Technical Note
Operating Systems

R. Stockton Gaines
Editor

Decentralized Extrema-Finding in Circular Configurations of Processors

D.S. Hirschberg and J.B. Sinclair
Rice University

This note presents an efficient algorithm, requiring $O(n \log n)$ message passes, for finding the largest (or smallest) of a set of n uniquely numbered processors arranged in a circle, in which no central controller exists and the number of processors is not known a priori.

Key Words and Phrases: decentralized algorithms, distributed system, operating systems

CR Categories: 4.32, 4.35, 5.25, 5.32

Introduction

We are given n processors that are loosely coupled in a circular arrangement and work asynchronously. Each of the processors has an associated unique value (of which it alone is aware) and none of the processors has a priori knowledge of the number of processors in the circle. The problem is to designate by consensus a unique processor from the circle. The total number of data transmissions (messages passed) among the n processors is a measure of the complexity of a solution algorithm.

LeLann [2] presented an algorithm that requires $O(n^2)$ messages. Chang and Roberts [1] proposed an improved algorithm that requires only $O(n \log n)$ messages on the average but, in the worst case, still requires $O(n^2)$ messages. Both of the above algorithms assume the capability of each processor to pass a message "to the left" in a global sense.

We consider the case in which the processors can pass messages in either or both directions, that these directions are distinguished, that processors can detect from which direction a received message originated, but that "left" may not mean the same to all processors. We propose an algorithm that requires $O(n \log n)$ messages in the worst case. The algorithm as given elects the processor with the highest value.

In the algorithm given below, a processor can initiate messages in both directions by a *sendboth* directive. A processor can pass a (possibly modified) message in a circular manner by a *sendpass* directive. A processor can send a responsive message back in the direction from which that processor received a message by a *sendecho* directive.

The Algorithm

```
To run for election:
    status ← "candidate"
    maxnum ← 1
    WHILE status = "candidate" DO
        sendboth ("from", myvalue, 0, maxnum)
        await both replies (but react to other messages)
 IF either reply is "no" THEN status ← "lost"
        maxnum ← 2*maxnum
    OD
On receiving message ("from", value, num, maxnum):
    IF value < myvalue THEN sendecho ("no", value)
    IF value > myvalue THEN DO
        status ← "lost"
        num ← num + 1
        IF num < maxnum THEN sendpass ("from", value, num,
        maxnum)
            ELSE sendecho ("ok", value)
    OD
    IF value = myvalue THEN status ← "won"
On receiving message ("no", value) or ("ok", value)
    IF value ≠ myvalue THEN sendpass the message
        ELSE this is a reply the processor was awaiting
```

The processors initiate messages that are passed in both directions along paths of predetermined lengths (which are successive powers of 2). Processors on the path read the message. If a processor determines, from reading the message, that it cannot win the election, then it will pass the message and it will not *initiate* any further messages of its own. If a processor determines that the message originator cannot win the election, it echos back a message informing the originator of this fact. The processor at the end of the path echos back a message informing the originator that all processors along the path defer to the originator.

A processor receiving its own message will have won the election since all other processors in the circle will have deferred to it. It is then a simple matter for the winner to send a message informing all other nodes that the election has been satisfactorily concluded.

Not accounted for in the algorithm as written (but easily added) is the possibility of a processor being unaware of an election in progress. Such a processor, upon receiving a message, would *then* be aware of the election and, if not beaten by the originator of the message it just received, would become a candidate.

If a communication link between two nodes should fail or if a node should fail, then unless the winner has

D.S. Hirschberg's research was supported in part by the National Science Foundation under grant MCS80-03431; J.B. Sinclair's research was supported in part by the National Science Foundation under grant MCS80-04107.

Authors' present addresses: D.S. Hirschberg and J.B. Sinclair, Department of Electrical Engineering, Rice University, Houston, TX 77001.

already been determined, all other nodes will eventually enter the state in which they await a reply.

