Title: RESOLVING CACHE CONFLICTS

Abstract: Preventing cache conflicts within microprocessors and/or computer systems. More particularly, embodiments of the invention relate to a technique to manage cache conflicts within a processor and/or computer system in which a number of accesses may be made to a particular cache or group of caches.
RESOLVING CACHE CONFLICTS

FIELD

[0001] Embodiments of the invention relate to microprocessors and microprocessor systems. More particularly, embodiments of the invention relate to resolving cache access conflicts within a processor or computer system in which a number of accesses occur to the same cache or group of caches.

BACKGROUND

[0002] Prior art processors and computer systems may be limited in the number of accesses to a particular cache or group of caches that can be concurrently managed. One prior art technique used to combat this problem has been the use of an inclusive cache structure whose cache entries correspond to the cache entries of one or more processor core-specific caches, such as level 1 (L1) caches. In other words, prior art multi-core processors and/or multi-processor computer systems have attempted to reduce cache access conflicts within core caches by simply directing some of the cache accesses to a shared inclusive cache structure, such as a last level cache (LLC), that contains all of the cache entries of the processor cores or agents to which the inclusive cache structure corresponds. In the case of a cache access from a core within a multi-core processor, however, the core will typically attempt to access data first from its own cache and then resort to the shared cache. The shared inclusive cache structure is sometimes referred to as a "cache filter", as it shields core caches from excessive cache accesses, and therefore bus traffic, from other agents by
providing the requested data to these agents from the inclusive cache instead of the core’s cache.

[0003] The prior art technique of using a cache structure, such as an LLC, for servicing cache requests from various agents is helpful in allowing requesting agents to obtain the data they need without resorting to a cache of a processor core, for example, if the data is not exclusively owned or modified by a particular processor core. To the extent that an agent, such as a processor or processor core owns the cache line of its cache that the requesting agent is trying to access, a cache structure, such as an LLC, can allow the requesting agent to obtain the data it is requesting rather than waiting for the owning agent to share the data.

[0004] However, other conflicts can occur when using an LLC to service cache requests. Figure 1, for example, illustrates two cores attempting to access the same cache line of an LLC during an eviction of the accessed line from the LLC. Particularly, core 0 has initiated a core cache request to a line in core 1’s cache (via an LLC snoop) at substantially the same time as the line is being evicted from the LLC, while core 1 is initiating a writeback of new data. In this case, core 0 may retrieve erroneous data from the LLC if core 0’s request is made before the writeback from core 1 has taken place. In some cases, a snoop may need to be done by the LLC to core 1’s cache ("cross snoop") in order to fulfill the core request of core 0, resulting in a four-way cache conflict between core 0’s request, LLC’s cross snoop to core 1, the LLC eviction, and core 1’s writeback of updated data to the LLC.

[0005] The prior art problem depicted in Figure 1 is exacerbated as the number of processor cores or other bus agents increases in the system. For example, the conflicts depicted in Figure 1 may double in a multi-core processor containing four
cores instead of the two illustrated in Figure 1. Similarly, as the number of processors increase in a computer system, so does the number of accesses to any particular core cache, thereby increasing the number of conflicts that can occur during an LLC eviction.

[0006] Cache conflicts, such as those depicted in Figure 1, can have adverse effects on processor performance as requesting agents either wait for the LLC eviction and corresponding write-back to complete, or detect and recover from retrieving incorrect data as a result of the conflict. Accordingly, the number of agents that may access a particular cache structure can be limited in prior art processor and/or computer systems.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention are illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

[0007] Figure 1 illustrates a conflict between a number of access to the same cache line in a prior art processor or computer system.

[0008] Figure 2 illustrates a caching bridge architecture according to one embodiment of the invention.

[0009] Figure 3 illustrates a cross snoop state machine for a processor used in conjunction with one embodiment of the invention.

[0010] Figure 4 is a flow diagram illustrating operations used in conjunction with at least one embodiment of the invention.

[0011] Figure 5 illustrates a front-side bus computer system in which at least one embodiment of the invention may be used.
Figure 6 illustrates a point-to-point computer system in which at least one embodiment of the invention may be used.

DETAILED DESCRIPTION

[0013] Embodiments of the invention relate to caching architectures within microprocessors and/or computer systems. More particularly, embodiments of the invention relate to a technique to manage cache conflicts within a processor and/or computer system in which a number of accesses may be made to a particular cache or group of caches.

