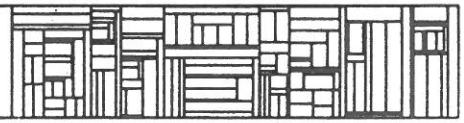
ISSN 0105-8517

Symmetric Distributed Termination

Ole Eriksen
Sven Skyum

DAIMI PB - 189
January 1985

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade - DK 8000 Aarhus C - DENMARK
Telephone: 06 - 12 83 55



Introduction

In this paper we present a simple algorithm for deciding when to terminate a distributed computation. Former solutions to the termination problem, e.g. [2, 4], are restricted to rings and undirected connected networks, resp., and they rely on the existence of a special master processor present in the network. The class of configurations for which our algorithm is applicable is the most general possible, namely the class of configurations, where the underlying (directed) networks are strongly connected. If the network is not strongly connected then there exists a group of processors which cannot send messages to the rest and it becomes impossible to detect when to terminate. Furthermore, the algorithm does not require a special master processor acting differently from the others. The only information needed is an upper bound on the diameter of the network (the number of processors, for example). The protocol for all processors is in other words identical.

In [3] it is shown that the existence of a master processor in a strongly connected network makes it possible to calculate the number of processors in the configuration while the opposite is impossible in general (see [1]). Our algorithm therefore generalizes present algorithms with respect to both the class of configurations for which it is applicable and the initial information available.

The Model and the Termination Problem

A *configuration* is a collection of processors each of which is connected to some other processors by one-way communication lines. The underlying network (or graph) is a strongly connected (directed) network where the nodes are associated with the processors and the edges with the communication lines. Each communication line is a buffer containing messages, namely those sent but not yet received. Writing (sending) and reading (receiving) messages happens *instantaneously*, i.e. we consider buffer operations to be indivisible. *Computations are asynchronous* and since communication goes via buffers, *the communication is asynchronous* as well. Finally, we assume no processor to stop computing.

Consider a configuration in which each processor can be either *active*, *passive* or *terminated* and for which (a) through (d) below hold for any computation:

(a) All processors are either active or passive initially.

(b) An active processor can send *activation messages* to any processor *regardless of the existence of a communication line between them.*

The motivation for, in general, allowing activation to happen independently of communication lines is to be able to use a termination algorithm for special networks as ringshaped in [2] on a larger class of networks. If a network contains a known Hamiltonian circuit (or ring), the circuit can be used for termination while activation messages can follow all edges in the larger network (potentially there can then be a communication line between each pair of processors). Since our algorithm applies to all strongly connected networks, the motivation for "free" activation becomes less transparent.

(c) A passive processor receiving an activation message becomes active.

(d) An active processor may become passive spontaneously.

Since only active processors can "activate" other processors, no more computing is done when no more processors are active.

The *termination problem* is to detect when all processors are passive and then let them all terminate.

The following notation is used in the next section.

Let $G = (V, E)$ be a network. For $p, q \in V$, $dist(p, q)$ denotes the length of the shortest directed path between p and q. The diameter of G is identified as

$$\max_{p,q \in V} \{dist(p, q)\}$$

We call $(p, q) \in E$ an *inbuffer* for q (the *receiver*) and an *outbuffer* for p (the *sender*). Finally, we use the word processor instead of vertex.

A Solution to the Termination Problem

We present a protocol which in a sense makes the configuration work synchronously. We achieve this by introducing a time concept defined locally. This time scale is smooth (or continuous) enough for the individual passive processors to "know" that if they have not heard from any active

processor for some "time", then there are no more active processors in the configuration. The amount of time any passive processor has to wait is given by a linear function of the diameter of the network.

The time concept is introduced by splitting up the computation for a processor into parts separated by a broadcast to all outbuffers followed by a reading from each inbuffer.

The protocol has the following form:

```
    :
  broadcast a message.
cycle
  await that no inbuffers are empty;
  read one message from each inbuffer;
    :
  broadcast a message;
end;
```

Figure 1

We number the broadcasts from 0 and the reads from 1.

Since the processors are asynchronous they can do any finite computation between the i'th reading and the i'th broadcast. We call the cycle starting with the i'th read and finishing with the i'th broadcast for phase i or p-phase i if we wish to emphasize the processor (p).

The duration of a phase in real time differs from processor to processor and phase to phase. The time during which a processor is active is contained in a single phase. Thus an active processor (participating in the underlying distributed computing) does not take part in the termination process apart from holding messages back.

Note that the protocol ensures that the maximal number of messages ever contained in a buffer is bounded by the length of the shortest cycle containing the corresponding edge in the network (see [5]).

The following Lemma shows that the protocol makes the configuration act synchronously in some sense.

Lemma 1 If $p, q \in V, p \neq q$, then p-phase k happens before q-phase $k + dist(p, q)$ for any $k \geq 1$.

