

METHOD, SYSTEM AND APPARATUS FOR TIMESTAMP-BASED CACHE COHERENCE

Field of Invention

5 The invention generally relates to cache coherence protocol and more particularly, method, system and apparatus for efficient timestamp-based cache coherence.

Background

 Among all existing coherence protocols, timestamp-based coherence
10 protocol shows potential in terms of cache performance, especially when the coherence protocol is applied to a large-scale system.

 One existing timestamp-based coherence protocol is Library Cache Coherence (LCC) protocol. LCC protocol maintains coherence by stalling on every write until timestamps of all the shared copies have expired and all the
15 shared copies have been self-invalidated. This will increase network traffic and memory access latency, thus significantly affecting the cache performance.

 Another existing timestamp-based coherence protocol is Temporal Coherence (TC) protocol which improves LCC protocol for GPU coherence by only stalling on memory fences, and shows significant speedup. However, the
20 problem of TC protocol is that the release operations need to be stalled when memory fence operations encountered until all the global timestamps returned

by writes which are maintained in a Global Write Completion Times (GWCT) table expire.

It is therefore desirable to provide a more efficient timestamp-based cache coherence strategy to further improve the cache performance.

5 **Summary of Invention**

Embodiments of the invention provide a method for timestamp-based cache coherence in a system comprising a plurality of cores, a plurality of private caches, each of the private caches for storing/recording unexpired write operations initiated by a local core in a Local Write Set (LWS) and by at least one remote core of the system in a Remote Write Set (RWS), and at least one shared cache each configured for storing/recording unexpired write operations associated with a predetermined group of memory addresses released by the private caches in the system in a Global Write Set (GWS), the method comprising:

15 in a write operation, writing, by a first private cache, a data stored in a memory address to a first corresponding shared cache; returning, by the first corresponding shared cache, a timestamp of the data to the first private cache; determining, by the first private cache, whether the timestamp is expired based on a current system time; and adding, by the first private cache, the address
20 associated with the data into a Local Write Set (LWS) of the first private cache; and

in a release operation, merging, by a second private cache, the RWS and the LWS in the second private cache and releasing the merged RWS and LWS to a second corresponding shared cache; merging, by the second corresponding shared cache, the RWS and LWS received from the second private cache with
5 the GWS of the second corresponding shared cache.

In other embodiments of the invention, the method further comprises:

in an acquire operation, sending, by a third corresponding shared cache, a requested data and the GWS in the third corresponding shared cache to a third private cache which requests the data; acquiring, by the third private cache the
10 requested data and merging, by the third private cache, the received GWS with RWS of the third private cache;

in a read operation, checking, by a fourth private cache, whether a memory address of a data is recorded in RWS of the fourth private cache, self-invalidating the data and loading the data from a fourth corresponding shared
15 cache if the address of the data is recorded in the RWS of the fourth private cache.

In one embodiment of the invention, the system comprises only one shared cache.

In one embodiment of the invention, each of the LWS, RWS and GWS
20 has a predefined signature data structure which includes a time stamp and a filter field, wherein the filter field is a bit vector having a plurality of bits, wherein a predetermined group of bits in the bit vector indicate whether any one of a

predetermined group of addresses has been written by a core in the system and is not globally visible in the system, wherein the predetermined group of bits include at least one bit.

Preferably, any one of the LWS, RWS and GWS is a Bloom filter.

5 In order to reduce size of the LWS in each private cache, the method further comprises:

maintaining, by each private cache, a write FIFO queue to record a local write miss which is returned with a timestamp by one of the at least one shared cache, and in a write operation, the method further comprising: queuing a local
10 write miss at end of the write FIFO in a fifth private cache, and when the local write miss reaches head of the write FIFO, adding, by the fifth private cache, the address associated with the local write miss into LWS of the fifth private cache if the local write miss has an unexpired timestamp.

To further improve the cache performance, the method further comprises:

15 each cache line in a private cache includes a state field for indicating a current status of data in the cache line, wherein the current status is one of four stable states: Invalid, Shared, Exclusive, and Modified;

each cache line in a shared cache includes a state field for indicating a current status of data in the cache line, wherein the current status is one of
20 three stable states: Invalid, Shared, Exclusive.

To reduce redundant self-invalidations due to expiration of a timestamp, the method further comprises: adding a Timestamp Bypass (TB) bit into each cache line in each private cache.

According to a second aspect of the invention, a system for timestamp-based cache coherence is provided. The system comprises:

a plurality of cores,

a plurality of private caches each configured to store/record unexpired write operations initiated by a local core in a Local Write Set (LWS) and by at least one remote core in the system in a Remote Write Set (RWS), and

at least one shared cache, each shared cache configured to store/record unexpired write operations associated with a predetermined group of memory addresses released by the private caches in a Global Write Set (GWS),

wherein each private cache is further configured to, in a write operation, write a data stored in a memory address to a first corresponding shared cache; determine whether a timestamp of the data returned by the first corresponding shared cache expires based on a current system time, and add the address in which the data is stored into LWS of the private cache; the first corresponding shared cache is further configured to return a timestamp of a data to the private cache;

each private cache is further configured to, in a release operation, merge the RWS and LWS in the private cache and release the merged RWS and LWS to a second corresponding shared cache; the second corresponding shared

cache is further configured to merge the RWS and LWS received from the private cache with GWS of the second corresponding shared cache.