[0014] This disclosure describes various embodiments of the invention to address problems associated with prior art caching techniques in multi-processor and/or multi-core computer systems, including conflict resolution and avoidance when a number of requesting agents attempt to access the same line of cache. In at least one embodiment of the invention, an inclusive cache structure, such as a last level cache (LLC), is used in conjunction with a number of processors or processor cores having an associated cache, such as a level 1 (L1) cache. Inclusive cache structures, such as an LLC, include those that contain at least the same data as other caches to which the inclusive cache structure corresponds. By maintaining coherence between the inclusive cache structure and corresponding core and/or processor caches, accesses to the corresponding core/processor caches are serviced by the inclusive cache, thereby reducing bus traffic to the corresponding cores/processors and allowing the cores/processors.

[0015] Embodiments of the invention, in which an inclusive cache structure is used, can also reduce or even alleviate the number and/or types of conflicts that can occur when a number of processors and/or processor cores attempt to access
the same line of cache within the inclusive cache structure. For example, at least one embodiment of the invention alleviates cache conflicts resulting from a cache request from a processor, in a multi-processor system and/or from a core, in a multi-core processor, to a line within an inclusive cache structure, such as an LLC, that is being evicted as a core cache, that is-being evicted and result of another fill to the LLC to the same set, and a write back from a core which has been evicted line corresponds. Furthermore, at least one embodiment alleviates conflicts resulting from a cache request from a processor, in a multi-processor system, and/or from a core, in a multi-core processor, to a line within a shared inclusive cache, such as an LLC, that is being filled and the resulting eviction of the shared inclusive cache line. Other embodiments may resolve other conflicts resulting from multiple accesses to an evicted inclusive cache line from various requesting agents.

[0016] Figure 2 illustrates a caching bridge architecture, according to one embodiment of the invention, which resolves conflicts among a number of accesses to an evicted inclusive cache line. Specifically, the cache bridge architecture of Figure 2 illustrates an LLC 201 that may be accessed by external agents via a computer system interconnect interface 205, such as a front-side bus interface or a point-to-point interface. Furthermore, the LLC may be accessed by core 0 210 and/or core 1 215 via core interconnect interfaces 213 and 217, respectively. The cache bridge scheduling and ordering (CBSO) logic 220 manages the accesses to the LLC from the external and core agents, in at least one embodiment of the invention, using internal and external request queues 225 and 230, respectively, which can be used to store command, address, and/or data corresponding to access to the LLC made by the external and/or core agents.
[0017] In at least one embodiment of the invention, the CBSO logic may be used to manage and resolve conflicts resulting from a number of transactions, including an LLC look-up, LLC cache eviction, LLC line fills, and cross snoop transactions.

[0018] An LLC look-up, typically involves read and read-for-ownership transactions from the cores accessing the LLC to read or gain ownership of a desired line of cache. If the LLC look-up results in a miss, the request may be allocated to the external request queue corresponding to the computer system interconnect interface. If the LLC look-up resulting in a hit, however, and the corresponding LLC line is not exclusively owned by another core or processor then the request can be completed and data returned to the requesting core. Accesses to a particular core from a requesting agent may be reduced by maintaining a record of whether another core has exclusive ownership of a requested line of the LLC. The record may be a number of bits in a register corresponding to the number of cores in a processor, each bit indicating whether a core/processor to which it corresponds has ownership of a requested LLC line. However, the record may be implemented in other ways.

[0019] An LLC eviction, may require a snoop ("back snoop") to one or more cores or processors to refill the LLC cache line. If the back snoop is sent to multiple cores or processors, there may be situations in which one or more cores/processors does not receive the back snoop. Accordingly, conflicts may result.

[0020] Fills to the LLC typically result from a core or processor writing data to an LLC if the original request missed the LLC. New data and coherence state can be obtained from a memory agent which can be an on-die memory controller or
off-die memory controller. This line then filled in to the LLC after returning the new data and coherence state to the requesting core. If the cache set into which the fill is occurring is full, an eviction is caused from LLC, this eviction is sometimes referred to as a "capacity eviction" as it is caused due to the capacity limitations in the LLC. Fills may originate from cores within a multi-core processors, depending on the core to which the LLC line to be filled corresponds. Furthermore, in one embodiment of the invention, the filled line of the LLC may be in a number of ownership states, such as shared, exclusive or modified. In some multi-core processors the LLC coherency states may include extended states to indicate the state of the cache line to the cores versus the state of the cache line to agents external to the multi-core processor. For example, in some embodiments, LLC coherency state ES to indicate to agents external to the multi-core processor that the filled LLC line is exclusively owned by a particular core, while indicating to other cores that the filled LLC line is shared. Similarly, the MS coherency state may indicate to external agents that the LLC line is modified while indicating to the cores that the LLC line is shared.

