JOURNAL OF ALGORITHMS 3, 245–260 (1982)

An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle

DANNY DOLEV, MARIA KLAWE, AND MICHAEL RODEH*

IBM Research, K51-61F, 5600 Cottle Road, San Jose, California 95193

Received April 26, 1981; revised October 21, 1981

In this paper we present algorithms, which given a circular arrangement of n uniquely numbered processes, determine the maximum number in a distributive manner. We begin with a simple unidirectional algorithm, in which the number of messages passed is bounded by $2n \log n + O(n)$. By making several improvements to the simple algorithm, we obtain a unidirectional algorithm in which the number of messages passed is bounded by $1.5n \log n + O(n)$. These algorithms disprove Hirschberg and Sinclair's conjecture that $O(n^2)$ is a lower bound on the number of messages passed in undirectional algorithms for this problem. At the end of the paper we indicate how our methods can be used to improve an algorithm due to Peterson, to obtain a unidirectional algorithm using at most $1.356n \log n + O(n)$ messages. This is the best bound so far on the number of messages passed in both the bidirectional and unidirectional cases.

1. INTRODUCTION

Consider a circular arrangement of n asynchronous communicating processes in which communication occurs only between neighbors around the circle. All processes have the same program, and differ only by having distinct numbers (known only to the owners) stored in their local (non-shared) memory. Our objective is to obtain an efficient algorithm which finds the maximum of these numbers.

Rapid progress has been made on this problem during the last few years. In 1977 LeLann [6] presented an algorithm which requires $O(n^2)$ messages. Chang and Roberts [2] in 1979, proposed an improved algorithm which uses only $O(n \log n)$ messages on the average (assuming that all permutations of the data are equiprobable); however, their algorithm still requires $O(n^2)$ messages in the worst case. Also in 1979, Gallager *et al.* [3] obtained, as part of an $O(e + n \log n)$ algorithm for finding a minimum spanning tree in a

*On sabbatical from IBM Israel Scientific Center, Technion City, Haifa, Israel.

0196-6774/82/030245-16 $02.00/0

network with e edges and n nodes, an algorithm which finds the maximum in a circular arrangement of processes in $O(n \log n)$ time. Another $O(n \log n)$ algorithm was suggested by Hirschberg and Sinclair [4], and later improved slightly by Burns [1], who also proved that $\Omega(n \log n)$ is a lower bound on the number of messages.

All of the $O(n \log n)$ algorithms mentioned above depend on messages being allowed to pass in both directions around the circle. In fact, Hirschberg and Sinclair [4] made the conjecture that "models in which message passing is unidirectional must, in the worst case, have quadratic behavior and that bidirectional capability is necessary in order to achieve $O(n \log n)$ performance." In this paper we begin by presenting a simple unidirectional algorithm which disproves this conjecture, since it requires at most $2n \log n + n$ messages. (In this paper, by $\log n$ we mean $\log_2 n$.) After submitting the original version of our paper containing this algorithm, we learned that Peterson had also submitted a paper [7] containing the same algorithm. In revising our paper we managed to modify our algorithm to obtain an algorithm using at most $1.5n \log n$ messages. After seeing our modifications Peterson obtained a quite different and much simpler algorithm using approximately $1.4402n \log n$ messages. By making analogous modifications to Peterson's new algorithm, we were able to obtain an algorithm using at most $1.356n \log n$ messages, the lowest number so far in either the unidirectional or bidirectional cases.

As far as average behavior is concerned, all the above algorithms send $O(n \log n)$ messages. Recently, Itai and Rodeh [5] proposed probabilistic algorithms which solve the problem in linear time, assuming that the processes are synchronous and that the size of the circle is known. Thus far, no lower bound has been established on the average number of messages needed to solve the problem in an asynchronous system. There is also no nontrivial lower bound known for the synchronous version of this problem, nor for the case that the size of the ring is known.

The computation model we use in this paper assumes that each process has a queue in which incoming messages are stored and processed in first come first serve order, and moreover that communication channels between processes preserve the order of the messages sent. We also assume that each process can only send messages to the process on its right, and only receives messages from the process on its left.

In the next section we present the simple unidirectional algorithm using at most $2n \log n + O(n)$ messages. We begin Section 3 by describing several possible improvements to this algorithm, and then go on to prove that by incorporating these improvements into our algorithm, we are able to bound the number of messages by $1.5n \log n + O(n)$. Finally, in the last section we indicate how Peterson's new algorithm can be modified to obtain an algorithm using at most $1.356n \log n + O(n)$ messages.

2. The Simple Unidirectional Algorithm

2.1. *The Algorithm*

Before describing the algorithm formally, we give an intuitive explanation of how the algorithm proceeds. The basic idea depends on the fact that, since order of communication is preserved, the effect of the algorithm is as though the activities of the system could be separated into synchronous phases. Initially all processes are active. In each phase, sufficient messages are passed so that each active process learns the current numbers of the two closest active processes on its left. With this information, each active process can determine if it is the active process immediately following a local maximum. If so, it assumes the local maximum as its own number, and continues as active on to the next phase. If not, it becomes passive and merely acts as a communication relay, passing on the messages initiated by active processes during later phases.