According to a third aspect of the invention, an apparatus for timestamp-based cache coherence is provided. The apparatus comprises:

- 5 a Local Write Set (LWS) configured to store/record unexpired write operations initiated by a local core which is comprised in a system which further comprises at least one remote core, and at least one shared cache each configured to store/record unexpired write operations associated with a predetermined group of memory addresses in a Global Write Set (GWS), and
- 10 a Remote Write Set (RWS) configured to store/record unexpired write operations initiated by at least one remote core in the system;

wherein the apparatus is configured to, in a write operation, write a data stored in a memory address to a first corresponding shared cache in the system; determine whether a timestamp of the data returned by the first corresponding
15 shared cache expires based on a current system time, and add the address in which the data is stored into the LWS;

the apparatus is further configured to, in a release operation, merge the RWS and LWS and release the merged RWS and LWS to a second corresponding shared cache in the system.

- 20 According to a fourth aspect of the invention, an apparatus for timestamp-based cache coherence is provided. The apparatus comprises:

a Global Write Set (GWS) configured to store/record unexpired write operations associated with a predetermined group of memory addresses released by a plurality of private caches which are comprised in a system which further comprises a plurality of cores, wherein each private cache is configured
5 to store/record unexpired write operations initiated by a local core in a Local Write Set (LWS) and by at least one remote core of the system in a Remote Write Set (RWS),

wherein in a write operation, the apparatus is configured to return a timestamp of a data to a requesting private cache;

10 in a release operation, the apparatus is further configured to merge RWS and LWS received from a private cache with the GWS.

Brief Description of the Drawings

The invention will be described in detail with reference to the accompanying drawings, in which:

15 Figure 1(a) is a block diagram illustrating a system for timestamp-based coherence protocol according to one embodiment of the invention;

Figure 1(b) shows an example of a predefined signature;

Figure 2 is a flow chart illustrating a method for timestamp-based cache coherence according to the system shown in Figure 1;

20 Figure 3 shows cache line format of a private cache and shared cache according to one embodiment of the invention;

Figure 4 shows an example of the process of self-invalidating the cache lines according to one embodiment of the invention;

Figure 5 is a block diagram illustrating structure of a private cache and a shared cache according to another embodiment of the invention.

5 **Detailed Description of Embodiments of the Invention**

In the following description, numerous specific details are set forth in order to provide a thorough understanding of various illustrative embodiments of the invention. It will be understood, however, to one skilled in the art, that embodiments of the invention may be practiced without some or all of these
10 specific details. It is understood that the terminology used herein is for the purpose of describing particular embodiments only, and is not intended to limit the scope of the invention. In the drawings, like reference numerals refer to same or similar functionalities or features throughout the several views.

Embodiments of the invention provide an efficient solution for timestamp-
15 based cache coherence protocol. This solution can effectively reduce release stall during a write release operation by using a signature scheme to improve cache performance.

A system for timestamp-based coherence protocol provided in embodiments of the invention includes a plurality of cores/GPUS/processors, a
20 plurality of private caches each configured to store or record un-expired write operations initiated by a corresponding local core and other remote cores in the system by predefined signatures of the un-expired write operations, and at least

one shared cache each configured to store/record un-expired write operations associated with a predetermined group of memory addresses released by the private caches in the system.

Figure 1(a) is a block diagram illustrating a system for timestamp-based coherence protocol according to one embodiment of the invention. The system 100 includes two cores, e.g. core 0 and core 1, two private caches, e.g. L1 cache-0 110a and L1 cache-1 110b, and a shared cache, e.g. L2 cache 120.

The L1 cache-0 110a includes a Local Write Set (LWS) 111a configured to record un-expired write operations initiated by core 0; a Remote Write Set (RWS) 112a configured to store or record un-expired write operations initiated by other remote cores in the system, i.e. core 1 in this system, .

The L1 cache-1 110b includes a Local Write Set (LWS) 111b configured to record un-expired write operations initiated by core 1; a Remote Write Set (RWS) 112b configured to store or record un-expired write operations initiated by other remote cores in the system, i.e. core 0 in this system.

The shared cache L2 cache 120 includes a Global Write Set (GWS) 121 configured to store/record un-expired write operations released by all of the private caches in the system, i.e. L1-cache-0 110a and L1-cache-1 110b.

