A Technique for Execution of Written Values on Shared Variables

Parvinder S. Sandhu, Vijay K. Banga, Prateek Gupta, Amit Verma

Abstract—The current paper conceptualizes the technique of release consistency indispensable with the concept of synchronization that is user-defined. Programming model concreted with object and class is illustrated and demonstrated. The essence of the paper is phases, events and parallel computing execution. The technique by which the values are visible on shared variables is implemented. The second part of the paper consist of user defined high level synchronization primitives implementation and system architecture with memory protocols. There is a proposition of techniques which are core in deciding the validating and invalidating a stall page.

Keywords—synchronization objects, barrier, phases and events, shared memory

I. INTRODUCTION

RELEASE consistency[4] with user-definable high level synchronization primitives(RCHS) provides a pattern in which users can define their own synchronization primitives, called synchronization classes. RCHS also constrains the execution of synchronization primitives but any synchronization primitive that follows the paradigm can be used in this memory model. The paradigm provides:

- Better interface for generating algorithms
- Improves the performance of subsequent applications.

Primitives (RCHS) are designed for a software distributed shared memory system in a high latency network, where the cost of traditional atomic operations, for example fetch & add and busy waiting for the purpose of synchronization is considerable, and has an adverse effect on the performance[6] of the computation.

There are two types of objects those communicate with in RCHS:

- Shared memory
- Synchronization objects (see Fig. 1).

The shared memory consists of a pattern in memory as array (like) in virtual memory. Two types of operations are allowed in shared memory, read and write. Anything a process writes in shared memory may be visible to other processes. In addition to shared memory, processes can also communicate with each other via synchronization objects. Each process accesses synchronization objects only by calling operations defined in synchronization classes[6]. Synchronization objects can not access other shared objects. Each synchronization operation [7] may be annotated with following attributes, A: attributes(diagram 1)

Dr. Parvinder S. Sandhu is Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA

(Phone: +91-98555-32004; (Email: parvinder.sandhu@gmail.com).

Er. Prateek Gupta is Lecturer with Electronics & Communication Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA

Prof. Vijay K. Banga is Professor with Electronics & Communication Engineering Department, Amritsar College Of Engineering and Technology, Amritsar (Punjab)- INdia

Er. Amit Verma is Assistant Professor with Electronics & Communication Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA

Fig. 1. Programming Model
(for two object can extended for n object)

Fig. 2. Synchronization operation attributes

These attributes are used to define the visibility of the values in shared memory. Informally, the annotation release may be thought of as meaning that the process “puts” its
visible shared memory writes onto the synchronization object, and AC may be thought of as meaning that the process “gets” the visible writes from the synchronization object. For example,

Suppose that process \( p \) writes to shared variable \( X \) and then executes an operation provided by synchronization object \( S \) with attribute \( R \). \( S \) can see the newly written value of \( X \) (figure 3.4). If process \( q \) subsequently executes an operation of \( S \) with attribute \( acquire \), \( q \) obtains what \( S \) can see. So \( q \) can read the new value of \( X \) which \( p \) has written. Similarly, AC-R and R-AC may be thought of as a combination of the two. In the context of following discussion, release operations are operations with attribute \( release \), acquire release or release acquire. Acquire operations are operations with attribute acquire, release acquire, or acquire release.

![Fig. 3. Example with shared memory](image)

Suppose that process \( p \) writes to shared variable \( X \) and then executes an operation provided by synchronization object \( S \) with attribute \( R \). \( S \) can see the newly written value of \( X \) (figure 3.4). If process \( q \) subsequently executes an operation of \( S \) with attribute \( acquire \), \( q \) obtains what \( S \) can see. So \( q \) can read the new value of \( X \) which \( p \) has written. Similarly, AC-R and R-AC may be thought of as a combination of the two. In the context of following discussion, release operations are operations with attribute \( release \), acquire release or release acquire. Acquire operations are operations with attribute acquire, release acquire, or acquire release.

