Efficient Synchronization of Asynchronous Processes

Sandeep Lodha[1], Punit Chandra[2], Ajay Kshemkalyani[2], and Mayank Rawat[2]

[1] Riverstone Networks Inc.
Santa Clara, CA 95054, USA.
[2] EECS Department, University of Illinois at Chicago
Chicago, IL 60607-7053, USA.

Abstract. Concurrent programming languages including CSP and Ada use synchronous message-passing to define communication between a pair of asynchronous processes. This paper presents an efficient way to synchronize these processes by improving on Bagrodia's algorithm that provides binary rendezvous. Simulation results are presented to show the better performance of the optimized algorithm for two cases - the case where the interaction set is composed of all possible pairs and the case where the set of next allowable interactions is of cardinality one. For the latter, the optimized algorithm also improves upon the best case delay for synchronization. The client-server computing model, the producer-consumer interaction, and interaction between processes executing parallelized tasks represent some broad classes of computations which can leverage the proposed improvements.

1 Introduction

Concurrent programming languages including CSP [6] and Ada [1] use synchronous message-passing to define communication between a pair of asynchronous processes. Although this synchronous style of programming compromises the possible concurrency in the computation, it offers simplicity in program design and verification. The synchronous programming style is also known as binary rendezvous which is a special case of multiway rendezvous, also known as the barrier or committee coordination problem [5].

The generalized alternate command of CSP allows a process to select any one of several binary rendezvous, identified by the *interaction set*, for the next interaction or rendezvous. Several algorithms implement this rendezvous [3,8,7,9]. Buckley and Silberschatz [4] presented four criteria to determine the "effectiveness" of algorithms that implement this construct. Using these criteria, they pointed out some major drawbacks of previously published algorithms and presented an algorithm that meets the criteria. Bagrodia [2] came up with an algorithm that was simpler and more efficient than [4]. This paper describes an algorithm that improves upon the message overhead of Bagrodia's algorithm, and presents simulation results for the same. Section 2 describes the proposed enhancements. Section 3 gives the results of the simulation. Section 4 concludes.

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 352–357, 2001.

2 Bagrodia's Algorithm and Proposed Enhancements

2.1 Bagrodia's Algorithm

Bagrodia's algorithm associates a unique token with each synchronization, also known as interaction. The token contains the ProcessIDs of the two processes involved in the interaction. When some process P_i becomes $IDLE$, it determines if an interaction (P_i, P_j) with process P_j can be executed by requesting that interaction. An $IDLE$ process requests interactions from its interaction-set in increasing order of priority. A process may request only those interactions for which it possesses the corresponding token. When P_i requests an interaction (P_i, P_j), it sends the corresponding token to P_j. A process may request at most one interaction at any time.

On receiving a token, a process P_j may either commit to the corresponding interaction, refuse to do so, or delay its response. If P_j is $IDLE$, it commits to the interaction. A process commits to an interaction by sending the token back to the requesting process. On the other hand, if P_j is $ACTIVE$, it refuses the interaction. A process refuses an interaction by capturing the token and sending a $CANCEL$ message to the requesting process. This implies that the process that last refused an interaction has the responsibility to initiate the next request for the interaction. A process that has requested an interaction but has not received a response to its request is a REQ process. A REQ process may receive (zero or more) requests for other interactions, before receiving a response to its own request. A REQ process P_j that receives a token for another interaction E_k delays the request E_k if priority of E_k is more than that of the interaction P_j is currently requesting, otherwise it refuses the interaction and sends a $CANCEL$ message. This prevents deadlocks in the system. If a REQ process P_i delays an interaction, the algorithm guarantees that P_i will either commit to its requested interaction or to the delayed interaction. Thus, it is only necessary for a process to delay at most one interaction. Tokens received by a REQ process that has delayed an interaction can immediately be refused by the process, irrespective of the relative priority of the two interactions.

2.2 Proposed Enhancements

We observe the following two drawbacks of Bagrodia's algorithm that cause some inefficiencies, and propose improvements to overcome them.

