Chapter 6.
GLOBAL COMPLETION DETECTION

6.1. Problem Setting
Let us consider a network with n processors, each executing an application code (AC) that needs local computation and communication with other application codes over the network. There is no global clock, just local ones. Each AC can be found either in the active or passive state. In the former case, the AC is currently executing, it may receive and it can send messages. In the latter, the AC has reached local completion and stopped. In this state, the AC may only be resumed upon receiving new messages. There is no global knowledge about the state of the entire system. Initially, all the ACs are started in the active state. The problem consists in devising a distributed algorithm that makes the processors cooperate to detect global completion. Global completion is defined as a state in which: (1) all the ACs have been locally completed, and (2) all the messages sent have reached their targets so that no AC can possibly be resumed.

6.2. Detecting Completion via Counting

Figure 1. A collect session stating wrong global completion.

J.-M. Adamo, Multi-Threaded Object-Oriented MPI-Based Message Passing Interface
Many algorithms for detecting distributed completion have been proposed in the literature. The first solutions were given by Francez and Dijkstra [FRA 80] [DIS 80]. Many other algorithms have been proposed that are listed in Mattern's paper [MAT 87] (see also [RAY 92]). We choose Mattern's [MAT 87] that we found simple and potentially well suited to efficient implementations on arbitrary network topologies. In this algorithm, each processor \( p_i \) maintains two counters, say \( s_i \) and \( r_i \), respectively counting the number of messages sent and received by this processor. From time to time (see section 6.3 for details), one processor, say \( p_0 \), collects a set of triples: \((s_i, r_i, b_i)\) for \( 0 \leq i < n \) where \( b_i \) is a boolean associated to the current status of the AC run by processor \( p_i \). Boolean \( b_i \) takes the value true if the AC is passive or false otherwise. Each collect operation gathers data by using point-to-point communication within a collective process that will be detailed later. Let us denote:

\[
S = \sum_i s_i, \quad R = \sum_i r_i, \quad B = \bigland_i b_i,
\]

and assume that some collect session brings back values such that:

\[
S = R, \quad \text{and } B = \text{true}.
\]

Should we conclude from these results that global completion has been reached? The case illustrated in Figure 1 shows this is not true (see also [RAY 92]). As there is no global clock, no assumption can be made by processor \( p_0 \) about the time at which data were locally collected on the other processors. This might have happened along the cut denoted as (C) in Figure 1 for which the conditions above are met although global completion is not achieved. The problem obviously comes from the non-simultaneity of local collect operations. A solution is given by Mattern's proposition below.

**Proposition.**

Let us consider two successive non-overlapping collect sessions denoted as (1) and (2). Let \((S_1, R_1, B_1)\) and \((S_2, R_2, B_2)\) be the aggregates of data collected in (1) and (2) respectively. If \( B_2 = \text{true} \) and \( R_1 = S_2 \) then global completion has been reached.

The result is quite intuitive, the proof is as follows (see [RAY 92] for details). Assume there is some observer that can look over the entire network and can assign a global time value to each collected data. Let us denote: \( t_{10}, \ldots, t_{1n-1} \) the time at which the data are collected on each processor in the first collect session, \( t_1 \) and \( z_1 \) the time at which the first collect starts and is completed. Let us denote \( t_{20}, \ldots, t_{2n-1} \), \( a_2 \) and \( z_2 \) similar data for the second collect session. Let us additionally denote \( r_i(t) \) and \( s_i(t) \) for any processor \( i \), \( R(t) \) and \( S(t) \) the value of counters and aggregates at time \( t \). Note that:

1. counters and aggregates behave as non-decreasing functions of time.
2. \( a_1 \leq t_1 \leq z_1, z_1 \leq a_2 \) (collects do not overlap) and \( a_2 \leq t_2, z_2 \).
3. \( S_1 = R \) actually means \( R(z_1) = S_2 \) with the latest notations, which implies by definition of collect sessions that:

\[
\sum_i r_i(t_1) = \sum_i s_i(t_2).
\]

It follows from 1., 2., and 3. that: