MESIF: A Two-Hop Cache Coherency Protocol for Point-to-Point Interconnects (2009)

J.R. Goodman
University of Auckland

&

H.H.J. Hum
Intel Corporation

Abstract—*We describe MESIF, the first source-snooping cache coherence protocol. Built on point-to-point communication links, the protocol maintains no directory, and mimics the broadcast behavior of a snooping cache protocol. MESIF supplies data cached in other nodes in a single round-trip delay (2-hop latency) for all common operations. Because of the speed of the links, the protocol can outperform a bus-based protocol for a small number of nodes, but scales through hierarchical extension to a large-scale multiprocessor system. Salient features of the protocol are described. The introduction of a novel forwarding state is used to assure a single response to shared data and to simplify conflict resolution. In the hierarchical extension, auxiliary hardware data structures can be included to provide 2-hop latency for most operations.*

The recently revealed Intel® Quick-Path Interconnect™ protocol is derived from MESIF. Some design differences are highlighted.

The attached manuscript was submitted on 17 November 2009 to be considered for inclusion in the *32nd International Symposium on Computer Architecture (ISCA'10)*. To preserve its historical authenticity, it is presented verbatim (including known errors).

The MESIF protocol described was the first of a collection of MESIF protocols developed at Intel Corporation. A forerunner of the QPI protocol subsequently developed by Intel, it was first proposed in 2001 by Herbert Hum, an Intel employee, and James Goodman, a consultant on sabbatical from the University of Wisconsin-Madison.

MESIF: A Two-Hop Cache Coherency Protocol for Point-to-Point Interconnects

***Abstract*—*We describe MESIF, the first source-snooping cache coherence protocol. Built on point-to-point communication links, the protocol maintains no directory, and mimics the broadcast behavior of a snooping cache protocol. MESIF supplies data cached in other nodes in a single round-trip delay (2-hop latency) for all common operations. Because of the speed of the links, the protocol can outperform a bus-based protocol for a small number of nodes, but scales through hierarchical extension to a large-scale multiprocessor system. Salient features of the protocol are described. The introduction of a novel forwarding state is used to assure a single response to shared data and to simplify conflict resolution. In the hierarchical extension, auxiliary hardware data structures can be included to provide 2-hop latency for most operations.*

The recently revealed Intel® Quick-Path Interconnect™ protocol[1] is derived from MESIF. Some design differences are highlighted.

1. Introduction

There are two well-known categories of cache coherence protocols: directories and snooping caches. Snooping cache coherence protocols distribute the coherence information around the system, making cache controllers responsible for maintaining information about the whereabouts of copies of the memory locations for which they have special privileges. They depend on a broadcast to assure that all the nodes witness events that might require their intervention. While this idea has been extended with the use of a logical bus[2, 3], the basic idea of a broadcast is fundamental to the snooping protocol.

Directory-based schemes depend on a single place—the directory—to store information about a cache line. While this information may be distributed for different addresses, each address is associated with a given location where the coherence information is maintained (or at least, where it can be found[4, 5]). Unfortunately, this means that many common operations require a

minimum of three serial communications to complete a memory transaction, as opposed to two for a snooping protocol. Directory lookup time can be overlapped in fetching uncached data from memory. However, if the requested data is cached—increasingly likely as parallel applications become more commonplace and caches continue to grow in size—an opportunity for supplying data directly and quickly from another cache is lost. If the data is modified in a cache, or if the requester wishes to modify the data, the directory model is even more complicated, requiring communications with other nodes before the requesting node receives a response.

New technologies and energy considerations make broadcast through a common electrical medium less attractive. Point-to-point communications today provide both lower latency and higher bandwidth communication[1,6,7]. In anticipation of this development we investigated the possibility of "broadcasting" through parallel, point-to-point links, achieving the 2-hop latency of snooping protocols while exploiting the higher potential bandwidth of point-to-point links. We achieved this with the MESIF protocol, which provides a 2-hop latency for all the common memory operations. This protocol, and others it inspired, have led directly to the "source snooping" cache coherence protocol present in the QuickPath Interconnect™ (QPI) [1], recently introduced by Intel®. To our knowledge, MESIF was the first protocol that can be described as source snooping on a point-to-point interconnect.