The algorithm uses messages of two types:

M1. of the form $\langle 1, i\rangle$, and

M2. of the form $\langle 2, i\rangle$,

where i denotes one of the numbers stored in the processes. Each process v has a variable $\max(v)$, in which initially its number is stored, and another variable $\text{left}(v)$ which it can use, when active, to store the number of the active process to its left. As the algorithm proceeds, if v is an active process, then $\max(v)$ will contain the number that v is currently representing.

Since passive processes simply pass on unchanged whatever messages they receive, in order to describe the algorithm (hereby referred to as Algorithm A) we need only define the behavior of an active process v, which is as follows:

A0. Send the message $\langle 1, \max(v)\rangle$.

A1. If a message $\langle 1, i\rangle$ arrives do as follows:

1. If $i \neq \max(v)$ then send the message $\langle 2, i\rangle$,
   and assign i to $\text{left}(v)$.
2. Otherwise, halt—$\max(v)$ is the global maximum.

A2. If a message $\langle 2, j\rangle$ arrives do as follows:

1. If $\text{left}(v)$ is greater than both j and $\max(v)$
   then assign $\text{left}(v)$ to $\max(v)$,
   and send the message $\langle 1, \max(v)\rangle$.
2. Otherwise, become passive.

The behavior of a process consists of a sequence of steps of the form A0, A1, A2, A1, A2, A1, A2, ... In A1 the process handles messages of type M1, while in A2 it handles messages of type M2. Because of the alternation between the steps A1 and A2, it is not really necessary in practice to

distinguish between the two message types. Hence the algorithm could be implemented using a single type of message of the form $\langle i \rangle$. We have refrained from doing this as the message types M1 and M2 perform different logical functions, and distinguishing between them helps in the analysis of the algorithm.

2.2. *Correctness and Complexity*

In order to deal more easily with the questions of correctness and complexity, let us define the notion of phase a little more precisely. For an active process we say that the pth phase begins immediately before it sends its $(p+1)$st message of type M1. Similarly, for a passive process we say that the pth phase begins immediately before it receives its $(p+1)$st message of type M1. To be able to indicate precisely the value of $\max(v)$ at certain stages of the execution of the algorithm, we use $p\text{-max}(v)$ to denote the value that $\max(v)$ has at the beginning of the pth phase.

It is straightforward to verify the following facts about the behavior of Algorithm A.

(2.2.1) During each phase each process sends exactly two messages.

(2.2.2) If u and v are distinct processes which are active in the pth phase, then $p\text{-max}(u) \neq p\text{-max}(v)$.

(2.2.3) If u is active in the pth phase, and if $p\text{-max}(u)$ is a local maximum among the processes active in the pth phase, then the process v, which is closest to u on the right among those active processes, will be active in the $(p+1)$st phase, and will have $(p+1)\text{-max}(v) = p\text{-max}(u)$.

Conversely if $p\text{-max}(u)$ is not a local maximum, then no process v which is active in the $(p+1)$st phase will have $(p+1)\text{-max}(v) = p\text{-max}(u)$.

(2.2.4) If there are at least two active processes in the pth phase, then for each such active process v, the M1 message $\langle 1, i \rangle$ it receives in the pth phase has $i \neq p\text{-max}(v)$.

Conversely if v is the only active process, then $i = p\text{-max}(v)$.

These facts show that if the algorithm ever terminates, it does so correctly. We now observe that the algorithm must terminate in at most $\log n + 1$ phases.

LEMMA 2.2.5. *The number of processes active in the* $(p+1)$st *phase is at most half the number active in the* pth *phase.*

Proof. In any arrangement of k numbers on a circle, the number of local maxima is at most $k/2$, since for any number which is a local maximum, the number immediately following it is not. The proof is completed by observing that (2.2.2) and (2.2.3) guarantee that the number of processes active in

the $(p+1)$st phase is exactly the number of active processes in the pth phase whose values are local maxima. □

COROLLARY 2.2.6. *The number of messages sent using Algorithm A is at most* $2n \log n + n$.

By (2.2.1) at most $2n$ messages are sent in each phase. Furthermore, in the last phase at most n messages are sent as only one process is active. Finally, the preceding lemma shows that there are at most $\log n + 1$ phases, which completes the proof. □

3. THE IMPROVED UNIDIRECTIONAL ALGORITHM

3.1. *Improvements to Algorithm A*

Algorithm A is based on the recognition of local maxima. In this section we investigate techniques by which we can avoid sending type M2 messages as often as possible, while still guaranteeing that local maxima (or at least the global maximum) are recognized. Let u, v, and w be three consecutive active processes. According to Algorithm A, process v sends $\langle 1, \max(v)\rangle$ to w, and then w waits for v to send on $\langle 2, \max(u)\rangle$ in order to check whether $\max(v)$ is the largest of $\max(u)$, $\max(v)$ and $\max(w)$. A straightforward observation is that the relationship between $\max(u)$ and $\max(v)$ is known to v as soon as it receives the message $\langle 1, \max(u)\rangle$. Thus v could simply send w *one bit* of information, 1 if $\max(u) < \max(v)$ and 0 otherwise. This modification reduces the number of bits sent but does not influence the number of messages.