[0021] Cross snoop transactions to the LLC typical result when an ownership request from a core or other agent determines that the LLC line is owned by another core or agent. In this case, the core/agent requesting ownership will perform a snoop to the core/agent ("cross snoop") owning the line, which can result in the line state changing from "exclusive" to "invalid" or "shared", depending on the particular coherency protocol being used.

[0022] If any of the above transactions (back snoops, cross snoops, reads, and evictions) occur substantially simultaneously, conflicts may arise that have adverse effects on processor and/or system performance. Accordingly, one
embodiment of the invention prevents, or at least manages, conflicts between two of these transactions ("two-way conflict" management). Furthermore, another embodiment of the invention prevents, or at least manages, conflicts between three of these transactions ("three-way conflict" management).

5 [0023]  In one embodiment of the invention, the CBSO logic manages, or prevents, conflicts resulting from a write back to the LLC from a core or external bus agent to a line being evicted from the LLC. In the case that a writeback is occurring to a same LLC line being evicted, a conflict can occur between the back snoop resulting from the eviction and the writeback operation if the back snoop is retrieving data from another core or agent than the one performing the writeback. The conflict can result in incorrect data being written to the evicted LLC line.

[0024]  In another embodiment, the CBSO logic manages, or prevents, conflicts resulting from a snoop to an LLC line from an agent on the computer system interface of Figure 2, a writeback to the LLC line from a core, and an LLC back snoop to fill the line. In the case that an external snoop occurs to the same LLC line for which a back snoop and writeback is occurring, the external agent could retrieve incorrect data, because LLC line could be filled with either the writeback from a core, or data from a core resulting from the back snoop.

[0025]  Figure 3 is a state diagram illustrating operations associated with a typical cross snoop transaction, according to one embodiment of the invention. From the idle state 301, a read transaction to the LLC, such as from a core within a multi-core processor, causes the state diagram to transition to pending state 303 until the line can be granted to the requesting agent, at which time the state changes to the look-up state 305. During the look-up state, the LLC returns the coherency state of the requested line to the requesting core, which can indicate
that another core currently owns the requested line. In the case that another core owns the requested line in the LLC, a cross snoop from the LLC to other cores or agents is initiated at state 308. After an acknowledgement is sent from the core to which the cross snoop to which the cross snoop is to initiated, the cross snoop is issued at state 310. After the cross snoop data is retrieved from the core, the cross snoop is complete at state 313 and the cross snoop data is delivered to the requesting core at state 315. The LLC is updated with the cross snoop data at state 320 and returned to the idle state.

During states 308 to 320, the cross snoop can experience conflicts with operations resulting from an eviction of the LLC to which the request corresponds. One operation resulting from an LLC eviction that can conflict with the cross snoop is a writeback from a core to which the evicted LLC line corresponds. Another conflict can occur if a read request causes a cross snoop to a core from which a writeback is to be made to a line evicted in the LLC. If the writeback occurs before the cross snoop, the wrong data may be returned to the requesting core or agent. Furthermore, a conflict can occur in the case where an external snoop is made to the LLC at approximately the same time involving the same LLC address as an eviction, a cross snoop, and a writeback.

In one embodiment of the invention, the above-mentioned conflicts can be avoided by copying the coherence information of the line to which a request is made in the LLC to a temporary storage location and invalidating the corresponding LLC line, such that the line will appear invalid to subsequent transactions ("atomic"), thereby avoiding an eviction of the LLC line that may result in transactions conflicting with a cross snoop resulting from the request. By storing the LLC line coherency information after receiving a read request, the
resulting cross snoop is guaranteed to deliver the most recent data to the requestor. Furthermore, by atomically invalidating the LLC line, eviction of the LLC is avoided by subsequent transactions, and therefore, no conflicting LLC evictions will occur to the LLC line.

[0028] After delivering the requested data to the requestor, the data and coherency information may be stored to the invalidated LLC line to preserve inclusion. In an alternative embodiment, a mechanism may be used to cancel any transactions that may prevent an access to the LLC from resulting in a cross snoop. This condition may arise, for example, if a writeback to an LLC line occurs after a read to the LLC line.