The protocol was designed to exploit the fact that shared data can be retrieved from another cache faster than it can be fetched from memory. Shared data, as well as modified data, is supplied out of a cache rather than fetching it from memory whenever possible.

The goals of the MESIF protocol are the following:

1. High efficiency for a small number of nodes,
2. Two-hop protocol for common memory operations within a leaf cluster,
3. No back-off/retry,
4. Maximize throughput by handling concurrent requests efficiently,
5. Scale to large system, without need for directory protocol,

6. Scalable in a hierarchical manner,
7. Independent of switching fabric, i.e., no ordering requirements, and
8. Support for multiple memory models, including sequential consistency.

The resulting MESIF protocol met all the above goals. It included a hierarchical scheme that allowed for the protocol to be applied recursively to allow a cluster to interact with the rest of the system as if it were a single node. This approach was abandoned in QPI since it can be readily extended with a directory protocol at the cluster level.

An interconnect that provides broadcast capability offers an important advantage: serial observability. The serial nature of a bus provides a global order in which events are observed, even if they are observed at slightly different times. But even if messages along each link are delivered in-order, point-to-point interconnects can still suffer from "time-warp" (see Fig. 1). MESIF recaptures most of the properties of a bus by requiring a sequence of responses, assuring that individual messages, even if delayed, will not prevent the system from achieving serial observability. The protocol includes the concept of a transaction that begins with a broadcast, where a node sends a (nearly identical) message to each of the other nodes, requesting privileges, status, and/or data.

Because broadcasts may overlap, transactions are not explicitly ordered. Rather, each node wishing to read or write data is responsible for assuring the required ordering constraints are met. The node is assisted by receiving a completion acknowledgement for each operation from the Home node (node containing the memory that stores the cache line), notifying it that the contents of the cache line are now visible to the rest of the system. While the node may have previously received the data in a writable state and continued execution, it may not make the result of these changes visible externally until it receives this acknowledgment.

MESIF does not dictate sequential consistency, but supports it, along with more relaxed protocols. Sequential consistency is achieved if all processors commit instructions that modify memory only after receiving the completion acknowledgement (write atomicity) and commit their instructions in-order (write

serialization). Processors can be designed to achieve weaker memory models by relaxing these constraints and making changes visible earlier. They can also weaken the memory model by committing a load instruction using a value from an invalidated cache line after acknowledging the invalidation request.