Now let us reconsider the case $\max(u) > \max(v)$. In this case w will become passive independent of the relationship between $\max(v)$ and $\max(w)$. If v refrains from initiating any message of type M2, the next message w receives will be of type M1. By modifying the algorithm so that any active process receiving two consecutive M1 type messages becomes passive, the algorithm still functions correctly when active processes like v (i.e., receiving an M1 message with value higher than their own) do not initiate an M2 message. In implementing this modification, we add a third state for processes, the "waiting" state. This modification clearly reduces the number of messages sent, but it can be shown that by itself it does not suffice to bring the number down to $(3/2)n \log n + O(n)$.

Next, let u and v be two adjacent active processes with v to the right of u. If v is still active in the next phase, the value it will represent is $\max(u)$. Thus if in some way we can determine that $\max(u)$ is not the global maximum, then we could make v passive without affecting the correctness

of the algorithm. Such an opportunity occurs if there is some (passive) process x lying between u and v, with $\max(x) > \max(u)$. When passing on the M1 message initiated by u, x can observe that $\max(u)$ is not the global maximum. Our second modification, again resulting in the saving of some M2 messages, is as follows. If the above process x then receives an M2 message initiated by u, x does not send on this M2 message towards v. As a consequence v becomes passive at the start of the next phase. Note that x must pass on u's M1 message in order to allow v to initiate an M2 message if need be, since $\max(v)$ could be the global maximum.

The third modification deals with the possible existence of unusually long sequences of passive processes. The problem with the first two modifications is that if, after the first few phases, it happens that all the active processes are situated in a small fraction of the circle, it is possible that all the savings of M2 messages occur on that small section, whereas one M2 message is passed on for almost the entire length of the circle. We concentrate now on preventing this from occurring. Our aim is to guarantee that an M2 message initiated in the pth phase will never travel farther than 2^p from its initiator. To accomplish this, we add a counter with the initial value of 2^p to each M1 message initiated in the pth phase. Active processes receiving M1 messages process them as before, but passive and waiting processes decrement the counter before sending the message on. Moreover, whenever a passive or waiting process v receives an M1 message with counter value $= 1$ and number at least $\max(v)$, after sending on the decremented M1 message, v then enters the waiting state, setting $\max(v)$ to the number carried in the M1 message. If the next message v receives is of type M2 then v will be active in the next phase; otherwise, v will return to the passive state. Combining this modification with the first two, guarantees that if an active process u initiates an M1 message in the pth phase, and then also initiates an M2 message in the same phase, the M2 message will stop within distance 2^p at some process v which is either in the waiting state, or has $\max(v) > \max(u)$.

The first two changes, though reducing the number of M2 messages sent, did not significantly alter the behavior of the algorithm. The third, however, definitely does. In particular, processes which become passive during some phase may become active again at a later phase. Moreover, there may be values represented by active processes in the pth phase, which are not local maxima among the values represented by active processes in the $(p-1)$st phase. In spite of these differences, we will prove that the algorithm does function and terminate correctly. In fact we will show once again that it terminates in at most $\log n + 1$ phases. In addition we will be able to show that the number of M2 messages sent in the pth phase is at most $n2^p/(2^{p+1} - 1)$ which yields the desired bound on the number of messages.

3.2. *Algorithm B*

Each process v maintains the following local variables:

(i) phase(v)—initially 0.

(ii) state(v)—contains one of "active," "waiting," or "passive."

(iii) id(v)—the number originally stored in v.

(iv) max(v)—the accumulated maximum stored in v. Initially max(v) = id(v).

This algorithm uses two types of messages, M1 and M2, similar to those in Algorithm A. The form of M2 is unchanged, while M1 is of the form $\langle 1, i, \text{phase}, \text{counter}\rangle$, where $0 \leq \text{counter} \leq 2^{\text{phase}}$.

The behavior of the processes is as follows:

B1. An active process v initiates the message

$$\langle 1, \max(v), \text{phase}(v), 2^{\text{phase}(v)}\rangle.$$

When a message of the form $\langle 1, i, p, c\rangle$ arrives, the process executes the following program:

if $i >$ max(v)) **then begin**
  max(v) : = i;
  state(v) : = "waiting";
  end;
else begin
  state(v) : = "passive";
  send $\langle 2, \max(v)\rangle$;
  end.

Note that the number carried in message of type M2 in Algorithm B is different from that in Algorithm A, i.e., $\langle 2, \max(v)\rangle$ is sent rather than $\langle 2, i\rangle$. This difference is a key factor in our implementation of the second modification, which we explain now in more detail. A passive process x receiving the M2 message $\langle 2, \max(v)\rangle$ knows that it need not send it on if $\max(v) < \max(x)$. This is because the process waiting for that M2 message in order to become active in the next phase would be representing max(v) if it did become active, and max(v) is clearly not the global maximum.

Notice also that an active process never receives messages of the form $\langle 2, j\rangle$. Such messages are received only by processes which are either waiting or passive.