[0029] Figure 4 is a flow diagram illustrating operations involved in one embodiment of the invention. At operation 401, a read request is detected to a core cache line and the corresponding LLC line is accessed in response thereto if a “miss” results from a read request to the corresponding core cache. At operation 405, the coherency state information of the LLC line is saved. In one embodiment, the coherency state data is saved to a register within the CBSO logic of Figure 2. In other embodiments, the coherency information may be saved to memory or some other storage structure. After the coherency state information is saved, the corresponding line in the LLC is atomically invalidated at operation 410, such that subsequent transactions will see the LLC line as invalidated, if the request will cause a cross snoop and no cancellation signal was detected by the CBSO logic. A cross snoop by the LLC to the appropriate core or processor will return the requested data from the core or processor to the requesting agent at operation 415.
In one embodiment of the invention, at least some of the operations illustrated in Figure 4 are performed by the CBSO logic of Figure 2. In other embodiments, the operations may be performed by other means, such as software, or some other logic within the cache bridge architecture of Figure 2.

Figure 5 illustrates a front-side-bus (FSB) computer system in which one embodiment of the invention may be used. A processor 505 accesses data from a level one (L1) cache memory 510 and main memory 515. In other embodiments of the invention, the cache memory may be a level two (L2) cache or other memory within a computer system memory hierarchy. Furthermore, in some embodiments, the computer system of Figure 5 may contain both a L1 cache and an L2 cache, which comprise an inclusive cache hierarchy in which coherency data is shared between the L1 and L2 caches.

Illustrated within the processor of Figure 5 is one embodiment of the invention 506. In some embodiments, the processor of Figure 5 may be a multi-core processor.

The main memory may be implemented in various memory sources, such as dynamic random-access memory (DRAM), a hard disk drive (HDD) 520, or a memory source located remotely from the computer system via network interface 530 containing various storage devices and technologies. The cache memory may be located either within the processor or in close proximity to the processor, such as on the processor’s local bus 507. Furthermore, the cache memory may contain relatively fast memory cells, such as a six-transistor (6T) cell, or other memory cell of approximately equal or faster access speed.

The computer system of Figure 5 may be a point-to-point (PtP) network of bus agents, such as microprocessors, that communicate via bus signals
dedicated to each agent on the PtP network. Within, or at least associated with,
each bus agent is at least one embodiment of invention 506, such that store
operations can be facilitated in an expeditious manner between the bus agents.

[0035] Figure 6 illustrates a computer system that is arranged in a point-to-
point (PtP) configuration. In particular, Figure 6 shows a system where
processors, memory, and input/output devices are interconnected by a number of
point-to-point interfaces.

[0036] The system of Figure 6 may also include several processors, of which
only two, processors 670, 680 are shown for clarity. Processors 670, 680 may
each include a local memory controller hub (MCH) 672, 682 to connect with
memory 62, 64. Processors 670, 680 may exchange data via a point-to-point
(PtP) interface 650 using PtP interface circuits 678, 688. Processors 670, 680
may each exchange data with a chipset 690 via individual PtP interfaces 652, 654
using point to point interface circuits 676, 694, 686, 698. Chipset 690 may also
exchange data with a high-performance graphics circuit 638 via a high-
performance graphics interface 639.

[0037] At least one embodiment of the invention may be located within the
processors 670 and 680. Other embodiments of the invention, however, may
exist in other circuits, logic units, or devices within the system of Figure 6.

Furthermore, other embodiments of the invention may be distributed throughout
several circuits, logic units, or devices illustrated in Figure 6.

[0038] Embodiments of the invention described herein may be implemented
with circuits using complementary metal-oxide-semiconductor devices, or
“hardware”, or using a set of instructions stored in a medium that when executed
by a machine, such as a processor, perform operations associated with
embodiments of the invention, or "software". Alternatively, embodiments of the invention may be implemented using a combination of hardware and software.

While the invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense.

Various modifications of the illustrative embodiments, as well as other embodiments, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.
CLAIMS

What is claimed is:

1. An apparatus comprising:
   ordering logic to prevent cache access conflicts between a read
   access to a first line of cache and a writeback access to the first line of cache.

2. The apparatus of claim 1 wherein the line of cache is within a shared
   inclusive cache memory.

3. The apparatus of claim 1 wherein the read access to the first line of cache
   is a cross snoop access from a shared inclusive cache memory to a core cache
   memory.