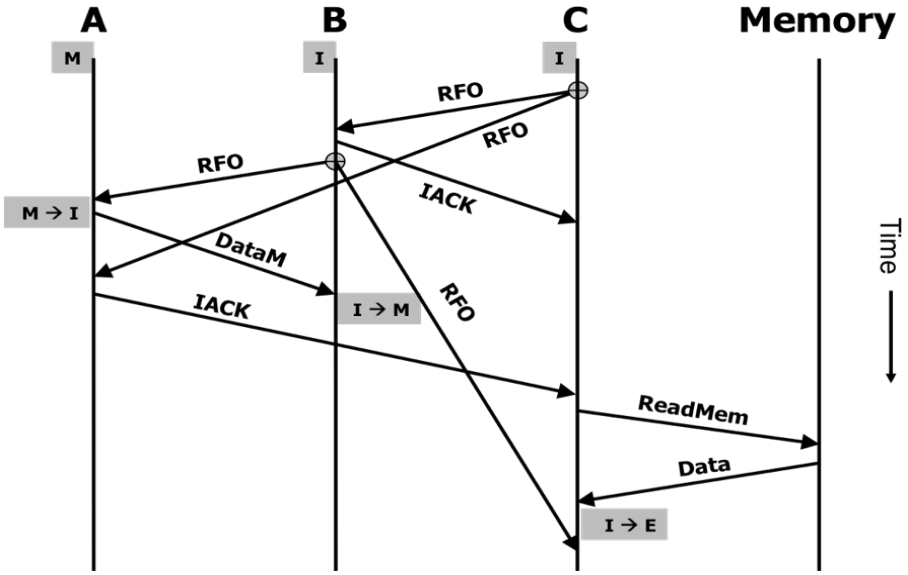


Fig. 1. Time Warp. Node C first initiates a Read-for-ownership (RFO, a transaction requesting an exclusive copy in preparation for a write), sending requests to A and B. Node B responds to the RFO, indicating it does not have the line, later initiating its own RFO for the same line. Node A receives the RFO from B first. Having the data in Modified state, it responds to the request by passing the data to B and invalidates its own copy. Now when it receives RFO from C, it also responds that it does not have the line. C now requests the cache line from memory, having verified that neither A nor B have it. But B has the copy sent from A, creating two incompatible copies.

Request/response assures that every pair of nodes concurrently attempting transactions on the same cache line will become aware of the conflict. The MESIF protocol guarantees that a conflict will be resolved quickly and handled efficiently. While the winner is determined immediately—receiving the data in a cache-to-cache transfer, for instance—the Home node is responsible for assuring that the loser's request is queued and handled efficiently and fairly. The scheme naturally extends to multiple concurrent requests.

In this paper we describe the original MESIF protocol completed in June, 2001. Because of commercial considerations, the protocol remained obscure for the last eight years. With the recent introduction of the QPI protocol, we are now able to describe the MESIF protocol. A very

substantial corporate effort was devoted to establishing both the correctness of the protocol and the performance of this and many other possible implementations, though this information is not publicly available. This document is not a detailed description of the protocol, but rather an attempt to convey its general nature and to suggest possible research directions for assuring cache coherence over a point-to-point network. While some comparisons will be made to QPI to highlight some of the design choices, it is beyond the scope of this document to provide a detailed description of QPI. The reader is referred to [1] for other details of QPI not mentioned in this paper. The QPI book does not detail how a 2-hop protocol can maintain cache coherency in the possible presence of conflicting information that can arise due to the out-of-order arrival of requests and responses. This paper attempts to elucidate the principles of imposing order to an out-of-order point-to-point interconnect.

2. Protocol Description

2.1. The Basic MESIF Protocol

A cache line may be in one of five states in each of the nodes: M (Modified), E (Exclusive), S (Shared), I (Invalid), or F (Forwarding). The first four states correspond to the classic states of a MESI protocol[8]. The Forwarding state indicates a "first-among-equals," that is, a state assigned to a single node among those sharing a cache line, and responsible for providing its content when a request is received. This state is similar to an "Owned" state for modified data, but coherent with memory and useful for read requests rather than write requests. The Forwarding state in MESIF primarily facilitates the rapid response of a cached copy (requestor broadcasts and responder provides a cached copy for a 2-hop latency) in the presence of multiple cached copies. The F state also simplifies the conflict resolution mechanism in the protocol. This will become evident in a later section.

A transaction is initiated by a broadcast to all caching nodes for a cache line, including to the Home node (the node containing the memory location where the cache line resides). All nodes

except Home must respond. A response may include data (from at most one node), and indicates:

1) The state of the cache line in the responding cache, and
2) An indication if a conflicting request (actually a list of conflictors) has been discovered.

In order to guarantee the serializability of transactions, a node supplying data may not respond to any further requests for the same cache line until it has received acknowledgement from the requesting node. The requesting node may use the received data, but it is not allowed to acknowledge receipt from the sender until it receives acknowledgement from Home. This process of acknowledgement chaining allows the Home node to ultimately retain full control of the request; control that is used to sequence conflicting nodes when required.

After responses have been received from all nodes, a second message (second phase) is sent to Home. This message is either a READ message requesting data, or a cancel message (CNCL) indicating that it has received data. In either case, the message must also list all conflicting requests it detects for this cache line between the time it initiated the request and the time it received the last response. If a conflict has been detected, the node signals either that it was the winner (CNCL) or a loser (READ).