- First, when a process that is committed to an interaction or is ACTIVE receives a token from another process, it sends a CANCEL to that process and later bears the responsibility of initiating the interaction with that process. This leads to a total of four messages to set up the interaction. There is a wide range of applications for which the interaction set of at least one of the processes participating in an interaction is one. It is unnecessary to send a CANCEL to such a process and later try to reestablish the interaction. The proposed improvement is that instead of sending the CANCEL, the token

requesting the interaction can be queued up and later responded to, thereby cutting down on the message overhead.
Some classes of applications that have an interaction-set size of one are described here. The producer-consumer problem is one example where the producer's interaction-set size is one. The producer needs to interact only with the buffer-process. So once the producer is ready with the data, it cannot proceed unless it delivers the data to the buffer-process. Even if the buffer-process sends a CANCEL message to the producer, the producer cannot proceed. In such cases, we can avoid CANCEL messages by making the producer block on the buffer-process. Client-server applications form another class where this proposed improvement is useful. A client in the client-server model has to interact only with the server; thus all clients have an interaction-set size of one. A client that wants to synchronize (interact) with the server cannot proceed unless synchronized. In such applications, we can avoid CANCEL messages by making clients block on the server. Applications that have a high degree of parallelism form another class where the proposed improvement is particularly useful. All worker processes have to interact only with the master process. Thus all worker processes have an interaction-set size of one. Divide-and-conquer class of problems is an example where the worker processes, when done with the assigned computation, cannot proceed unless synchronized with the central process. In such applications, one can avoid CANCEL messages by making worker processes block on the central process.

– The second drawback of Bagrodia's algorithm is that a process that is ready to synchronize cannot initiate the synchronization if it does not have the token. If both processes involved in an interaction have a token each, then this process can initiate the interaction. Its token could then be kept pending at the other end until that process was ready to perform the interaction, at which time, only a single message transmission overhead would be incurred to complete the interaction setup. This scheme is the proposed enhancement. This scheme also increases concurrency in the system when both processes become ready to interact at the same physical time, and reduces the best case delay for synchronization, as explained below. Thus, each interaction can be assigned two tokens, one for each partner.

The proposed algorithm is an extension to Bagrodia's algorithm and addresses the aforementioned drawbacks. In the proposed algorithm, there are either two tokens or a unique token for each interaction, depending on the interaction-set size. There is a unique token for interactions between P_i and P_j if both P_i and P_j have an interaction-set size of more than one. This avoids the danger of extra CANCEL messages. For a pair of processes P_i and P_j such that one of the processes, say P_i (P_i could be the client/worker/producer process in the application), has an interaction-set size of one (P_i always interacts with P_j), there is either one token or two tokens for this interaction. It is up to the application process to decide. The number of messages is independent of the number of tokens in this case. For interactions between processes P_i and P_j,

which have two tokens – one with P_i, the other with P_j – either P_i or P_j can send a REQUEST carrying the correct token to the other. Both processes can send their REQUESTs concurrently. This increases concurrency in the system. If P_i sends a REQUEST to P_j and receives a concurrent REQUEST from P_j, then this REQUEST from P_j acts as a REPLY. On $P_j's$ side, $P_i's$ REQUEST serves as a REPLY. This improves the best case delay for synchronization (from round-trip delay to one-way delay).

As mentioned above, in Bagrodia's algorithm, an $ACTIVE$ process P_j always refuses the interaction. In the proposed algorithm, a REQUEST from process P_i, where P_i interacts only with P_j, is not refused. Instead, this request is kept in the $DelaySet_j$ of P_j. P_i should have a way to let P_j know that P_i wants to block on P_j (P_j is the only process that P_i interacts with). So in the proposed algorithm, we have an extra bit in the token, called $BlockFlag$, for this purpose. P_i sets it to $true$ if it wants to block on P_j, else it is $false$. This saves two messages. In the proposed algorithm, a CANCEL message is sent either to prevent a deadlock in the system or when the requesting process does not want to block.

3 Results

To demonstrate the efficiency of the optimized algorithm, we simulated both Bagrodia's algorithm (henceforth called BA) and the optimized BA. We compared the average number of messages of the optimized BA and the BA algorithms. The experiments were conducted using Intel Pentium III 866MHz computers with 128 Mb SDRAM, running RedHat Linux, with the algorithms implemented using C++.