B2. A waiting process may receive messages of both types.

1. If a message of type M2 arrives then j must be equal to max(v).
   In this case v sets:
   phase(v) : = phase(v) + 1 and
   state(v) : = "active";
   and begins executing B1.

2. If the arriving message is of type M1,
   v sets state(v) : = "passive";
   and continues the execution of step B3.1 (see below).

B3. A passive process v behaves as specified in B3.1 or B3.2 depending on the type of the arriving message:

1. When $\langle 1, i, p, c\rangle$ arrives, the following code is executed:
   if ($i \geq \max(v)$ and $c \geq 1$) **then begin**
   $\max(v) := i$;
   if $c > 1$ **then send** $\langle 1, i, p, c-1\rangle$;
   else if $c = 1$ **then begin**
   state(v) : = "waiting";
   send $\langle 1, i, p, 0\rangle$;
   phase(v) : = p;
   end;
   end;
   else send $\langle 1, i, p, 0\rangle$.
2. When a message $\langle 2, j\rangle$ arrives, the process does the following.
   if $j < \max(v)$ **then skip**
   else begin
   send $\langle 2, j\rangle$;
   end

As described above, the algorithm does not explicitly terminate. However, it is easy to modify it slightly by always testing whether an arriving number is equal to id(v). If this happens, then id(v) is the global maximum, and the algorithm terminates. Another possible modification is to exclude the phase number from messages of type M1. Instead, every process could count the number of messages of type M1 which pass by, and use this as the phase number.

3.3. *An Analysis of Algorithm B*

We begin this section by establishing some properties of Algorithm B which follow in a straightforward manner from its definition. We then proceed to prove, using induction on the phase number, some slightly less obvious statements about its behavior, which we will need for its complexity analysis.

To begin with, note that for each p, each process receives exactly one message of type M1 whose phase value is p. Thus, for each process v, we can define p-max(v) by 0-max(v) = id(v) and for $p \geq 1$, p-max(v) is the value of the variable max(v) immediately after v has received and processed its M1 message whose phase value equals $p - 1$. We also define $A(p)$ to be the set of processes which initiate (as in step B1 of Algorithm B) an M1 message with phase value equal to p. In other words $A(p)$ is the set of

processes which are active at the beginning of the pth phase, and p-max(v) is the value of max(v) at that moment.

The following statements are easy to verify from the definition of Algorithm B. We include the proofs of the first two statements to give the reader the flavor of the arguments involved.

(3.3.1) If u and v are distinct processes in $A(p)$ then p-max(u) $\neq$ p-max(v).

Proof. This is obviously true for $p = 0$, so assume $p \geq 1$ and that the statement holds for $p - 1$. Now since the processes in $A(p-1)$ have distinct $(p-1)$-max values, it is clear that any pair of M2 messages in the $(p-1)$st phase, which are initiated by different processes in $A(p-1)$, must be carrying different values. Now since any M2 message causes at most one process v to be in $A(p)$, and moreover that v will have p-max(v) equal to the value that the M2 message was carrying, it follows that the processes in $A(p)$ must have distinct p-max values. □

(3.3.2) During the pth phase each message of type M2 travels distance at most 2^p from the process which initiates it.

Proof. Let u be a process which initiates an M2 message in the pth phase, and let v be the process in $A(p)$ closest to u on the right. If the distance from u to v is greater than 2^p, then the process y at distance exactly 2^p from u enters the waiting state upon receiving the M1 message that u sends in the pth phase. Thus u's M2 message cannot travel beyond y, because waiting processes do not pass on M2 messages. On the other hand if the distance from u to v is no more than 2^p, then if max(v) < max(u), v enters the waiting state after receiving u's M1 message, and as before u's M2 message cannot travel beyond v. If max(v) > max(u), then again u's M2 message cannot travel beyond v, since the value carried by that M2 message is less than max(v), and hence v cannot send it on. □

(3.3.3) During any phase at most one message of each type passes through an edge.

(3.3.4) Let M be the largest of the values id(v) over all processes v. Then if some u in $A(p)$ has p-max(u) = M, then there is a process v in $A(p+1)$ with $(p+1)$-max(v) = M.

By induction on p, (3.3.4) immediately yields

(3.3.5) There is always some process v in $A(p)$ with p-max(v) = M.

It is this fact that ensures the correctness of the algorithm.

Before proving the next lemma we introduce some notation to make our task a little easier. First of all, for any two processes u and v we use $d(u, v)$ to denote the distance from u to v on the circle, travelling to the right from u

(i.e., in the clockwise direction from u). We also use $u < v < w$ to signify that if we travel to the right around the circle from u, we will encounter v before w. When necessary we will use $\leq$ instead of $<$ to denote the possibility that two of the processes are the same. Finally for any number j we will use origin(j) to denote the process v which had j as its original value, i.e., which has id(v) = j.

LEMMA 3.3.6. *Let v be a process in $A(p)$.*

(a) $d(\text{origin}(p\text{-max}(v)), v) = 2^p - 1$.

(b) $p\text{-max}(y) = p\text{-max}(v)$ for each process y with origin(p-max(v)) $\leq y \leq v$.