Home must respond to a READ request either by sending the data from memory or (for conflict cases) instructing another node to forward the data. In the case of a conflict, Home must identify all parties to the conflict and provide for the serial transfer of the cache line to each of the requesting nodes. It does this by sending forwarding messages to the winner and all but one of the losing nodes in the conflict, also informing each losing node to await data from a specified node. If there is no conflict, Home responds to a CNCL by acknowledging (ACK) receipt of the message and thus terminating the transaction.

A transaction may be initiated by any of the following requests.

1) ReadShared (RS),
2) Read for Ownership (RFO),

3) Write back (dirty eviction), and

4) Invalidation request (upgrade to ownership[1]).

One of the following responses will be returned from each node:

1) IACK: invalid state acknowledgement (one IACK may include data),

2) SACK: shared stated acknowledgement (no data included),

3) Data&State: data sent along with state (F, E, or M) to transition to, and

4) Conflict: there is already a pending request for the same cache line.

Home is permitted to respond immediately upon receiving a READ or CNCL message from the requestor. It must not wait for all conflicting requests to arrive, because there are cases where some cannot be generated until it has responded.

2.1.1. Transaction Flow for a Read Request

We start by describing a non-exclusive read request. A requesting node broadcasts to all peer nodes a request and waits for their responses. There are four possible cases. For the moment we ignore all conflicts; i.e. overlapping requests for the same cache line. Requests for different cache lines can be arbitrarily overlapped.

i) *The requested line is uncached.* All nodes respond with an IACK (Invalid copy ACKnowlegement). After all IACKs have been received, the node sends a confirming READ request to Home. Home responds with the data, completing the transaction. The node assigns the E state to the cache line.

ii) *The requested line exists in one or more nodes in state S, with at most one node in state F.* (There may not be an F copy, but rather S copies. In this case, the requesting node reads the data from Home and receives it in state F). If the line exists with state F in some node, that node forwards a copy of the line to the requesting cache, changing its own state to S. The

[1] There are many other transaction types such as requests for handling data crossing cache lines, non-coherent transactions, etc. that are present in a production-quality protocol. We limit our discussions to these basic requests for simplicity.

receiving cache acquires the new line in state F and the requestor can effectively use the data. After responses have been received from all nodes except Home, the requesting node sends a cancel message to Home. Upon receiving an acknowledgement message from Home, a DACK message is sent from the requestor to the sending node, completing the transaction (see fig. 2). The DACK message frees the old owner to respond to other requests for the same line. (Conflicts are described more fully in section 2.2.)

The F state allows the current owner, which may be a node other than Home, to determine directly the new owner for the next request to the same cache line. In this manner, multiple nodes can serve as proxies for the Home node (in deciding a new owner) on simple, uncontended requests—the vast majority of requests. Note that the E state and M state (since there can be at most one copy in the system) also conveys the same Home proxy properties to the owner.