In this embodiment, each un-expired write operation is stored/recorded using a predefined signature. The predefined signature includes a timestamp and a filter field/ bit vector. A predetermined group of bits in the bit vector indicate whether a predetermined group of memory addresses is written by a

core in the system and whether the timestamp of the corresponding write operation remains unexpired, i.e. not globally visible. The predetermined group of bits includes at least one bit. Figure 1(b) shows an example of the predefined signature. In this example, the bit vector includes n bits $b_1, b_2, \dots, b_n$, wherein
5 each bit in the bit vector indicates whether a predetermined group of memory addresses is written by a core in the system and whether the timestamp of the corresponding write operation remains unexpired. Specifically, b_1 corresponds to 1st group of addresses, b_2 corresponds to 2nd group of addresses and b_n corresponds to the n th group of addresses. If any one of the predetermined
10 group of addresses is written by a core in the system and the timestamp of the corresponding write operation is still not expired, the corresponding bit of the bit vector will be set as true, e.g. 1.

There are four major operations on the predefined signature: set, lookup, merge and clear. Set operation refers to mapping a memory address to a
15 predetermined group of bits of the filter field and setting the predetermined group of bits as true; lookup operation refers to mapping a memory address to a predetermined group of bits of the filter field and returning the value of the bit; merge operation refers to an “OR” logical operation, i.e. “OR” the corresponding two bits for the two filter fields and the timestamp will be the maximal timestamp
20 of the two merged signatures; clear operation refers to setting all the bits of the filter field in the signature as False. The predetermined group of bits include at least one bit.

Figure 2 is a flow chart illustrating a method for timestamp-based cache coherence according to the system shown in Figure 1. Referring to Figure 1, in this embodiment, the processing system includes two cores, e.g. core 0 and core 1, two private caches, e.g. L1 cache-0 and L1 cache-1, and one shared cache, e.g. L2 cache. Core 0 and core 1 will respectively operate the following instructions sequentially:

core 0: st A; st_rel T; st D

core 1: ld_acquire T; ld A

In these instructions, A and D refer to data, "st" refers to a write operation; "rel" refers to a release operation; "ld" refers to a read/load operation.

In Block 2001, in a normal write operation, i.e. during execution of the instruction st A, the private cache, e.g. L1 cache-0 writes data A including a corresponding address to the shared cache.

In Block 2002, the shared cache, e.g. L2 cache returns a Timestamp of the data A to the L1 cache-0.

In Block 2003, the L1 cache-0 determines whether the Timestamp of the data A is expired based on the current system time: if the Timestamp is not expired, the flow chart proceeds to block 2004; if the Timestamp is expired, the flow chart proceeds to block 2005.

In Block 2004, the L1 cache-0 adds the address of Data A into LWS.

In one example of the embodiment, the LWS of the private cache includes a timestamp and a bit vector having 4 bits, each bit corresponding to a group of addresses, and address of data A corresponds to the third bit in the bit vector of the LWS in L1 cache-0. This step is a set operation; accordingly, the third bit of
5 the bit vector is set as true.

In Block 2005, in a release operation, i.e. during execution of the instruction `st_rel T`, the L1 cache-0 merges RWS and LWS to generate a signature to be released to the shared cache, e.g. L2 cache, and release T and the generated signature to the L2 cache. The flow chart proceeds to block 2006
10 to perform the write operation in the L2 cache, and proceeds to block 2007 to perform the next write operation (st D).

The merge operation on RWS and LWS refers to performing an “OR” logical operation on RWS and LWS. In one example of the embodiment, RWS of the L1 cache-0 includes a timestamp and a bit vector having 4 bits and all of the
15 4 bits are false, i.e. there is no unexpired write operation initiated by other remote core in the system. After the merge operation, the generated signature includes a timestamp and a bit vector having 4 bits. The timestamp is the longer one of the RWS and the LWS, and the bit vector is same as the bit vector in the LWS since all bits in the bit vector of RWS are false.

20 In Block 2006, the L2 cache writes T and merges the signature received from the L1 cache-0 with its own GWS.

In one example of the embodiment, the GWS of the L2 cache includes a timestamp and a bit vector having 4 bits which are false. Therefore, after the GWS is merged with the received signature from the RWS of the L1 cache-0, the result is same as the received signature.

5 In Block 2007, in this normal write operation, i.e. during execution of the instruction st D, the private cache L1 cache-1 writes data D including a corresponding address to the shared cache L2 cache.

In Block 2008, the L2 cache returns a Timestamp of the data D to the L1 cache-1.

10 In Block 2009, the L1 cache-1 determines whether the Timestamp of the data D is expired based on the current system time: if the Timestamp is not expired, the flow chart proceeds to block 2010; if the Timestamp has been expired, no further operation is to be performed, the flow chart proceeds to next instruction to be executed by the L1 cache-1 if any.

15 In Block 2010, the L1 cache-1 adds the address of data D into LWS. Then, the flow chart proceeds to next instruction to be executed by the L1-cache-1 if any.

20 In one example of the embodiment, the LWS of the private cache includes a timestamp and a bit vector having 4 bits, each bit corresponding to a group of addresses, and address of data D corresponds to the first bit in the bit vector of the LWS in L1 cache-0. This step is a set operation; accordingly, the first bit of the bit vector is set as true.