(c) If w is a distinct process from v in $A(p)$, then $d(w, v) \geq 2^p$.

Proof. The statements are obviously true for $p = 0$, so assume that $p \geq 1$ and that they hold for $p - 1$. Let u be the process in $A(p-1)$ which is closest on the left to v. Since v is in $A(p)$, in the $(p-1)$st phase v must receive an M2 message which was initiated by u. Thus p-max(v) = $(p-1)$-max(u), and moreover, by (3.3.2), $d(u, v) \leq 2^{p-1}$. For convenience let us denote p-max(v) by j. Since v received u's M2 message, for each process y with $u \leq y < v$ we must have $(p-1)$-max(y) $\leq j$, which in turn shows that p-max(y) = j. We now claim that $d(u, v) \geq 2^{p-1}$. This follows from (c) in the inductive hypothesis if v is in $A(p-1)$. If v is not in $A(p-1)$, then the only way v could enter the waiting state in the $(p-1)$st phase (and hence possibly become active) is if $d(u, v) = 2^{p-1}$. Thus we have in any case that $d(u, v) = 2^{p-1}$, and hence

$$\begin{aligned} d(\text{origin}(j), v) &= d(\text{origin}(j), u) + d(u, v) \\ &= 2^{p-1} - 1 + 2^{p-1} \\ &= 2^p - 1 \end{aligned}$$

as desired.

In order to prove (b), all that remains to be shown is that if origin(j) $\leq y < u$, then p-max(y) = j, since we have already covered the case $u \leq y \leq v$. By the inductive hypothesis we have $(p-1)$-max(y) = j, and hence it suffices to show that the M1 message y receives in the $(p-1)$st phase carries a max-value less than j. Let x be the process in $A(p-1)$ closest on the left to u. Then as $d(x, u) \geq 2^{p-1}$, we have $x < y < u$, and hence the M1 message y receives in the $(p-1)$st phase is the one initiated by x. Now we are done since the fact that u initiated an M2 message implies that the value carried in x's M1 message, $(p-1)$-max(x), is less than j.

To prove (c), it suffices to show that if $0 < d(y, v) < 2^p$, then y is not in $A(p)$. However, this follows immediately from (a), (b) and 3.3.1, since if $d(y, v) < 2^p$, then origin(j) $\leq y < v$, and hence p-max(y) = j = p-max(v), which shows that both y and v cannot be in $A(p)$. □

COROLLARY 3.3.7. *For each* p, $|A(p)| \leq n/2^p$.

Proof. This follows directly from Lemma 3.3.6(c). □

LEMMA 3.3.8. *Let* u *be a process which initiates an* M2 *message in the* p*th phase, let* f *be the furthest process to the right of* u *which sends on* u*'s* M2 *message in the* p*th phase, and let* g *be the process with* $d(f, g) = 2^p - 1$. *Then for any process* y *with* $f < y \leq g$, y *does not send an* M2 *message in the* p*th phase.*

Proof. Since f is the last process to send on u's M2 message, and since only processes in $A(p)$ can initiate M2 messages in the pth phase, it suffices to show that there is no process y in $A(p)$ with $f < y \leq g$ which initiates an M2 message in the pth phase. Let w be the process in $A(p)$ which is closest to u on the right. First suppose that $p\text{-max}(w) < p\text{-max}(u)$. Then w does not initiate an M2 message in pth phase, and if z is any process in $A(p)$ other than u or w, then we cannot have $f < z \leq g$, since $d(u, z) = d(u, w) + d(w, z) \geq 2^{p+1}$ by Lemma 3.3.6(c), but $d(u, g) = d(u, f) + d(f, g) \leq 2^{p+1} - 1$ by 3.3.2.

Suppose now that $p\text{-max}(w) = j > p\text{-max}(u)$. In this case we are able to show that w is to the right of g, and hence there is no process in $A(p)$ between f and g. To see this, notice that since $p\text{-max}(\text{origin}(j)) = j > p\text{-max}(u)$, even if origin$(j)$ receives the M2 message initiated by u in the pth phase, origin(j) does not send it on. This shows that f is to the left of origin(j). By Lemma 3.3.6(a) $d(f, w) \geq 2^p$, and hence we have $f < g < w$. □

COROLLARY 3.3.9. *The number of messages of type M2 sent in the* p*th phase is at most* $n2^p/(2^{p+1} - 1)$.

Proof. This follows immediately from 3.3.2 and Lemma 3.3.8. □

THEOREM 3.3.10. *The total number of messages sent is less than* $(3/2)n \log n + (8/3)n$.

Proof. By Corollary 3.3.7 the algorithm terminates in at most $\log n + 1$ phases. As in Algorithm A, no M2 messages are sent in the last phase, and thus by Corollary 3.3.9 the number of M2 messages sent is less than $(1/2)n \log n + (5/3)n$. Finally, the number of M1 messages is obviously at most $n(\log n + 1)$, which completes the proof. □

To evaluate the number of bits sent, notice that every message of type M1, assuming that the phase number has been eliminated, contains $1 + \log n + m$ bits, where m is the size of the representation of the individual numbers. By modifying the algorithm to represent messages of type 2 as a single bit, the number of bits sent in the algorithm becomes $n \log n \times (\log n + m) + O(n \log n)$. This is worse than the optimized version of Algorithm A which uses $nm \log n + O(n \log n)$ bits.