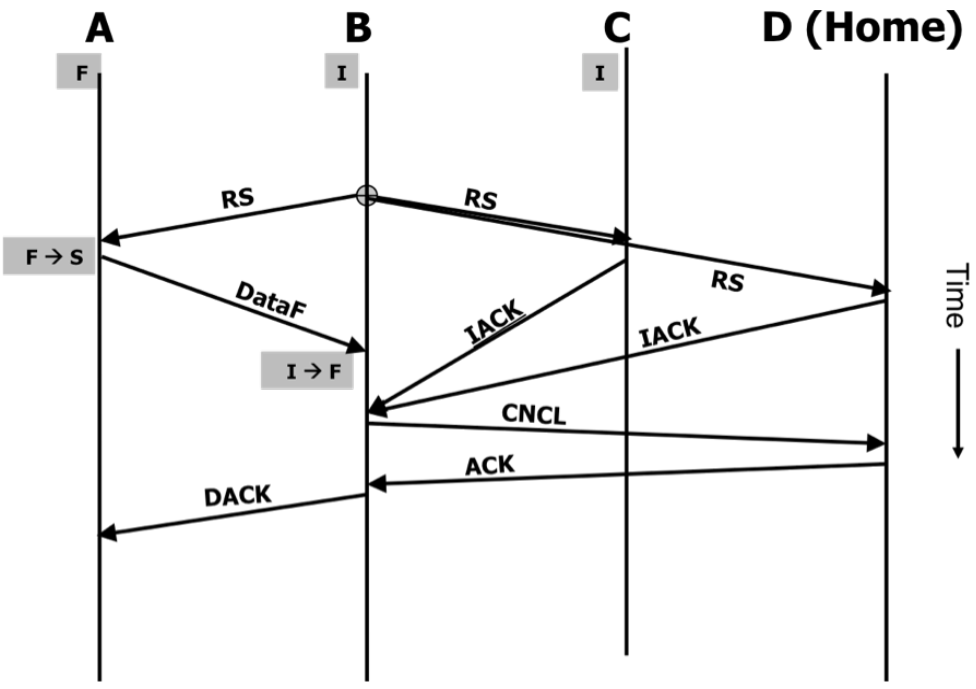


Figure 2. A ReadShared transaction flow with a cached F copy.

iii) *The requested line exists in a single node in E state.* The case is treated exactly the same as if the cache line were in F state instead of E.

iv) *The requested line exists in a single node in M state.* This case is tricky because the line is being requested for read-sharing but the data is modified. In the basic protocol, the line is transmitted to the requesting node as if the request had been a Read-for-Ownership(RFO). A variant allows the data to be shared, allowing the responding node to keep a copy in state S. The requesting node now must "cleanse" the line by writing it back to memory, possibly as a variant of the CNCL request. Alternatively, it may maintain the cache line in a sixth state--Shared-Modified (FM)--deferring the requirement to cleanse the line. The latter choice, while seeming to complicate the protocol, is actually the easiest to implement.

2.1.2. Transaction Flow for a Read For Ownership Request

A read request with permission to modify, known as a Read for Ownership (RFO) is handled similarly to a read request. There are three possible cases.

i) *The requested line is uncached.* This case is handled exactly like a read request.

ii) *The requested line exists in a single node in M or E state.* This case is handled like a read request, except that the delivered data is purged from the sending cache and the state of the received line does not change.

iii) *The requested line exists in one or more nodes in state S, with at most one node in state F.* This case is handled like a read request except that each node invalidates its cache line before responding (they respond with an IACK). This includes the node in the F state, if one exists. Such a node responds to the requesting node with an IACK message containing the data.

Once the requester receives the line, the local processor may commence modifying the data. Where the data is already cached (cases *ii* and *iii*), as for a read request, the requester must send a cancel request to the Home node. In this second phase of completion, the system via the Home node acknowledges that the requester is the new owner (through an ACK message or a forwarding (XFR) message, the latter effectively exposing the modified data to the rest of the system). This is called the Global Observation (G.O.) point. We borrow this 2-phase completion

concept from the x86 front-side bus (FSB) where the G.O. point is when all agents have responded to the snoop request but before the data has arrived at the requester. Because the phases are reversed for MESIF relative to x86, with the data arriving before the global acknowledgement of permission to write, MESIF's G.O. point is a performance advantage over the x86 FSB in cache-to-cache transfers for x86 cores that implement a less strict consistency model than the sequential one.

The delay of the second phase is insignificant with regard to the processor, which may proceed beyond this instruction, though the result must not be exposed system-wide until the G.O. point. Because the processor is able to proceed as soon as the data has arrived, execution need not be delayed by the second phase. That is why MESIF is a 2-hop protocol even though some operations require four sequential hops.

In the case where a line is not cached, the memory controller in the Home node will speculatively fetch the data while all the peer nodes respond to the requestor. It is expected that the READ signal from the requester arrives before the speculative read returns from main memory, yielding the same latency as a conventional 2-hop protocol reading directly from memory.