In Block 2011, in a read-acquire operation, i.e. during execution of the instruction `ld_acq T`, the L2 cache sends data T and GWS to the L1 cache-1.

In Block 2012, the L1 cache-1 acquires data T and merges the received GWS to its own RWS.

5 In Block 2013, in a read operation, i.e. during execution of the instruction `ld A`, the L1 cache-1 checks whether the address of data A is recorded in its own RWS, if yes, the flowchart proceeds to block 2014, if no, the flow chart proceeds to block 2015.

10 In one example of the embodiment, the L1 cache-1 checks whether the bit value of the bit vector of the RWS corresponding to the address of data A is set true, e.g. whether the bit value is 1.

In Block 2014, the L1 cache-1 self invalidates data A and loads data A from L2 cache. Then, the flow chart proceeds to next instruction to be executed by the L1 cache-1 if any.

15 In Block 2015, the L1 cache-1 loads A from L1cache-1 if timestamp of data A is not expired, or loads data A from L2 cache if timestamp of data A is expired. Then, the flow chart proceeds to next instruction to be executed by the L1 cache-1 if any.

20 It should be noted that although in this embodiment, both the write operation and the release operation are executed on L1 cache-0, in other embodiment, these two operations may be executed on different private cache in the system. Similarly, in this embodiment, both the acquire operation and the

read operation are executed on the L1 cache-1, in other embodiment, these two operations may be executed on different private cache in the system. In addition, although in the embodiment shown in Figure 1 and Figure 2, the system only includes one shared cache, in other embodiment, the system may include a plurality of shared cache, and each shared cache is configured to store or record unexpired write operations associated with a predetermined group of memory addresses released by the private caches in the system.

As explained above, with the signature scheme, the write visibility constraints are relaxed from the time of release to any time when another core in the system actually acquires the synchronization variable that has been released. In this scheme, a set of unexpired write operations are maintained in each private cache and shared cache in the system such that these unexpired write operations are not forced to be coherence at a release operation. Instead, when other cores in the system try to communicate with the core with a release operation, they need to check if the read address belongs to the set of write addresses of the core with the release operation and in that case self-invalidate their stale copies. Here, the stale copies include copies of data in a private cache which have been updated by other remote core.

In one example of the embodiment, at least one of the LWS, RWS in the private cache and GWS in the shared cache is a Bloom filter. A Bloom filter can be used to generate a signature at releases that tracks the local write operations that returned an unexpired global timestamp. A Bloom filter is a space-efficient structure to test if a target data is within a set of predetermined data, where false

positives are possible but false negatives are not permitted. During an acquire operation, the L1 cache obtains the corresponding signature and for subsequent read operations in Shared state, the requested line is self-invalidated if the address hits in the signature even if the line's timestamp has not expired. By
5 keeping track of uncompleted writes before release operations and selectively self-invalidating stale lines, the heavy burden of release-stalling is effectively removed.

In the timestamp-based coherence protocol provided in embodiments of the invention, one advantage of using Bloom filter for write-tracking is that the
10 signature naturally inherits a timestamp from the coherence protocol, indicating the global completion time of the tracked write operations. When the timestamp of the signature expires, the filter field (a bit-vector) can be cleared because all the write operations tracked in the signature have become globally visible.

In order to further improve the cache performance, in one embodiment of
15 the invention, a data classification scheme is applied. The data classification scheme introduces a state field to each cache line in L1 cache and L2 cache to indicate current status of each cache line. For L1 cache, there are four stable states: Invalid, Shared, Exclusive and Modified, while for L2 cache, there are three stable states: Invalid, Shared and Exclusive. The cache line of the shared
20 cache, i.e. L2 cache, may further include an owner field to indicate an exclusive owner. The Exclusive state in L2 cache corresponds to both Exclusive state and Modified state in L1 cache. For L1 Exclusive/Modified lines, i.e. L1 cache line in Exclusive/Modified state, a pointer is maintained in the L2 cache line to keep

track of the exclusive ownership. However, the sharing vectors for L1 cache lines in shared state are not stored in L2 cache.

A normal write operation hits on L1 cache lines in Exclusive/Modified state, also referred to as L1 Exclusive/Modified lines. A write miss in the L1
5 cache for other states and an exclusive request (GetX) is sent to L2 cache. For a write miss, along with data, a timestamp may be returned from the L2 cache that captures the time when the write operation will become globally visible. If the L2 cache line receiving the GetX request is in Shared state, it immediately responds with data and the global timestamp stored in the cache line, which is different
10 from directory protocol where the other L1 copies have to be invalidated immediately. If the L2 cache line is in Exclusive state, the request is forwarded to the tracked owner who invalidates its line and sends the data to the requester. Note that it is possible for a L2 Exclusive line to have an unexpired global timestamp as there can still be stale copies lingering around in L1 caches other
15 than the owner. In that case, the timestamp in the L2 cache line is also transferred in the forwarded request, which is re-forwarded to the original requester by the owner. For an access to an L2 Invalid line, data is loaded from main memory and sent to L1Cache. Upon receiving the response from the L2 cache, the L1 cache writes the data to its line and transitions to Modified state.
20 To track the global timestamps returned by write operations, the GWCT needs to be updated if the response contains a larger timestamp. The L1 cache completes the transaction by sending an acknowledgment to the L2 cache,

which transitions the cache line to Exclusive state and changes the ownership of the line to the requester.