4. The Modified Peterson Algorithm

In this section we present an algorithm which uses the least number of messages known so far. We begin by describing a new algorithm due to Peterson [7], and stating several of its properties. Because of the similarity of Peterson's new algorithm to the simple algorithm in Section 2, we are able to make analogous modifications which reduce the number of messages sent. For the sake of brevity we will not include proofs which are directly analogous to those in Section 3, but rather concentrate on the new ideas which are necessary to analyse the performance of the modified Peterson algorithm.

We now present Peterson's new algorithm which he proved uses at most $cn \log n$ messages, where $c = 1/\log((1 + \sqrt{5})/2) \simeq 1.440420$. We actually present a version which is slightly different from Peterson's in order to make the task of modification easier. We can think of the algorithm proceeding by alternating between two types of phases. Each process is active at the beginning of phase 0, and later becomes passive at some point during the execution of the algorithm.

At the beginning of each even-numbered phase, each active process initiates an M1 message containing its current max value. This message is passed on by passive processes until it reaches the next active process. When an active process receives an M1 message containing the value i, if i is greater than its own max value, the process becomes passive. Otherwise it starts the next phase (odd-numbered) by initiating an M2 message containing its own current max value, which likewise is passed on by passive processes until it reaches the next active process. When an active process receives an M2 message containing the value i, if i is less than its own current max value, the process becomes passive. Otherwise it changes its current max value to i, and begins the next even-numbered phase, issuing an M1 message containing i. We also assume that whenever a passive process v passes on an M2 message carrying a value i which is greater than its own max value, the process v updates its own max value to i.

As before let $A(k)$ denote the set of processes which are active at the beginning of the kth phase. Peterson obtained the $1.44042\ldots n \log n + O(n)$ bound as follows. He first observed that $|A(k)| \geq |A(k+1)| + |A(k+2)|$ for each k, which shows that if the algorithm terminates in p phases, then $n \geq F(p+1)$, the $(p+1)$st Fibonacci number. Since $F(p+1) = ((1 + \sqrt{5})/2)^{p+1}/\sqrt{5} + O(1)$, solving for p yields the desired result.

Peterson's algorithm also has the following properties, which are not hard to prove by induction on the phase number. As before, k-max(v) denotes the current max value held by the process v at the beginning of the kth phase, and origin(i) denotes the process which held i as its initial value. For convenience, if v is a process in $A(k)$, we define the left neighbor of v in

$A(k)$ to be the process in $A(k)$ which is closest to v on the left. Also, as noted before, $F(j)$ denotes the jth Fibonacci number, where we assume $F(0) = 0$, $F(1) = 1$, etc.

4.1. Let u and v be in $A(k)$ such that u is v's left neighbor in $A(k)$. If k is even then $k\text{-max}(u) > k\text{-max}(v)$ implies that $d(u, v) \geq F(k + 1)$, and $k\text{-max}(u) < k\text{-max}(v)$ implies that $d(u, v) \geq F(k + 2)$. If k is odd then $k\text{-max}(u) > k\text{-max}(v)$ implies that $d(u, v) \geq F(k + 2)$, and $k\text{-max}(u) < k\text{-max}(v)$ implies that $d(u, v) \geq F(k + 1)$.

4.2. Let k be odd and $u \in A(k)$. Then

$$d(\text{origin}(k\text{-max}(u)), u) \geq F(k + 1) - 1.$$

4.3. If u and v are different processes in $A(k)$ then $k\text{-max}(u) \neq k\text{-max}(v)$.

4.4. If u is in $A(k)$ then for each process y with $\text{origin}(k\text{-max}(u)) < y \leq u$, we have $k\text{-max}(y) = k\text{-max}(u)$.

4.5. If u and v are in $A(k)$ and u is v's left neighbor, then for each process x with $u < x \leq v$, we have $k\text{-max}(x) \leq k\text{-max}(v)$.

As a consequence of these properties it is straightforward to verify that the following modifications will not affect the correctness of the algorithm.

Each M2 message initiated in the kth phase will now carry, as well as its usual max value, a counter which is initialized to $F(k + 2)$. When a passive process x receives an M2 message carrying max value i and counter value c, then if $i > k\text{-max}(x)$ and $c > 1$, x decrements the counter value, sets $k\text{-max}(x) = i$, and sends on the M2 message. If $i > k\text{-max}(x)$ and $c = 1$ then x sets $k\text{-max}(x) = i$, becomes active, and begins the $(k + 1)$st phase, sending out an M1 message. Finally if $i \leq k\text{-max}(x)$, the process x does nothing, thereby stopping the M2 message.

As before it is necessary to introduce a waiting state, into which active processes enter after initiating M2 messages. Waiting processes handle arriving M2 messages in the same way as passive processes, except that in the case $i \leq k\text{-max}(x)$ the process x becomes passive instead of doing nothing. From the next remark it can be seen that the case $i > k\text{-max}(x)$ and $c > 1$ can never arise when x is a waiting process.

