Look-Ahead Memory Consistency Model

Chao-Chin Wu, Der-Lin Pean, and Cheng Chen

Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, ROC
Tel: 886-3-5712121 ext: 54734 Fax: 886-3-5724176
Eager E-mail: cchen@eceicpa5.csie.nctu.edu.tw

Abstract

In this paper, we propose a hardware-centric look-ahead memory consistency model that makes the data consistent according to the special ordering requirement of memory accesses for critical sections. The novel model imposes fewer restrictions on event ordering than previously proposed models thus offering the potential of higher performance. The architecture has the following features: (1) blocking and waking up processes by hardware, (2) allowing instructions to be executed out-of-order, (3) until having acquired the lock can the processor allow the requests for accessing the protected data to be evicted to the memory subsystem. The advantages of the look-ahead model include: (1) more program segments are allowed parallel execution, (2) locks can be released earlier, resulting in reduced waiting times for acquiring lock, and (3) less network traffic because more write requests are merged by using two write caches.

1. Introduction

A memory consistency model determines the order in which memory references can be executed by the system, and greatly affects the implementation and performance of a system [1-9]. The most commonly assumed memory model is sequential consistency (SC) [1]. While SC provides a simple model for programmers, it imposes many constrains on architecture and compiler optimizations that exploit the reordering and overlap of memory references. To achieve better system performance, several relaxed memory models have been proposed, for example, processor consistency (PC) [2], weak consistency (WC) [3, 4], and release consistency (RC) [5, 6]. These models are referred to as hardware-centric ones because they are defined in terms of relatively low-level hardware constraints on the ordering of memory references [8]. Currently, the release consistency model is the generally accepted hardware-centric relaxed memory consistency model because of its performance and implementation complexity [9].

In this paper we propose a novel hardware-centric memory consistency model called look-ahead model. This model allows more buffering and pipelining than previously proposed models, including the release consistency model. The new model further classifies each acquire and each release operation as either exclusive or non-exclusive. It is easy to distinguish between them because it is according to whether an acquire or release operation is for implementing a critical section or not. Our model thus can exploit the parallelism between critical section and the code segment following the critical section. We can perform memory accesses in the subsequent code segment even before all memory accesses in the critical section have been performed.

None of the existing processor architectures can utilize the parallelism exploited by our proposed model because those architectures are blocked or spinning when lock failure occurs, so there is no chance to execute the code segment following the critical section. Consequently, we designed a new processor. When being informed of lock failure, the processor prevents only the shared memory requests in the associated critical section from being issued to the memory subsystem. As with out-of-order execution, the processor can continue to execute the instructions following the critical section.

Look-ahead model can be easily implemented using hardware and adds little complexity. The advantages of the look-ahead model are as follows. (1) Waiting time for acquiring lock can be hidden by executing independent instructions in the code segment following the critical section. (2) Locks can be released earlier, resulting in reduced waiting times. (3) More network traffic for write requests are reduced.
The organization of the rest of this paper is as follows. In Section 2, we review the release consistency model. In Section 3, we define our look-ahead model. In Section 4, we describe the processor architecture support for the new model. In Section 5, we analyze the performance potential of the look-ahead model. In Section 6, we conclude the paper and state the future work directions.

2. Release consistency model [5, 6]

Release consistency (RC) is the generally accepted hardware-centric relaxed memory consistency model because of its performance and implementation complexity. It categorizes memory references, as depicted in Figure 1, to relax restrictions on memory access ordering. Memory references are classified as shared or local references in multiprocessor systems. Reference to memory block shared by at least two processors is called a shared memory reference. Reference to memory block belonging to a single processor is called a local memory reference. We further divide shared memory references into special and ordinary references according to whether or not they compete. Special references are further distinguished as synchronization and non-synchronization ones. References used to enforce orderings among processes are called synchronization references. On the other hand, non-synchronization references correspond to asynchronous data operations or special operations that are not used for synchronization. Synchronization references can be further classified into acquire and release references. An acquire synchronization reference grants access to a set of shared locations and a release synchronization reference grants permission for this reference. 

![Figure 1. Categorization of shared memory accesses](image)

The following gives the conditions for ensuring release consistency:

(A) before an ordinary LOAD or STORE reference may be performed with respect to any other processor, all previous acquire references must have been performed, and
(B) before a release reference may be performed with respect to any other processor, all previous acquire references must have been performed, and
(C) special references are processor consistent with respect to one another.

In the following section, we propose a new memory consistency model by extending release consistency model.