The simulation model explored two cases - the first was a client-server type of communication pattern where clients had interaction set size of one, while the other had interactions between all pairs of processes (completely connected graph). The tokens were distributed randomly for both the cases in BA.

In the experiments, the message transfer time was assumed to be negligible. Thus, we abstracted away network latency. The active time - the time for which the process is in $ACTIVE$ state - was a configurable parameter. For a given number of processes, the simulation was repeated 10 times. The variance in the number of messages was negligible (see full paper for details).

Client-server case: As expected, the optimized BA gives better results. The mean number of messages per interaction for BA is nearly 3.06 while it is 2 for the optimized BA. So in the case of the client-server architecture, optimized BA is 35% more efficient than BA (Figure 1). The performance of the optimized BA is independent of the active time.

Fully connected case: The average number of messages for both BA and optimized BA varies with the active time parameter; the graph (Figure 2) shows the results for 10,000 clock ticks. The optimized algorithm shows only a slight improvement over Bagrodia's algorithm. This is largely because as the interactions complete, the system's communication pattern gradually moves towards a client-server system (where the size of the interaction set of clients is one), for

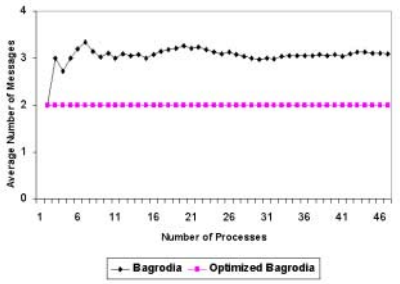

Fig. 1. The client-server case

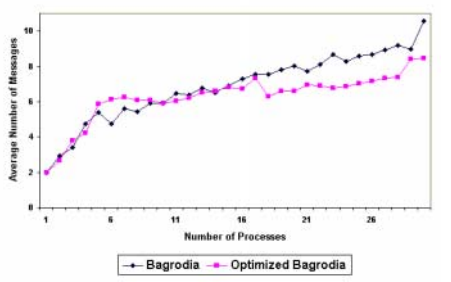

Fig. 2. The fully connected case

which the optimized BA algorithm has a lower message overhead. Furthermore, it appears that as the system size increases, the optimized BA seems to behave increasingly better than BA.

4 Concluding Remarks

Concurrent programming languages such as CSP and Ada use synchronous message-passing to define communication between a pair of asynchronous processes. We presented an efficient way to synchronize processes by improving on Bagrodia's algorithm which is one of the best known algorithms to implement synchronous communication between asynchronous processes. Simulation results showed that the optimized BA is always more efficient than BA. An efficiency gain of nearly 35% was achieved for a wide class of applications in which the set of next allowable interactions is of cardinality one.

Acknowledgements

This work was supported by the U.S. National Science Foundation grants CCR-9875617 and EIA-9871345.

References

1. Ada 95 Reference Manual (RM) Version 6.0: Intermetrics, Inc., January 1995. (URL http://lglwww.epfl.ch/Ada/rm95) 352
2. R. Bagrodia, Synchronization of asynchronous processes in CSP, *ACM TOPLAS*, 11(4):585-597, 1989. 352
3. A. Bernstein, Output guards and nondeterminism in communicating sequential processes, *ACM TOPLAS*, 2(2):234-238, April 1980. 352
4. G. Buckley, A, Silberschatz, An effective implementation of the generalized input-output construct of CSP, *ACM TOPLAS*, 5(2):223-235, April 1983. 352

5. M. Chandy, J. Misra, *Parallel Program Design: A Foundation,* Addison-Wesley, 1978. 352
6. C. A. R. Hoare, Communication sequential processes, *CACM*, 21(8):666-677, Aug. 1978. 352
7. F. Schneider, Synchronization in distributed processes, *ACM TOPLAS*, 4(2): 125-148, April 1982. 352
8. J. Schwarz, Distributed synchronization of communicating sequential processes, *Tech. Report, University of Edinburgh,* July 1978. 352
9. Van de Snepscheut, Synchronous communication between asynchronous components, *Information Processing Letters*, 13(3): 127-130, Dec. 1981. 352