It is straightforward to verify that the modified algorithm still has properties 4.1, 4.2, 4.3, 4.4, and 4.5, where now of course $A(k)$ and all the other notation refers to the modified algorithm. Thus by 4.1, the algorithm still terminates in at most $1/\log((1 + \sqrt{5})/2)\log n$ phases. As in the modified Algorithm B of Section 3, it is easy to see that because of the presence of counters in M2 messages, each M2 message in the kth phase carrying a value which survives into the next phase, travels distance exactly $F(k + 2)$. As a consequence, for any v in $A(k + 1)$ where k is odd, if u is the process

in $A(k)$ which initiated the M2 message received by v in the kth phase, then $d(u, v) = F(k + 2)$.

For $k \geq 2$ and even, and v a process in $A(k)$, we say that v is cramped if the distance between v's left neighbor and v is less than $F(k + 2)$. Let $C(k)$ denote the cramped processes in $A(k)$. First notice that if v is in $C(k)$ then v is not in $A(k + 1)$, since if u is v's left neighbor in $A(k)$, by 4.1 we have k-max$(u) > k$-max(v), and hence v becomes passive during the kth phase. This shows that $|C(k)| \leq |A(k)| - |A(k + 1)|$, hence $|C(k)| + |C(k + 1)| + \cdots + |C(p)| \leq |A(k)|$, where p is the number of the last even-numbered phase.

LEMMA 4.6. *Let v be a process in $A(k) \setminus C(k)$ where k is even and ≥ 2, and let $\ell(v)$ be v's left neighbor in $A(k)$. Also, let $D = \{x: \ell(v) \leq x < v$ such that x sent an M2 message in phase $k - 1\}$. Then*

$$|D| \leq (2F(k + 1)/(F(k + 3) - 1))\, d(\ell(v), v).$$

Proof. Let u be the process in $A(k - 1)$ which initiated the M2 message which v received in phase $k - 1$. If there are no processes x in $A(k - 1)$ with $\ell(v) \leq x < u$ then $|D| \leq F(k + 1)$ since every M2 message in the $(k - 1)$st phase travels distance at most $F(k + 1)$, and since $d(\ell(v), v) \geq F(k + 2) \geq F(k + 3)/2$, we are done.

Thus suppose we have $\ell(v) \leq v(1) < \cdots < v(s) = u$ with each $v(i)$ in $A(k - 1)$ for $i = 1, 2, \ldots, s$ and $s \geq 2$, such that no other processes in $A(k - 1)$ lie between $\ell(v)$ and u. Since $\ell(v)$ is v's left neighbor in $A(k)$, we must have $(k - 1)$-max$(v(i)) < (k - 1)$-max$(v(i + 1))$ for $i = 1, 2, \ldots, s - 1$. Thus for each i, the M2 message initiated by $v(i)$ could not travel further than origin$((k - 1)$-max$(v(i + 1)))$. By 4.2 we see that between $v(i)$ and $v(i + 1)$ there are at least $F(k) - 1$ processes which do not send an M2 message, and as before, at most $F(k + 1)$ processes which do send an M2 message. Also no processes between $\ell(v)$ and $v(1)$ send an M2 message. This shows that $|D| \leq sF(k + 1)$ and that $d(\ell(v), v) \geq (s - 1)(F(k) - 1) + |D|$. Thus

$$\begin{aligned} |D| &\leq (sF(k + 1)/((s - 1)(F(k) - 1) + sF(k + 1)))d(\ell(v), v) \\ &\leq 2(F(k + 1)/(F(k + 3) - 1))\, d(\ell(v), v) \end{aligned}$$

since $(s - 1)(F(k) - 1) + sF(k + 1) \geq (s/2)F((k + 3))$ for $k \geq 2$. □

COROLLARY 4.7. *The number of M2 messages sent in phase $k - 1$ is at most*

$$|C(k)|\, F(k + 1) + (n - |C(k)|\, F(k + 1))(2F(k + 1)/(F(k + 3) - 1)).$$

Proof. Let v be in $C(k)$, let $\ell(v)$ be v's left neighbor, and let u be the process in $A(k - 1)$ which initiated the M2 message received by v in phase

$k-1$. We claim that there is no x in $A(k-1)$ with $\ell(v) \le x < u$. This is easy to see, since if there were such an x we would have $d(\ell(v), v) \ge d(x, u) + d(u, v) \ge F(k) + F(k+1) \ge F(k+2)$, and hence v would not be in $C(k)$. This shows that the total number of M2 messages sent in the $(k-1)$st phase by processes which lie between some process v in $C(k)$ and its left neighbor $\ell(v)$ in $A(k)$, is at most $F(k+1)\,|\,C(k)\,|$. Combining this with the preceding lemma yields the desired bound. □

Corollary 4.7 shows that the total number of M2 messages sent is bounded by $n\Sigma\{2F(k+1)/(F(k+3)-1) : k \ge 2 \text{ and even}\} + \Sigma\{((F(k)-1)/(F(k+3)-1))F(k+1)\,|\,C(k)\,| : k \ge 2 \text{ and even}\}$. The first of these sums is easy to estimate since $F(j) = \phi^j/\sqrt{5} + O(1)$, where $\phi = (1+\sqrt{5})/2$. Thus the first sum is bounded by $0.382\,pn + O(n)$ where, as before, p is number of the last even-numbered phase. The next lemma gives an easy estimate of the second sum, which can be used to obtain a bound of $1.376n \log n + O(n)$ on the total number of messages sent. We will also sketch how to obtain a better bound on the second sum, which yields the desired $1.356n \log n + O(n)$ bound on the total number of messages.