2.1.3. Transaction Flow for Writeback Line Requests

When a modified cache line is evicted, a Writeback request is issued from the owner node to Home. During this transaction, the ownership is effectively transferred back to Home and while this transaction is pending, the node performing the writeback buffers any snoops to the cache line until Home responds with an acknowledgement. After receiving the acknowledgement from Home, the node can respond with an IACK to any incoming snoops for that cache line (see fig. 3). If the writing node B did not stall incoming snoop probes, then the IACK from B could beat all other messages (including the writeback data) to Home and confuse the Home to send a copy to A from main memory.

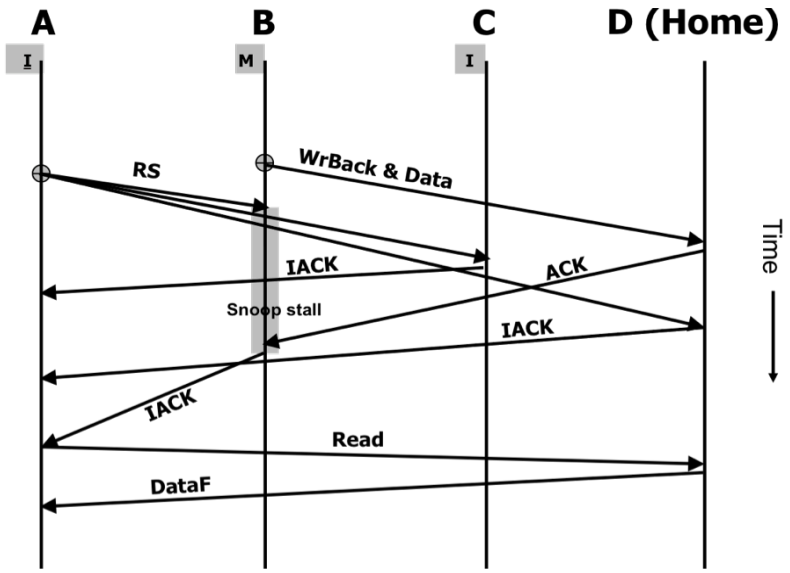


Figure 3. Writeback transaction stalling snoops to the same cache line[2].

2.2. Dealing with Conflicts in MESIF

The concurrency inherent in a point-to-point network means that two or more requests may overlap. If the requests are to different cache lines, the protocol guarantees a global order by establishing the point at which all copies of a cache line (other than the one being modified) have been eliminated. Two requests to the same cache line, however, require some sophistication to assure consistent state and correct global ordering. The Home plays a special role in every transaction not only to supply data that is not cached, but also to make sure that conflicting transactions to the same cache line are handled correctly and efficiently.

If two nodes, A and B, generate requests to the same cache line simultaneously, the order in which they are observed at other nodes may be different. Furthermore, at least one, and possibly both requesting nodes will observe an incoming request to occur after the local request is generated. This is guaranteed because it is impossible for each node to handle a concurrent remote request before it generates its own. Thus for any conflict, at least one of the nodes will detect that it has an outstanding request at the time a conflicting request is received.

[2] This example shows D as a Home node, i.e. it contains both the memory controller owning the cache line plus a caching agent.