3. Look-ahead consistency model

A critical section is bounded by a pair of acquire and release references to a synchronization variable [10]. An acquire reference occurs at the beginning of a critical section, and is used to gain access to a set of shared memory locations. A release reference occurs at the end of the critical section, and is used to signal that access is available. An acquire reference and a release reference together protect the shared data within the critical section from concurrent accessing. When a process executes an acquire reference to enter a critical section, it will later execute a release reference. According to the semantic of critical section, before we release a lock, data that can be accessed within the corresponding critical section must be made consistent with all processors. By the above observation, we propose a new consistency model called look-ahead consistency model.

We classify acquire and release accesses into two parts. The acquire and releases for implementing critical sections are called exclusive acquire and releases. Other acquire and releases are called non-exclusive acquire and releases. Moreover, exclusive acquire and exclusive releases are called exclusive synchronizations, and non-exclusive acquire and non-exclusive releases are called non-exclusive synchronizations. For convenience, we also define a segment as a code segment between two adjacent synchronization points. Segments bounded by a paired exclusive acquire and exclusive release are called exclusive segments. All others are called non-exclusive segments. Figure 2 shows an example to illustrate the concept of segments.

![Figure 2. An illustration of the concept of segments](image)
ess. (2) A non-exclusive release is to inform other processes that accesses appearing before it in the program order have completed. (3) An exclusive acquire access is to delay future accesses in the associated exclusive segment until being informed by another process. (4) An exclusive release access is to inform other processes that all the data accesses in the associated exclusive segment have been completed. According to their different semantics, we have the following formal conditions for the look-ahead model.

(A) before an ordinary LOAD or STORE access in the exclusive segment may be performed with respect to any other processor, the associated exclusive acquire must have been performed; and

(B) before an ordinary LOAD or STORE access in a non-exclusive segment may be performed with respect to any other processor, all previous non-exclusive acquire must have been performed; and

(C) before an exclusive-release may be performed with respect to any other processor, all previous ordinary LOAD and STORE accesses in the associated exclusive segment must have been performed; and

(D) before a non-exclusive release may be performed with respect to any other processor, all preceding ordinary LOAD and STORE accesses not in the exclusive segment must have been performed; and

(E) special accesses, not including exclusive acquire and exclusive releases, are processor consistent with respect to one another;

(F) exclusive acquire, and exclusive releases are processor consistent with respect to one another.

Our new model relaxes three ordering restrictions imposed by release consistency. (1) The ordinary LOAD and STORE accesses in non-exclusive segments do not have to be delayed for the previous exclusive acquire accesses to complete. An exclusive acquire access only prevents the shared data in the associated exclusive segment from being accessed by other processors, and has nothing to say about ordering of accesses in subsequent non-exclusive segments. Of course, local dependencies within the same processor must still be respected. (2) A non-exclusive acquire access need not be delayed for the preceding exclusive acquire access to be performed. Since a non-exclusive acquire is not giving the permission to any other process to read/write the pending locations in the previous exclusive segment, there is no reason for a non-exclusive acquire to wait for the previous exclusive acquire to complete. (3) An exclusive release does not wait for the ordinary accesses in the previous non-exclusive segments. Because the exclusive release access only gives the permission to other process to read/write the shared data in the associated exclusive segment, it has no need to be delayed for the accesses in the preceding non-exclusive segments to be performed.

4. Architecture support

Realizing the full potential of a model will generally depend on the access behavior of the application program and may require novel architecture and compiler techniques. In this section, we present the architecture supports for achieving the highest potential performance of the look-ahead model.

4.1. Basic requirements

To use the look-ahead model, the most important element is to distinguish between exclusive accesses and non-exclusive accesses. This requires the following three basic requirements.

(1) Programs must be properly labeled [5] by programmers or compilers to ensure correct execution results. The labeling procedure is the same as that for the release consistency model except that acquire and releases for implementing critical sections must be labeled as exclusive acquire and exclusive releases; others are labeled as non-exclusive acquire and non-exclusive releases. To distinguish exclusive and non-exclusive accesses, we have to add two new macros for exclusive acquire and exclusive release accesses. In addition, because the ordering restrictions on exclusive accesses are relaxed in look-ahead model, if we cannot make sure that an access is exclusive or non-exclusive, then we label this access as a non-exclusive access. Thus, the labeling procedure for the look-ahead model is very simple.