During a normal read operation, a normal read hits on L1 cache lines in Exclusive/Modified state. A read to L1 Shared lines need to check the stored
5 local timestamp: a tag match with an expired timestamp is treated as a read miss and a read request is sent to the L2 cache. Note that self-invalidating an L1 cache line due to timestamp expiration does not require explicit events; instead the read to that line is simply treated as a miss after the timestamp expires. A read also misses on L1 Invalid lines and the L2 cache has to be accessed. Upon
10 receiving a read request, the L2 cache will predict a life time for the requester if it gets a shared copy of the line. The choice of lifetime value is important as too short predicted lifetime will result in premature expirations and repeated L2 accesses. On the other hand, too long predicted lifetime will require long wait at release points. After every lifetime prediction, the L2 cache updates the line's
15 global timestamp to maintain the maximum timestamp among the copies. For a L2 read on Shared lines, the L2 directly responds with the data and a predicted timestamp to the requester. In the case of an L2 read on Exclusive lines, the request with the predicted timestamp is forwarded to the owner, who downgrades its exclusive copy to Shared and changes the local timestamp in the
20 line with the predicted timestamp. The owner then sends the data with the new predicted timestamp to the original requester who updates its data and local timestamp, with a transition to Shared state. A read operation on L2 Invalid lines

gives the requester exclusive ownership, resulting both the L1 cache lines and L2 cache lines in Exclusive state.

During a read-acquire operation, the read-acquire operation checks if the synchronization variable has been released, otherwise it makes the core to spin-
5 wait until it observes a release performed by another core. A read-acquire operation can be executed similar to a normal write (though a read-acquire does not modify data), which gains the L1 cache line with exclusive ownership. If the acquired synchronization variable has not been released, the core will spin locally in L1 cache, i.e. read the L1 cache line again and again like a directory
10 coherence protocol. The spin waiting stops once another core performs a write-release operation. This is because the core performing the release operation sends write request to L2 cache, which is forwarded to the core that is spin-waiting because it is the exclusive owner. The spin waiting core invalidates the L1 cache line and hence it receives the new value of the synchronization
15 variable on the next read in its spin-wait. This guarantees forward progress in the presence of synchronization.

As for eviction operation, evictions of L1 Shared lines are silent. A L1 eviction of Exclusive/Modified lines needs to inform the L2 cache which changes the state to Shared (as there can be other stale Shared copies in L1 caches with
20 unexpired timestamp). For L2 evictions, only cache lines with expired global timestamps can be evicted to maintain inclusion property. Unexpired timestamps are stored in L2 MSHR entries to eliminate stalling on evictions. Note that an eviction of L2 Exclusive line needs to invalidate the owner in L1 cache.

It should be noted that the description above about normal write operation, normal read operation, read-acquire operation and eviction operation only explains the data classification scheme in different operations, i.e. different operations when a state field is added in each cache line of each private cache
5 and each shared cache in the system to indicate current status of data in each cache line. The combination of the data classification scheme and the signature scheme will be explained in another embodiment shown in Figure 5.

In a timestamp coherence protocol, if a release operation has been observed by the corresponding acquire operation, the write operations before the
10 release operation are made visible to the acquire core since it will self-invalidate the stale cache lines. However, self-invalidating the cache lines again before the core performs another acquire operation is not required. This is illustrated with an example shown in Figure 4, in which two different cores communicate the value of data A. In initial state, data A is located in a Shared line in L1 cache of
15 Core 1. After Core 1 successfully acquires the synchronization variable T, the first read operation on A (R1) finds the expired timestamp and self-invalidates the cache line. R1 then gets the up-to-date value with a predicted timestamp from the L2 Cache. Before performing another acquire operation, core 1 executes another read operation on A (R2). The timestamp has already expired;
20 but self-invalidating the cache line is not necessary because core 1 has already obtained the up-to-date value from core 0 (via L2) on the first read of A.

In order to reduce redundant self-invalidations due to timestamp expirations, a Timestamp Bypass (TB) bit is added into each L1 cache line, as

shown in Figure 3. The TB bit of an L1 cache line is set after its self-invalidation. For a read operation on L1 Shared lines, the TB bit is examined first before the timestamp check: a read operation is allowed to hit in L1 when the TB bit is set, bypassing the timestamp check even if it has expired. To ensure the Acquire ->
5 Reads/Writes ordering, all the TB bits are reset after a read-acquire operation.