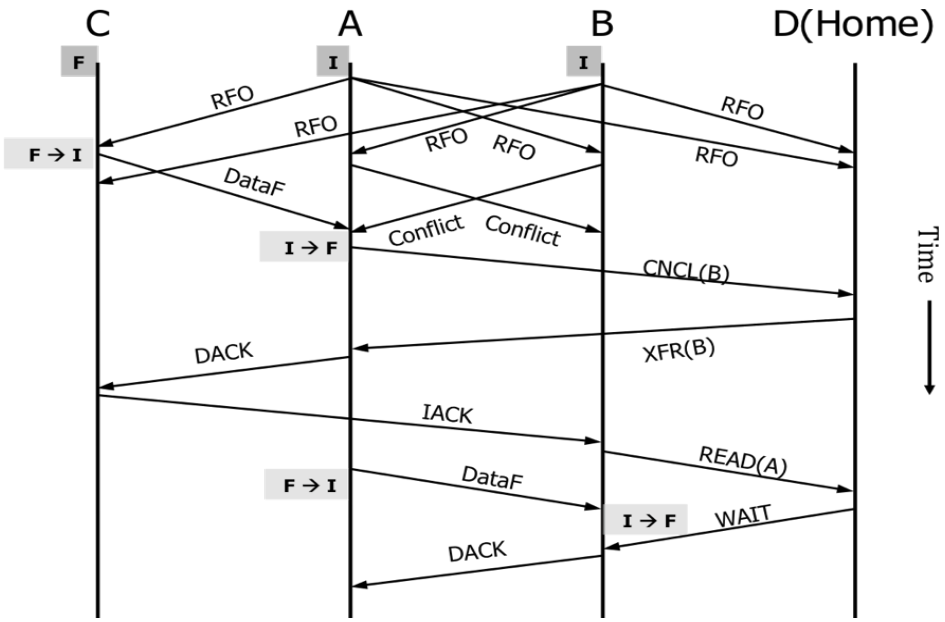


Fig. 4. Conflict resolution[3]. Nodes B and C simultaneously generate Read-for-Ownership requests (RFO) for a cache line that is present only in node A. Node B's request is received first by node A, which responds by sending the line (DataM) to B. Because it has supplied data in a transaction that is not completed, node A is not permitted to respond to the RFO request from node C and must stall all incoming snoops until it receives acknowledgement from B. Both nodes B and C respond to the other's request by indicating a conflict (CNFL). The winner, B, proceeds by canceling its request to Home and indicating the conflict with C (CNCL(C)). Home responds to the cancel request with conflict by instructing node B to forward the line to node C (XFR(C)). After receiving the response from Home, node B acknowledges the data received from A (DACK), and after accessing it momentarily, forwards the data to C (DataM). After receiving the response from A (IACK) but no data, node C now requests data from the Home node, indicating its conflict with B (READ(B)). Home has already instructed node B to forward the data to node C (XFR(C)), and node B may have already delivered the data to C by the time it receives the response from Home (WAIT)[4]. After receiving both signals, node C acknowledges receiving the data from node B (DACK).

Usually a conflict is quickly resolved by the node that supplies the data. The F state simplifies conflict resolution by determining a single node to respond to requests for data that is shared. (If the data is cached in E or M state, a single node plays the same role.) The node resolves the conflict by sending the data to one of the two nodes (presumably, but not necessarily, to the node whose request is received first). This simple resolution is quick, but it does not guarantee that the loser will eventually receive the data. A possible solution is simply for the loser to try again. For

[3] This example shows D simply as Home, i.e. without a caching agent.

[4] The WAIT message is equivalent to an ACK message, but is separately identified for ease of description.

a variety of reasons we rejected that solution, instead requiring that the Home be involved in all conflicts, directing the winner to forward the cache line to the loser. This is necessary even in the case of read-only (shared) data, because the node supplying the data initially (in F state) passes on the F state property to the winning requester. Allowing the node with the F-state to supply the cache line to two nodes might seem more efficient, but it greatly complicates the protocol, which requires no more than one response to each request for data, and because it might result in multiple nodes with a cache line in the F state. Figure 4 gives an example of a simple conflict between two nodes simultaneously requesting ReadShared data.

Because of timing delays, only one of two conflicting nodes may actually detect the conflict. The protocol is robust enough to handle these cases. The Home node is able to recognize and disambiguate all conflicts from the READ or CNCL it receives from each requesting node (indicating all detected conflicts). This is true even though some nodes will not be aware of a conflict and simply ask for data or cancel their request (because it has been satisfied).

Note that the old owner in figure 4 (node A) stalls all pending snoops to the same cache line during the period from when it has sent off the cache line until it receives the DACK from the new owner. This is to allow the winner to notify the Home. When the loser sends its second request to Home, the conflict will be detected and captured in the conflict list that is sent to Home.

2.2.1 Different Kinds of Requests May Conflict

A ReadShared request and an RFO request may occur concurrently. The protocol simply handles them in order, with the transfer assuring that the successful RFO will result in the invalidation of all copies but one. In some cases, a ReadShared request may also result in a modified (exclusive) copy being supplied.

