves in* k *rounds.*

Proof: The last 4 processors among these $k+4$ processors will send and receive exactly the same bits for the first k rounds. Hence, by Proposition 3.1 they cannot color themselves. ∎

Proposition 3.2 yields a lower bound of $\Omega(\log \log n)$ rounds for 3-coloring. There are only $2^{2^{k-1}}$ different protocols[1] up to the kth round. Hence, there must be $\frac{n}{2^{2^{k-1}}}$ processors that have the same protocols up to the kth round. If $k < \frac{n}{2^{2^{k-1}}} - 4$ and $k+4$ of those processors are consecutive, then they cannot color themselves. This implies that any algorithm takes at least $\Omega(\log \log n)$ rounds.

Instead of counting the number of different protocols as we did above, we can count the number of different WYGWYS paths, defined as follows: Let $P_1, P_2, \ldots, P_k$ be the sub-protocols for the first k rounds of a given processor. The What You Get is What You Sent (WYGWYS) path of length k is the sequence of messages sent by this processor when its k predecessors in the ring all have protocols $P_1, P_2, \ldots, P_k$ as their corresponding sub-protocols.

Note that if k consecutive processors in the ring all have the same WYGWYS path of length k, then the last processor will output its WYGWYS path for the first k rounds and therefore,

Proposition 3.3 $k+4$ *consecutive processors, all having the same WYGWYS path of length*

[1]The assumption here is that a processor cannot be idle; this can only change constant factors.

k, cannot color themselves in k rounds.

There are only 2^k different WYGWYS paths of length k. Hence, there must be $\frac{n}{2^k}$ different processors that have the same WYGWYS path of length k. If $k < \frac{n}{2^k} - 4$ and $k+4$ of those processors are consecutive in the ring, then the last four processors send and receive the same bit sequence and so by Proposition 3.1 they cannot color themselves in k rounds.

For sufficiently large n, taking k to be $\log n - \log\log n$, implies the claimed $\Omega(\log n)$ lower bound on the time. ∎

Remark 1: The proof can be generalized to work for bidirectional rings as well.

Remark 2: The case of randomized algorithms has been studied by Linial and Naor [8], when there is no restriction on the message length. It was shown to have complexity $\Theta(\log^* n)$. For the one-bit message model, it can be shown that $\Theta(\sqrt{\log n})$ bits are both necessary and sufficient.

References

[1] A. Bar-Noy and D. Dolev, *Families of Consensus Algorithms,* VLSI Algorithms and Architectures, 3rd Aegean Workshop on Computing, Corfu, Greece (1988), pp. 380-390.

[2] R. Cole and U. Vishkin, *Deterministic Coin Tossing and Accelerating Cascades: Micro and Macro Techniques for Designing Parallel Algorithms,* Proceedings of 18th Symposium on Theory of Computing, pp. 206-219, 1986.

[3] R. G. Gallager, P.A. Humblet and P.M. Spira, *A Distributed Algorithm for Minimum Weight Spanning Trees,* ACM Trans. on Program. Lang. & Systems, Vol. 5, pp. 66-77, January 1983.

[4] A. Goldberg, S. Plotkin and G. Shannon, *Parallel Symmetry Breaking in Sparse Graphs,* SIAM J. Disc. Math. Vol. 1, pp. 434-446 (1988).

[5] T. Leighton, *Tight Bounds on the Complexity of Parallel Sorting,* Proceedings of 16th Symposium on Theory of Computing, pp. 71-80, 1984.

[6] G. Le Lann, *Distributed Systems – towards a formal approach,* Information processing (editor b. gilchrist), pp. 155-160, 1977.

[7] N. Linial, *Distributive Graph Algorithms - Global Solutions From Local Data,* Proceedings of 28th Symposium on Foundations of Computer Science, pp. 331-335, 1987.

[8] N. Linial and M. Naor, In preparation.