(2) Hardware must have the capability to distinguish between exclusive synchronizations and non-exclusive synchronizations. Two approaches are available. We can either allocate the lock variables in a special space such that when the synchronization accesses are executed, we determine whether they are exclusive or non-exclusive by their addresses; or we can implement four separate instructions for exclusive acquire, exclusive release, non-exclusive acquire, and non-exclusive release.

(3) Two write caches [11] must be used to buffer memory access requests from exclusive segments and non-exclusive segments. Write caches are used to be an allocate-on-write-miss, write-back, and no-allocate-on-a-read-miss strategies with a single, combined dirty/valid bit per word. A write cache allocates a block frame to a write reference. The write reference is performed in the write cache by setting the corresponding dirty bit and writing the data. Therefore, all the write references belonging to the same block will be merged into a single write miss request. If the block is replaced or a write cache is flushed, only dirty words need to be
transferred to the next level in the memory hierarchy. Consequently, temporal and spatial locality in write references will result in less write traffic.

One of these two write caches is for buffering memory references from exclusive segments, and the other is for other references. For the look-ahead model, the write cache for exclusive segments is flushed to the write buffer when an exclusive release is issued. Similarly, the other write cache is flushed to the write buffer when a non-exclusive release is issued. Flushing the write cache means that all modified write-cache blocks are removed and the modifications are transferred to the write buffer. Therefore, during the lifetime of a write-cache block, temporal and spatial locality in write accesses to the block can be exploited resulting in less write traffic.

4.2. Requirements for processor architecture

As described in Section 3, we find that for the look-ahead model we can execute subsequent non-exclusive segments even when an exclusive acquire access has not been performed. However, unfortunately none of the existing processor architectures can overlap execution of exclusive segments with that of non-exclusive segments because the process will be blocked or spin if it can not acquire a lock to enter an exclusive segment. Therefore, the process has no chance to execute the codes following the exclusive segment. To achieve the highest performance potential, we need a new processor architecture that effectively supports the look-ahead model. The processor has to have ability to execute subsequent non-exclusive segments before it acquires a lock to enter a exclusive segment. Before having acquired the lock to enter the exclusive segment, it can execute its subsequent non-exclusive segment. To have such an ability, the processor architecture must meet the following requirements.

(1) Hardware blocking upon lock failure. Lock failure can be handled in two ways: blocking or spinning. Spinning is a software method whereby the process repeats reading the lock value using a while loop until the lock succeeds. However, this method gives the processor no chance to execute other codes. On the other hand, blocking can be a hardware or software method. With blocking, when a lock failure occurs, the process suspends execution and relinquishes control of the processor and it will be waked up after the lock is released by another process. For the blocked process, the processor is idle until it receives a signal allowing it to enter the critical section. However, the processor can do other work during this period. The procedures for blocking and waking up processes can be implemented with software or hardware. If we adopt software blocking method, two groups of instructions for blocking and waking up processes must be encoded in the program, which will consume processors' execution time. Alternatively, if hardware blocking is used, processors can continue to execute other useful work while cache controllers handle the procedures for blocking or waking up processes. To use the look-ahead model to execute subsequent non-exclusive segments of the same process during lock failure, hardware blocking must be employed.

(2) Data dependencies between exclusive segments and non-exclusive segments must be conserved. Though we allow the processor to execute non-exclusive segments before executing the preceding exclusive segment, the data dependencies of both memory locations and registers must be conserved between the two kinds of segment. To meet this requirement, we must calculate all the data addresses accessed in the exclusive segment before executing non-exclusive segments. Therefore, we must first decode all the instructions according to the original program order, then record the decoded instructions in a instruction buffer. We place a special mark on each instruction belonging to exclusive segments, including exclusive release instructions. Next, we can issue the decoded instructions without any dependency from the instruction buffer to available functional units to execute. However, each of the memory accesses cannot be sent to the memory subsystem until the addresses of all previous memory accesses have been calculated because we must conserve the data dependency of memory locations. In addition, we cannot send the requests to shared memory issued by the marked instructions to the memory subsystem until the processor receives permission to enter the exclusive segments because only the accesses to marked shared data in the exclusive segment are protected by the exclusive acquire and release instructions. These marked accesses must not be made visible to any other processor until it has received permission to access the exclusive segments. On the other hand, access to unmarked shared data outside the exclusive segments can be mad visible to other processors, provided data dependencies are conserved.

The concept of executing subsequent non-exclusive segments before completing the pending exclusive segments is similar to that of out-of-order execution. Therefore, in the next section, we will design a processor architecture for the look-ahead model by extending the out-of-order-execution superscalar architecture proposed by Mike Johnson [12].