Proof The result follows by induction on $dist(p, q)$ by observing that if $(p, q) \in E$ then q cannot enter phase $k + 1$ before it has read k messages from inbuffer (p, q) and p has to finish phase k to have written k times on outbuffer (p, q).

□

The termination protocol is based on two kinds of messages flowing around in the network, namely *ready*-messages and *warning*-messages with an "age" attached to them. When a processor p changes state from active to passive, it sends a warning to the other processors by broadcasting a warning-message of age one. This warns the other processors that some processors might have been activated by p during its last phase. Passive processors send on *warning*-messages which are not too old and add one to the minimal age received. If a *warning*-message becomes too old, a passive processor treats it as a ready-message. This ensures that a processor does not react on the same warning over and over again. When a passive processor has only received *ready*-messages for some time, it knows it can terminate and does so.

In Fig. 2 the termination protocol is shown in a more detailed form. Two constants L and n occur in the protocol. We show that the protocol is correct for any network with diameter d if $L \geq 2d$ and $n \geq d + 1$.

We consider reception of an *activation*-message as an interrupt. **disable** closes for reception of interrupts and **enable** opens. Hence a piece of program enclosed by **disable** and **enable** is indivisible.

The key Lemmas are the following two.

Lemma 2 Consider a network of diameter at most d. If processor p broadcasts (*warning*, 1) in p-phase k, then if $n \geq d$ each processor $q \neq p$ reads (*warning*, s) for some $s \leq dist(p, q)$ in q-phase $k + dist(p, q)$.

Proof Let $p = p_0, p_1, \ldots, p_r = q$ be a path from p to q of length $r = dist(p, q)$. It is easily shown by induction on t that for $1 \leq i \leq t \leq r$


```
active := true;
l := 0;
broadcast(warning, 1);
cycle
    await that no inbuffers are empty;
    disable;
    read one message from each inbuffer;
    enable;
        if l = L then terminate endif;
    disable;
    if active then
        broadcast(warning, 1);
        active := false;
        l := 0
    else
        if all inputs were (warning, n) or ready then
            broadcast(ready);
            l := l + 1
        else
            m := min{i | (warning, i) was read};
            broadcast(warning, m + 1);
            l := 0
        endif
    endif;
    enable;
endcycle;
```

Figure 2

processor p_i reads (*warning*, s) for some $s \leq i$, from inbuffer (p_{i-1}, p_i) in p_i-phase $k + i$ and then broadcasts (*warning*, s') for some $s' \leq s + 1 (\leq i + 1)$.

□

Lemma 3 Consider a network with diameter d. If processor p is active in p-phase $k > d$ then for all l, $1 \leq l \leq k - d$ there exists a processor q which is active in q-phase s for some $s \in \{l, l + 1, \ldots, l + d\}$.

Proof Let r be the first processor entering phase $l + d$ and let t be the time it does. Such one exists since $k \geq l + d$ and p entered p-phase k. From Lemma 1 we deduce that at time t each processor is in some phase s, where $s \in \{l, l + 1, \ldots, l + d\}$. If at that time all processors were passive, p could not be active in p-phase $k \geq l + d$ since it does not enter phase k until after time t.

□

Theorem 4 The termination protocol is correct for networks with diameter d if $L \geq 2d$ and $n \geq d + 1$.

Proof Assume that at time t processor p is going to enter a phase with $l = L$ and some processor q is active at the same time. Then p has entered phase $k + L + 1$ for some $k \geq 1$ and has not read any (*warning*, i) messages for any $i < n$ in phase $k + 1, k + 2, \ldots, k + L - d$. Then by Lemma 2 no processor r was active in r-phase $k + 1, k + 2, \ldots, k + L - d$ and by Lemma 1 we get that q was active in q-phase s, for some $s \geq k + L + 1 - d \geq k + d + 1$. This contradicts Lemma 3, since $L - d \geq d$.

Hence no processor terminates before all processors are either passive or terminated. On the other hand when no more active processors are present, all processors will terminate since *warning*-messages will eventually disappear when they become older than n.

□

Acknowledgement

We would like to thank our colleague Jørgen Staunstrup for many fruitful discussions.

References

[1] Augluin, D.: Local and global properties in networks of processors. *Proceedings of the 12th Ann. ACM STOC, 1980, pp. 82-93.*

[2] Dijkstra, E.W., Feijen, W.H.J., van Gasteren, A.J.M.: Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters, vol. 16, 1983, pp. 217-219.*

[3] Eriksen, O., Skyum, S., Staunstrup, J.: Distributed Algorithms for Network Problems. Manuscript.

[4] Francez, N.: Distributed termination. *ACM Toplas vol. 2, 1980, pp. 42-55.*

[5] Genrich, H.J., Thiagarajan, P.S.: A theory for bipolar synchronisation schemes. *Technical Report, PB-158, Computer Science Department, Aarhus University, Denmark.*