![Fig. 4. Operation with processes](image)
• **Receiving phase:** This phase of a synchronization object starts with a requesting event from a process, even though the requesting event is not in the phase.

• **Replying phase:** This phase of a synchronization object ends with a replying event.

Each phase of a process is identified by a unique time stamp. The time stamp of the initial phase is assumed with value 1 and the following phases attain a time stamp value incremented by one each time the next phase comes, i.e. time stamp gets values as say, a,a+1,a+2, and so on.

where $v,v_0= synchronization$ operation,

$$\text{Stamp gets values as say, } a,a+1,a+2, \text{and so on.}$$

An attribute of a synchronization object is $RA(RA=\text{acquire release})$. This case involves two

 edges: $1. R(R=\text{release})$

$2. A(A=\text{acquire})$

3. A process invokes a synchronization operation with the

 attribute $AR(AR= \text{acquire release})$.

4. A process invokes a synchronization operation with the

 attribute $RA(AR= \text{acquire release})$.

5. A process invokes a synchronization operation with the

 attribute $RA(AR= \text{acquire release})$.

where $v,v_0= synchronization$ operation,

A phase $Q0$ is reachable from $Q$, denoted by $Q - Q0$, if there is a path from $Q$ to $Q0$. It means $Q$ reaches $Q0$. Two phases are concurrent if there is no path between them. Competing accesses are two operations accessing the same shared variable in concurrent phases and one of the operations is write.

Assume two operations $o$ and $o0$ are executed in two phases, $Q$ and $Q0$ respectively. $o0$ is reachable from $o$. If any of the following conditions is satisfied:

1. $Q=Q0$ and $o$ is executed before $o0$,
2. $Q - Q0$.

V. **VISIBILITY OF WRITTEN VALUES ON SHARED VARIABLES**

A written value of a write operation, $w$, on shared variable $X$ in phase $Q$ is visible to a read operation on $x$ in phase $Q0$ if and only if one of following situations holds

1. if $Q=Q0$, the write operation is the last write operation on $x$ before the read operation.

2. if $Q \neq Q0(\text{not equal})$ and $Q0$ is reachable from $Q$, $w$ is the last write on $x$ in phase $Q$ and there is no other phase on the path from $Q$ to $Q0$ which has a write operation on $x$.

3. $Q$ and $Q0$ are concurrent.

The set of written values visible to the read operation $op$ on $X$ is called the visible set of $X$.

From the programmer's point of view, an acquire action takes place when a synchronization object replies to a requesting process. All the updates in shared memory that are visible to the synchronization object before the synchronization object replies are also visible to the process after the process receives the response from the synchronization object. The time instance when $R(R=\text{release})$ acts depends on how conservatively the updates in shared memory are expected to be propagated. If the attribute $RA(AR= \text{release acquire})$ is used, the updates of the requesting process are visible to the synchronization object when the object receives the request. If the operation is annotated with $AR(AR= \text{acquire release})$, the updates are made visible to the synchronization object after the object replies to the process. The attribute $AR(AR= \text{acquire release})$ can be used for atomic updates to the synchronization object. For example, in the consumer producer problem, if the queue buffer in the synchronization object buffer is full, the producer needs to be suspended until some products are taken.
by consumers. The updates in shared memory by the producer
do not have to be seen by others until the products are stored
in the queue buffer. The function PutItemPtr may be annotated
with the attribute AR(AR=acquire release). A
synchronization operation can have no attribute. The updates
of the process are not visible to such a synchronization object
when it executes the synchronization operation

REFERENCES
Vol.2,1992
[2] M. Dubois and C. Scheurich., “Memory access dependencies in shared-
memory multiprocessors” IEEE Transaction on Software Engineering,
June 1990.
-36k
Design(international computer system series )”Second Edition Published
by Addison-Wesley”, 1994.
Concepts and Design” ,third edition, published August 7, 2000 672
pages