LEMMA 4.8. *The sum*

$$S = \Sigma\{((F(k)-1)/(F(k+3)-1))F(k+1)\,|\,C(k)\,| : k \ge 2 \text{ and even}\}$$

$$\le 0.07295pn + O(n).$$

Proof. Since the function $((F(k)-1)/(F(k+3)-1))F(k+1)$ is strictly increasing in k, and since $|\,C(k)\,| + |\,C(k+2)\,| + \cdots + |\,C(p)\,| \le |\,A(k)\,| \le \lceil n/F(k+1)\rceil$, it is clear that an upper bound for S can be obtained by setting $|\,C(k)\,| = 1 + \lceil n/F(k+1)\rceil - \lceil n/F(k+3)\rceil = 1 + n(F(k+2)/F(k+1)(F(k+3))$. Using the fact that $F(j) = \phi^j/\sqrt{5} + O(1)$ and that $p \le \log_\phi n + O(1)$, we see that $S \le pn/(2\phi^4) + O(n) \simeq 0.07295pn + O(n)$. □

COROLLARY 4.9. *The total number of messages sent is bounded by* $1.376n \log n + O(n)$.

Proof. From Lemma 4.8 and the remarks preceding it we see that the total number of M2 messages is bounded by $0.455pn + O(n)$. Since the total number of M1 messages is $0.5pn + O(n)$, the total number of messages is bounded by $0.955pn + O(n)$. Now since $F(p+1) \ge n$ as observed before, this implies that $p \le (\log n/\log \phi) + O(1)$, and hence $p \le 1.4405 \log n + O(1)$. Combining these together yields the stated bound. □

We now indicate how to improve the bound on the sum S in Lemma 4.8. For each even $k \geq 2$ and v in $C(j)$ for j an even number greater than k, let $u(k, v)$ be the process in $A(k)$ such that $k\text{-max}(u(k, v)) = j\text{-max}(v)$. As $u(k, v)$ cannot be in $C(k)$, the distance from $u(k, v)$'s left neighbor in $A(k)$ to $u(k, v)$ must be at least $F(k + 2)$. This yields the inequality $F(k + 1) | C(k) | + F(k + 2)(| C(k + 2) | + | C(k + 4) | + \cdots + | C(p) |) \leq n$. It is quite straightforward to check that if for some $k > 2$ we have $F(k + 1) | C(k) | + F(k + 2)(| C(k + 2) | + | C(k + 4) | + \cdots + | C(p) |) < n$, then by increasing $C(k)$ to make equality hold, and decreasing $C(k - 2)$ by the appropriate amount so that $F(k - 1) | C(k - 2) | + F(k)(| C(k) | + | C(k + 2) | + \cdots + | C(p) |) \leq n$, the rest of the inequalities still hold, and the value of the sum S is increased. Thus an upper bound on S can be obtained by solving the equations $F(k + 1) | C(k) | + F(k + 2)(| C(k + 2) | + | C(k + 4) | + \cdots + | C(p) |) = n$ for $\{| C(k) |\}$ and substituting them into S. This method yields $S \leq 0.0591pn + O(n)$. Combining this with the other bounds as before achieves the bound of $1.356n \log n + O(n)$ on the total number of messages.

REFERENCES

1. J. E. Burns, "A Formal Model for Message Passing Systems," TR-91, Indiana University, September 1980.
2. E. CHANG AND R. ROBERTS, An improved algorithm for decentralized extrema-finding in circular configurations of processes, *Comm. ACM* **22** (1979), 281–283.
3. R. G. GALLAGER, P. A. HUMBLET, AND P. M. SPIRA, "A Distributed Algorithm for Minimum Spanning Tree," MIT, LIDS-P-906-A, October 1979.
4. D. S. HIRSCHBERG AND J. B. SINCLAIR, Decentralized extrema-finding in circular configurations of processes, *Comm. ACM* **23** (November 1980).
5. A. ITAI AND M. RODEH, "The Lord of the Ring or Probabilistic Methods for Breaking Symmetry in Distributive Networks," RJ-3110, IBM Research Report, April 1981.
6. G. LELANN, "Distributed Systems—Toward a Formal Approach," Information Processing 77, pp. 155–160, North-Holland, Amsterdam, 1977.
7. G. L. PETERSON, An $O(n \log n)$ unidirectional algorithm for the circular extrema problem, to appear in *Transactions on Programming Languages and Systems*.