Figure 5 is a block diagram illustrating structure of a private cache and a shared cache according to another embodiment of the invention. In this embodiment, the private cache L1 cache 510 further includes a Write FIFO 513, and a state field is added into each cache line of the private cache and the
10 shared cache, i.e. both the signature scheme and the data classification scheme described above are applied in this embodiment.

The Write FIFO 513 configured to record a local write miss which is returned with a timestamp by the shared cache L2 cache 520.

When a normal write operation hits on L1 cache lines in
15 Exclusive/Modified state, for a local write miss which is returned with a timestamp by the L2 cache 520, an entry which is formed by combining the write address with the returned timestamp is queued at the tail of the write FIFO 513. In one example of the embodiment as shown in Figure 5, a write operation on address C in L1 cache 510 is returned with a timestamp from the L2 cache 320
20 and therefore, an entry is queued to the W-FIFO 513. When the entry reaches head of the W-FIFO 513, it will replace an old entry at the head. If the replaced old entry has an unexpired timestamp, the address C is added into the LWS of the L1 cache 510. The LWS will also update its timestamp if the replaced old

entry has a longer timestamp. For an addition to the LWS, the signature is cleared first if its timestamp has expired. With the use of the W-FIFO 513, the size of write operation recorded in the LWS is reduced.

During a write-release operation, the L1 cache 510 triggers a W-FIFO
5 flush signal which dequeues the W-FIFO 513 until it reaches the head. The address corresponding to each evicted entry is added into the LWS and updates the signature's timestamp if the evicted entry has an unexpired timestamp. After the W-FIFO flush completes, the L1 cache 510 will send a release request (REL) containing the LWS to the L2 cache 520, according to the address of the
10 released synchronization variable. Note that if the RWS in the L1 cache 510 has not expired, it will be merged into the signature in the REL. This guarantees the transitivity property some programs may rely on. The timestamp of the signature will become the maximum timestamp of LWS and RWS, which also applies to other signature merge discussed later.

15 Upon receiving the REL in the L2 cache 520, it merges the received signature into the Global Write Set (GWS) signature. Note that a signature clear is performed in the GWS first if it has an expired timestamp. The L2 cache 520 then sends an acknowledgment to the requester, signalling the L1 cache 510 to proceed to the write part of the release operation, which is treated as a normal
20 write.

During a read-acquire operation, in order to make all write operations preceding the corresponding release visible to the acquiring core, the acquiring core needs to obtain the relevant signature in the L2 cache 520. As mentioned

earlier, a read-acquire may spin locally in L1 cache 510 if the synchronization data is still held by another core, which may result in repeated L2 accesses for obtaining the relevant signature. To address this issue, two new stable states are introduced in one embodiment of the invention: Exclusive A and Modified A in
5 the L1 cache 510 to distinguish normal private cache lines from those involved in spin-waiting. A read-acquire operation on L1 cache lines in these two states is not required to obtain the signature. A normal read or write operation will hit on L1 cache lines in Exclusive A/Modified A state, with normal write operations transitioning the cache line to Modified. The added Exclusive A/Modified A state
10 in private cache can also help to reduce the Timestamp Bypass (TB) bits resets in the private cache line, as a read-acquire operation involved in spin-waiting does not need to reset the TB bits.

A read-acquire misses on L1 Invalid or Shared lines, and an acquire request (ACQ) is sent to the L2 tile based on the address of the acquired
15 synchronization variable. An ACQ is similar to a GetX, with the difference that the L2 cache 520 also needs to transfer the GWS in the exclusive data response. After the L1 cache 510 receives the response, the L1 line transitions to Exclusive A state.

A read-acquire operation can hit on L1 cache lines in Exclusive/Modified
20 state, but an ACQ must be sent to the L2 cache 520 first, as the synchronization data may have been released but subsequently fetched to the L1 cache 510 by normal read/write operations. Since the L1 cache 510 is the current owner of the cache line, in this case the L2 cache 520 only needs to respond with the GWS

(and no data is transferred). After receiving response from the L2 cache 520, the read operation hits on L1 cache 510 and transitions the cache line from Exclusive or Modified state to Exclusive A or Modified A state respectively.

5 L1 cache lines in Exclusive A or Modified A allow a read-acquire operation to hit locally without sending an ACQ to the L2 cache 520, as the core is probably spin-waiting. The L1 cache line will be eventually invalidated by a release operation, hence a legitimate ACQ will be sent for the following read (within read-acquire spinning) to the Invalid cache line.

10 When the L1 cache 510 receives the response for an ACQ, L1 cache 510 merges the obtained signature to the RWS, which will be checked for subsequent normal read operations on cache lines in Shared state.

During a normal read operation, i.e. a normal read hits on L1 cache lines in Shared state with an unexpired timestamp, the L1 cache 510 checks the RWS signature to determine if the data has been modified by a remote core. As shown
15 in the example in Figure 5, the read operations on Address A and Address B are required to consult the RWS. If the address hits in the signature, the cache line is self-invalidated and a read request will be sent to L2 cache 520. During a check on the signature, a signature clear is performed if possible.