4. The apparatus of claim 1 wherein a cache line fill to the first line of cache
   causes an eviction of a line of cache within a shared inclusive cache memory.

5. The apparatus of claim 1 wherein the read access is from a first core in a
   multi-core processor and the writeback access is from a second core in the multi-
   core processor.

6. The apparatus of claim 1 wherein the read access is from a first processor
   in a multi-processor system and the writeback is from a first core in a multi-core
   processor.
7. The apparatus of claim 1 further comprising a storage unit to temporarily store coherency state information pertaining to the first line of cache in response to the read access.

8. The apparatus of claim 7 wherein the coherency state information includes at least one bit to indicate that the first line of cache is atomically invalidated as a result of the read access.

9. A system comprising:

   a first processor comprising a shared inclusive cache, the shared inclusive cache including a first cache line having an invalid state in response to a snoop from another core within the first processor, the invalid state to indicate to all transactions subsequent to the read access, but before the read access is complete, that the first cache line is invalid.

10. The system of claim 9 comprising a storage unit to store coherency information of the first cache line in response to the snoop.

11. The system of claim 10 comprising a plurality of processor cores each having at least one corresponding core cache.

12. The system of claim 11 wherein the shared inclusive cache is a last level cache to store the same data stored within the at least one corresponding core cache.
13. The system of claim 12 wherein the first processor comprises logic to set the invalid state and to store the coherency information in response to the snoop.

14. The system of claim 13 wherein the first processor comprises an internal and external request queue to, respectively, store accesses from and to the plurality of processor cores and a second processor.

15. The system of claim 14 wherein the first and second processors are coupled together via a point-to-point interconnect.

16. The system of claim 14 wherein the first and second processors are coupled together via a front-side bus interconnect.

17. A method comprising:

performing a plurality of accesses to the same line of an inclusive cache structure;

preventing conflicts among the plurality of accesses to the line from a plurality of processor cores, the plurality of accesses being caused by any two transactions of a group consisting of: an external snoop of the line, an eviction of the line, a cross snoop from the inclusive cache, and a writeback to the line.

18. The method of claim 17 wherein the preventing comprises storing coherency information of the line in response to one of the plurality of accesses.
19. The method of claim 18 wherein the preventing further comprises atomically invalidating the line in response to one of the plurality of accesses.

20. The method of claim 19 further comprising restoring the coherency information of the line in response to the one of the plurality of accesses being completed.

21. The method of claim 20 further comprising setting the line to a valid state in response to the one of the plurality of accesses being completed.

22. A processor comprising:

   means for preventing cache access conflicts between a read access to a first line of cache and a writeback access to the first line of cache.

23. The processor of claim 22 wherein the line of cache is within a shared inclusive cache memory.

24. The processor of claim 22 wherein the read access to the first line of cache is a cross snoop access from a shared inclusive cache memory to a core cache memory.

25. The processor of claim 22 wherein a cache line fill to the first line of cache causes an eviction of a line of cache within a shared inclusive cache memory.
26. The processor of claim 22 wherein the read access is from a first core in a multi-core processor and the writeback access is from a second core in the multi-core processor.

27. The processor of claim 22 wherein the read access is from a first processor in a multi-processor system and the writeback is from a first core in a multi-core processor.

28. The processor of claim 22 further comprising a storage unit to temporarily store coherency state information pertaining to the first line of cache in response to the read access.

29. The processor of claim 28 wherein the coherency state information includes at least one bit to indicate that the first line of cache is atomically invalidated as a result of the read access.
FIG. 1
(PRIOR ART)

FIG. 2
A READ REQUEST IS DETECTED TO A CORE CACHE LINE AND THE CORRESPONDING LLC LINE IS ACCESSED IN RESPONSE THERETO IF A "MISS" RESULTS FROM A READ REQUEST TO THE CORRESPONDING CORE CACHE 401

THE COHERENCY STATE INFORMATION OF THE LLC LINE IS SAVED 405

THE CORRESPONDING LINE IN THE LLC IS AUTOMATICALLY INVALIDATED IF THE REQUEST WILL CAUSE A CROSS SNOOP AND NO CANCELLATION SIGNAL WAS DETECTED BY THE CBSO LOGIC 410

A CROSS SNOOP BY THE LLC TO THE APPROPRIATE CORE OR PROCESSOR WILL RETURN THE REQUESTED DATA FROM THE CORE OR PROCESSOR TO THE REQUESTING AGENT 415

FIG. 4
FIG. 5