4.3. Processor architecture implementation

As described in Section 4.2, we know that the processor for the look-ahead model must have three features. (1) Out-of-order execution: the processor can execute subsequent non-exclusive segments even when the preceding exclusive segment has not been completed. (2) Hardware blocking upon lock failure: this allows the processor to execute other instructions even when lock failure occurs.
(3) Data dependency between exclusive segments and non-exclusive segments must be conserved. Below, we describe how to design a processor that has these features.

(1) The processor is an extension of Johnson’s superscalar architecture [12]. It can issue and execute instructions out-of-order. The central window issues instructions without data dependency to available functional unit. The reorder buffer performs register renaming and supports precise interrupts.

(2) Four different atomic instructions are implemented for exclusive acquire, exclusive release, non-exclusive acquire, and non-exclusive release, respectively. They can be identified after decoding instructions. The acquire instructions issue memory access requests to the memory subsystem. If lock variables have been set, the cache controller enqueues the processes on a blocked list and informs the processor of the lock failure. On the other hand, the release instructions let the cache controller wake up one of the processes in the blocked list, if the block list is not empty.

(3) If the central window has many paired exclusive acquires and exclusive releases, we must identify them. To simplify the architecture complexity, at any time the central window has at most only one exclusive acquire and one exclusive release in it. We use flag \( O \) to indicate when an exclusive acquire is in the central window. If the processor finds that the flag is set when decoding an exclusive acquire, it stop sending instructions to the central window. After decoding an exclusive acquire, the processor sets flag \( O \). Flag \( O \) is reset when an exclusive release is retired from reorder buffer. Therefore, we must add an extra bit \( R \) to each reorder buffer entry to indicate that the instruction is an exclusive release.

(4) We use signal \( E \) to indicate whether instructions are in an exclusive segment. Each central window entry has an extra field \( E \) to record the corresponding \( E \) value. When an exclusive acquire is decoded, the signal \( E \) is set, and after an exclusive release is decoded, the signal is reset. Consequently, the value of field \( E \) is one if an instruction is in an exclusive segment; otherwise, field \( E \) is zero.

(5) After decoding a non-exclusive acquire, the processor stops decoding and sending instructions to the central window until it is informed that the lock acquisition has been granted because all the instructions following the non-exclusive acquire can only be executed after the lock acquisition has succeeded.

(6) When an acquire finds that the lock variable has been set, the cache controller enqueues the request in the blocked list and informs the processor which lock acquisition has failed. If the lock failure is associated with non-exclusive acquire, then memory accesses requested by instructions with \( E \) set cannot be sent to cache memory until the processor resets the \( E \) value upon being informed of lock success. Otherwise, the processor continues decoding and sending instructions to the central window.

Figure 3 shows partial block diagram of our new processor architecture. In summary, the processor has the following features. (1) Decoder functioning is enhanced, it can detect exclusive acquires and releases and update the values of \( E \), \( O \), and \( R \). (2) We add to each central window entry a one-bit field \( E \) that indicates whether or not the corresponding instruction is in an exclusive segment. We can execute only those instructions whose \( E \) values in the central window are zero. The \( E \) values in the central window are reset when processor is informed of lock success. (3) Each reorder buffer has an extra one-bit field \( R \) that indicates whether or not the corresponding instruction is an exclusive release. The \( O \) and \( R \) values are maintained to ensure that at most only one exclusive segment is in the central window at any time. Consequently, the processor is not much more complex, and we believe it is cost-effective because the processor can execute subsequent non-exclusive segments even when a lock failure occurs.

![Figure 3](image-url)

**Figure 3.** Partial block diagram of our new processor architecture

5. Performance analysis

![Figure 4](image-url)

**Figure 4.** Possible overlap difference between the release consistency and the look-ahead model
In this section we explore the potential gains in performance for the look-ahead consistency model. Figure 4 shows an example that highlights the difference between release and look-ahead consistency models, assuming that there is no local dependence. Compared with the release consistency model, the look-ahead model redraws one arrow and eliminates two arrows. The look-ahead model redraws the arrow initiated from the non-exclusive segment labeled 1. The new destination is the non-exclusive release instead of the exclusive release. The two eliminated arrows are from the exclusive acquire to the non-exclusive acquire and to the non-exclusive segment labeled 4. Consequently, segments 2, 3, and 4 can be executed in parallel in look-ahead model. This is different from that in release model: segments 3 and 4 cannot be executed until the exclusive acquire in segment 2 is completed.