For a read operation on L1 cache lines in Shared state with the TB bit set,
20 the read operation is considered as a hit and the checks on both the timestamp and the signature are bypassed.

As will be appreciated from the above, with the signature scheme provide in the embodiments of the invention, an unexpired write operation is maintained by a predefined signature in both each private cache and the shared cache in the system such that the release operation does not need to wait for all of the global timestamps expire. That is to say, the codes or instructions to be executed by a core in the system after the release operation may be executed earlier since it is not necessary to wait for all of the global timestamps expire. Further, in order to further improve cache performance, a state field is added to each private cache line and each shared cache line to indicate the current status of the corresponding data. Also, a TB bit is added into each cache line of the private cache to reduce redundant self-invalidations due to expiration of a timestamp.

It is to be understood that the embodiments and features described above should be considered exemplary and not restrictive. For example, the above-described embodiments may be used in combination with each other. Many other embodiments will be apparent to those skilled in the art from consideration of the specification and practice of the invention. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled. Furthermore, certain terminology has been used for the purposes of descriptive clarity, and not to limit the disclosed embodiments of the invention.

Claims

1. A method for timestamp-based cache coherence in a system comprising a plurality of cores, a plurality of private caches, each of the private caches for storing/recording unexpired write operations initiated by a local core in a Local
5 Write Set (LWS) and by at least one remote core of the system in a Remote Write Set (RWS), and at least one shared cache each configured for storing/recording unexpired write operations associated with a predetermined group of memory addresses released by the private caches in the system in a Global Write Set (GWS), the method comprising:

10 in a write operation,

 writing, by a first private cache, a data stored in a memory address to a first corresponding shared cache; returning, by the first corresponding shared cache, a timestamp of the data to the first private cache; determining, by the first private cache, whether the timestamp is expired based on a current system time;
15 and adding, by the first private cache, the address associated with the data into a Local Write Set (LWS) of the first private cache; and

 in a release operation,

 merging, by a second private cache, the RWS and the LWS in the second private cache and releasing the merged RWS and LWS to a second
20 corresponding shared cache; merging, by the second corresponding shared cache, the RWS and LWS received from the second private cache with the GWS of the second corresponding shared cache.

2. The method according to claim 1, further comprising:

in an acquire operation,

5 sending, by a third corresponding shared cache, a requested data and the GWS in the third corresponding shared cache to a third private cache which requests the data; acquiring, by the third private cache the requested data and merging, by the third private cache ,the received GWS with RWS of the third private cache;

10 in a read operation, checking, by a fourth private cache, whether a memory address of a data is recorded in RWS of the fourth private cache, self-invalidating the data and loading the data from a fourth corresponding shared cache if the address of the data is recorded in the RWS of the fourth private cache.

3. The method according to claim 2, wherein the system comprises one shared cache.

15 4. The method according to any preceding claim , wherein each of the LWS, RWS and GWS has a predefined signature data structure which includes a timestamp and a filter field, wherein the filter field is a bit vector having a plurality of bits, wherein a predetermined group of bits in the bit vector indicate whether any one of a predetermined group of addresses has been written by a core in the system and is not globally visible in the system, wherein the predetermined
20 group of bits include at least one bit.

5. The method according to claim 4, wherein when two of the LWS, the RWS and the GWS are merged with each other to generate a merged signature, an “OR” logical operation is performed on bit vectors of two signature data structures of two of the LWS, RWS and GWS to be merged, and a longer one of
5 timestamps of the two signature data structures is set as the timestamp of the merged signature.

6. The method according to claim 4 or claim 5, further comprising:

if a timestamp of a signature data structure expires, clearing the filter field of the signature data structure.

10 7. The method according to any preceding claim, wherein any one of the LWS, the RWS and the GWS is a Bloom filter.

8. The method according to any preceding claim, further comprising:

maintaining, by each private cache, a write FIFO queue to record a local write miss which is returned with a timestamp by one of the at least one shared
15 cache,

in a write operation, the method further comprising: queuing a local write miss at end of the write FIFO in a fifth private cache, and when the local write miss reaches head of the write FIFO, adding, by the fifth private cache, the address associated with the local write miss into LWS of the fifth private cache if
20 the local write miss has an unexpired timestamp.

9. The method according to any preceding claim, further comprising:

each cache line in a private cache includes a state field for indicating a current status of data in the cache line, wherein the current status is one of four stable states: Invalid, Shared, Exclusive, and Modified;

5 each cache line in a shared cache includes a state field for indicating a current status of data in the cache line, wherein the current status is one of three stable states: Invalid, Shared, Exclusive

10. The method according to claim 9, wherein each cache line in a shared cache further includes an owner field for indicating an exclusive owner.

11. The method according to any preceding claim, further comprising:

10 adding a Timestamp Bypass (TB) bit into each cache line in each private cache to reduce redundant self-invalidations due to expiration of a timestamp.