Complexity Analysis

A processor, x, initiates messages along paths of length 2^i only if it is not defeated by a processor within distance 2^{i-1} (in either direction) from x. Within any group of $2^{i-1}+1$ consecutive processors, at most one can initiate messages along paths of length 2^i. Although possibly all n processors will initiate paths of length 1, at most $\lceil n/2 \rceil$ (read ceiling of $n/2$) processors will initiate paths of length 2, at most $\lceil n/3 \rceil$ of length 4, at most $\lceil n/5 \rceil$ of length 8, etc.

A processor initiating messages along paths of length 2^i causes messages to emanate in both directions, and return. At most $4*2^i$ messages will be passed as a result of that initiation. The sum total of all messages passed is therefore at most

$$4*(1*n + 2*\lceil n/2 \rceil + 4*\lceil n/3 \rceil + 8*\lceil n/5 \rceil + \ldots + 2^i*\lceil n/(2^{i-1}+1) \rceil + \ldots).$$

Each of the terms within the parentheses is less than $2n$. There are no more than $1 + \lceil \log n \rceil$ terms. (No processor will pass messages along paths of length $2n$ or greater since, once a processor initiates paths of at least n length and the message is acceptable all the way around the circle, the processor wins and stops initiating messages.) Thus, the total number of messages passed is less than $8n + 8\lceil n \log n \rceil = O(n \log n)$.

If one defines the time complexity to be the minimum time required for the completion of an election assuming as much message transmission overlap as possible, the worst case time complexity can easily be shown to be linear in the number of processors. The exact formula is

$$\text{Time} = 2*(1 + 2 + 4 + \ldots + 2^i + \ldots + n).$$

When n is an exact power of 2 (the best case), Time $\doteq 4n - 2$; when n is one more than an exact power of 2 (the worst case), Time $= 6n - 6$. Thus, the savings in worst-case message passages is paid for by an increase in the wall time.

We 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. Recently, Burns has shown [3] that $n \log n$ is asymptotically optimal.

Received 6/79; revised 5/80; accepted 7/80

References

1. Chang, E., and Roberts, R. An improved algorithm for decentralized extrema-finding in circularly configuraiions of processes. *Comm. ACM 22*, 5 (May 1979), 281–283.
2. LeLann, G. Distributed systems—Towards a formal approach. Inform. Proc. 77, North-Holland Pub. Co., 1977, Amsterdam, pp. 155–160.
3. Burns, J.E. A formal model for message passing systems. Tech. Rep. No. 91, Comptr. Sci. Dept., Indiana Univ., May 1980.

Computer Architecture and Systems

J.P. Hayes
Editor

Design of a LISP-Based Microprocessor

Guy Lewis Steele Jr. and
Gerald Jay Sussman
Massachusetts Institute of Technology

We present a design for a class of computers whose "instruction sets" are based on LISP. LISP, like traditional stored-program machine languages and unlike most high-level languages, conceptually stores programs and data in the same way and explicitly allows programs to be manipulated as data, and so is a suitable basis for a stored-program computer architecture. LISP differs from traditional machine languages in that the program/data storage is conceptually an unordered set of linked record structures of various sizes, rather than an ordered, indexable vector of integers or bit fields of fixed size. An instruction set can be designed for programs expressed as trees of record structures. A processor can interpret these program trees in a recursive fashion and provide automatic storage management for the record structures.

We discuss a small-scale prototype VLSI microprocessor which has been designed and fabricated, containing a sufficiently complete instruction interpreter to execute small programs and a rudimentary storage allocator.

Key Words and Phrases: microprocessors, large-scale integration, integrated circuits, VLSI, list structure, linked lists, garbage collection, storage management, direct execution, high-level language architectures, interpreters, tail recursion, LISP, SCHEME

CR Categories: 4.13, 4.21 4.22, 4.34, 6.21, 6.22, 6.33

This work was supported in part by the National Science Foundation under Grant MCS77-04828, and in part by Air Force Office of Scientific Research Grant AFOSR-78-3593.

Authors' present addresses: G.L. Steele, Jr., Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA 15213; G.J. Sussman, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.