Figure 5 compares the execution times of these two different models. In the release consistency model, none of the accesses can be performed before their previous acquire access has been performed. Therefore, the non-exclusive segment has to wait for the completion of its previous exclusive acquire access, as shown in Figure 5(a). However, in the look-ahead model, only non-exclusive acquire access will delay all of its subsequent access to be executed. The exclusive acquire access only delays all ordinary accesses in the associated exclusive segment and its associated exclusive release access. By taking advantage of this feature, the processor will not be blocked when a lock failure occurs, instead, the processor can execute the subsequent non-exclusive segments. Therefore, the waiting time for acquiring the lock by the exclusive acquire access can be hidden by the execution of the subsequent non-exclusive segment, as shown in Figure 5(b). Because the performance gain comes from overlapping exclusive segments and non-exclusive segments, the difference in performance between two models arises when there are more exclusive segments.

![Figure 5](image.png)

**Figure 5.** Execution overlap between exclusive segments and subsequent non-exclusive segments

Another performance advantage of look-ahead consistency model is shorter waiting time for acquiring lock. We explain the reason as follows. Release consistency requires that the release access cannot be performed until all its previous accesses have been performed. On the other hand, the look-ahead model relaxes the restriction by allowing exclusive release accesses to be performed when its associated exclusive acquire accesses and the ordinary accesses in the associated exclusive segment have all been performed. That is, an exclusive release access has to wait for fewer memory accesses to unlock its lock. Consequently, the next waiting process will obtain the lock ownership and enter the corresponding critical section earlier. Because the way that processes enter a critical section is one by one, when the waiting time for entering a critical section can be reduced, the performance can be enhanced. Furthermore, the more the processes contend for a lock at the same time, the more the performance gain can be obtained.

Finally, by using two write caches, the above advantage can be fully exploited. This is because an exclusive release reference only flushes the write cache for exclusive segments. Consequently, the exclusive release reference has to wait for only those write requests really in its associated exclusive segment. The write requests before the exclusive segment are still in the write cache for non-exclusive segments, and they will not delay the exclusive release reference.

The second advantage of two write caches is that the write accesses in different non-exclusive segments can be merged resulting in less write traffic. Figure 6 shows an example that illustrates this advantage. Merging write accesses can delay invalidations and reduce the number of invalidations for write invalidation protocols. It can delay invalidations for write accesses in non-exclusive segments until a non-exclusive release is encountered, even though several exclusive releases have been encountered. Delaying invalidations and reducing the number of invalidations can further cut coherence misses and false sharing misses.

![Figure 6](image.png)

**Figure 6.** Merging of write accesses by using two write caches

### 6. Concluding remarks

In multiprocessor systems, it is very important to provide an efficient memory consistency model. In this paper, we have proposed a hardware-centric memory consistency model, called look-ahead consistency model. The new
model extends the release consistency model by partitioning acquire and release accesses into exclusive and non-exclusive ones. It allows the overlap executions of exclusive and non-exclusive segments. By using two write cache, network traffic for write requests are reduced. Moreover, the waiting time for acquiring lock is shortened. These advantages make the look-ahead consistency model outperforms all other previously proposed consistency models.

The novel model requires several architecture and software supports to realizing its full performance potential. (1) Four instructions are implemented for exclusive acquire, exclusive release, non-exclusive acquire, and non-exclusive release, respectively. (2) Programs must be properly labeled. (3) Two write caches are used to buffer write accesses belonging to exclusive segments and non-exclusive segments, respectively. We have also designed a new processor architecture to support the novel model. The processor is extended from Johnson's superscalar architecture, and thus can execute instructions out-of-order. In addition, procedures for blocking processes and waking up the blocked processes are implemented by means of hardware. Because it can perform these two functions, even when lock failure occurs, it can continue executing instructions without dependencies in subsequent non-exclusive segments. Consequently, the waiting time for acquiring locks can be hidden by executing the subsequent codes.

We are currently building a simulator for the look-ahead system to quantify the performance difference between the release consistency and the look-ahead model on real applications. The comparisons include: (1) write traffic, (2) waiting time for acquiring locks, and (3) total execution cycle.

**Acknowledgement**

The authors would like to thank the reviewers for their helpful comments. This research was supported by the National Science Council of the Republic of China under contract numbers: NSC 87-2213-E009-048 and NSC 87-2213-E009-049.

**References**