12. A system for timestamp-based cache coherence, comprising:

a plurality of cores,

15 a plurality of private caches each configured to store/record unexpired write operations initiated by a local core in a Local Write Set (LWS) and by at least one remote core in the system in a Remote Write Set (RWS), and

at least one shared cache, each shared cache configured to store/record unexpired write operations associated with a predetermined group of memory addresses released by the private caches in a Global Write Set (GWS),

wherein each private cache is further configured to, in a write operation, write a data stored in a memory address to a first corresponding shared cache; determine whether a timestamp of the data returned by the first corresponding shared cache expires based on a current system time, and add the address in
5 which the data is stored into LWS of the private cache; the first corresponding shared cache is further configured to return a timestamp of a data to the private cache;

each private cache is further configured to, in a release operation, merge the RWS and LWS in the private cache and release the merged RWS and LWS
10 to a second corresponding shared cache; the second corresponding shared cache is further configured to merge the RWS and LWS received from the private cache with GWS of the second corresponding shared cache.

13. The system according to claim 12, wherein each of the at least one shared cache is further configured to, in an acquire operation, send an requested data
15 and GWS of a third corresponding shared cache to a corresponding private cache which requests the data; and each private cache is further configured to acquire a data and merge received GWS with RWS of the private cache; and

each private cache is further configured to, in a read operation, check whether a memory address of a data is recorded in RWS of the private cache,
20 self-invalidate the data and load the data from a fourth corresponding shared cache if the address of the data is recorded in RWS of the private cache.

14. The system according to claim 13, wherein the system comprises one shared cache.

15. The system according to any one of claim 13 to claim 14, wherein each of the LWS, RWS and GWS has a predefined signature data structure, which
5 includes a timestamp and a filter field, wherein the filter field is a bit vector having a plurality of bits each bit indicating whether any one of a predetermined group of addresses has been written by a core in the system and still not globally visible in the system.

16. The system according to claim 15, wherein each private cache and each
10 shared cache are further configured to generate an merged signature by performing an "OR" operation on bit vectors of two signature data structures of two of the LWS, RWS and GWS to be merged and set a longer one of timestamps of the two signature data structures as a timestamp of the merged signature.

15 17. The system according to claim 15 or claim 16, further comprising:

each private cache and each shared cache are further configured to clear filter field of a signature data structure if a timestamp of the signature data structure expires.

18. The system according to any preceding claim, wherein any one of the LWS,
20 RWS and GWS is a Bloom filter.

19. The system according to any preceding claim, wherein

each private cache is further configured to maintain a write FIFO queue to record a local write miss which is returned with a timestamp by a shared cache, and

5 each private cache is further configured to, in a write operation, queue a local write miss at end of the write FIFO in the private cache, and when the local write miss reaches head of the write FIFO, add the address associated with the local write miss into LWS of the private cache if the local write miss has an unexpired timestamp.

20. The system according to any preceding claim, wherein

10 each private cache is further configured to maintain a state filed in each cache line to indicate a current status of data in the cache line, wherein the current status is one of four stable states: Invalid, shared, Exclusive, and Modified;

15 each shared cache is further configured to maintain a state field in each cache line to indicate a current status of data in the cache line, wherein the current status is one of three stable states: Invalid, shared, Exclusive.

21. The system according to claim 20, wherein each shared cache is further configured to maintain an Owner field in each cache line to indicate an exclusive owner.

20 22. The system according to any preceding claim, wherein

each private cache is further configured to maintain a Timestamp Bypass (TB) bit in each cache line to reduce redundant self-invalidations due to expiration of a timestamp.

23. An apparatus for timestamp-based cache coherence, comprising:

5 a Local Write Set (LWS) configured to store/record unexpired write operations initiated by a local core which is comprised in a system which further comprises at least one remote core, and at least one shared cache each configured to store/record unexpired write operations associated with a predetermined group of memory addresses in a Global Write Set (GWS), and

10 a Remote Write Set (RWS) configured to store/record unexpired write operations initiated by at least one remote core in the system;

 wherein the apparatus is configured to, in a write operation, write a data stored in a memory address to a first corresponding shared cache in the system; determine whether a timestamp of the data returned by the first corresponding
15 shared cache expires based on a current system time, and add the address in which the data is stored into the LWS;

 the apparatus is further configured to, in a release operation, merge the RWS and LWS and release the merged RWS and LWS to a second corresponding shared cache in the system.

20 24. The apparatus according to claim 23, wherein the first corresponding shared cache and the second corresponding shared cache are a same shared cache in the system.

25. An apparatus for timestamp-based cache coherence comprising:

a Global Write Set (GWS) configured to store/record unexpired write operations associated with a predetermined group of memory addresses released by a plurality of private caches which are comprised in a system which
5 further comprises a plurality of cores, wherein each private cache is configured to store/record unexpired write operations initiated by a local core in a Local Write Set (LWS) and by at least one remote core of the system in a Remote Write Set (RWS),

wherein in a write operation, the apparatus is configured to return a
10 timestamp of a data to a requesting private cache;

in a release operation, the apparatus is further configured to merge RWS and LWS received from a private cache with the GWS.