2.2.2. More Than Two Nodes May Conflict Generating Overlapping Requests

In general, a conflict is detected whenever a node detects a conflicting request from another node after it has generated its own request and before it has received its last response. Once a

conflict is detected, the winner receives the cache line and Home is responsible for determining the order for all losers. Because of snoop stalls, additional conflicts may arise once the forwarding mode has begun and can continue indefinitely for a heavily shared line. Such heavily shared data are sometimes critical to an application and handling them efficiently is highly desirable. While the logic to handle this case is somewhat complex for the Home node, the sequence of cache line transfers is nearly optimal.

The protocol handles multiple conflicts in a natural extension to the simple conflict: each node other than the winner is assigned a position in the queue (that is kept in the Home node), and Home sends messages to each of the competing nodes. The messages from Home consist of two instructions:

1) The address of the node to which it should forward the line, and
2) Notification to wait for the cache line that will eventually be forwarded to it.

Obviously Home doesn't send the first instruction to the very last node, and it doesn't send the second instruction to the winner, which receives the data in the normal way.

The conflict resolution protocol outlined in this paper has been formally validated.

2.3. Virtual Channels for Deadlock Avoidance

Since different type messages flow in both directions between any two nodes, care must be taken to avoid deadlock. In MESIF, three virtual channels for each direction are required for deadlock avoidance:

Channel 1: initial requests such as RS, RFO, etc.

Channel 2: snoop responses such as IACK, SACK, and commands to Home, READ, CNCL, etc.

Channel 3: DACK, XFR from Home

The way to reason about virtual channels is that all message types terminating a transaction must be grouped together and must flow unimpeded to the requestor (channel 3). Data can be transferred on channel 3 if all data can be readily accepted at the destination. Responses to

requests must be buffered at the destination and must not be blocked by initial requests (channel 2). Lastly, initial requests can be stalled due to resource limitations and must be kept on a separate channel (channel 1).

3. Performance Evaluation

To evaluate the benefits of this 2-hop protocol, we compare the MESIF protocol to existing 3-hop protocols such as coherent HyperTransport™[6] and Scalability Port™[7]. 3-hop protocols require the requestor to send its request directly to the Home node and then let the Home node perform the snoop broadcast on its behalf. The quickest response happens on a hitM or hitE where the snoop hits an M or E cached copy and the requesting node gets its line in 3-hops when the line is forwarded directly back to the requestor. HitI indicates that the data is uncached, requiring 4 hops to supply data from main memory (actually 4 hops in this case is less important as the latency for fetching from main memory, which is overlapped with snoop probes and responses, dominates the overall transaction time). HitS also requires 4 hops to retrieve data from main memory even though the data is present in another cache. Below, the latency table compares MESIF to a 3-hop protocol. As the reader may notice, for the hitI/hitS cases, MESIF offers no benefits over the 3-hop protocols.

	MESIF	**3-hop**
hitI/hitS	2 hops + max(mem latency, (2 hops + cache latency))	2 hops + max(mem latency, (2 hops + cache latency))
hitE/hitM	2 hops + cache latency	3 hops + cache latency
hitF	2 hops + cache latency	N/A

To gauge the impact of a 2-hop protocol in a point-to-point interconnect versus a 3-hop protocol, we utilized a sophisticated Mean Value Analysis tool developed in-house by a

performance validation group for evaluating MESIF and QPI. The tool is fundamentally based on the latency formulas shown in the above table.

Specifically, the tool is used to project TPC-C performance as that is the most used, and some would say, the most indicative of server performance. The tool projects performance for various MP configurations where multiple parameters such as core speed, number of threads per core, number of cores per node, various cache sizes, speed of interconnect components, memory speed and bandwidth, etc. are entered. Some of the parameters such as cache miss rates, cache miss types, and I/O activity are captured from actual systems running the OLTP workload. The algorithms employed in the tool as well as many measured and derived parameters are trade secrets and cannot be divulged in this paper. Needless to say, the tool has been validated extensively with actual small MP systems (4 sockets). Projections of absolute performance fall within ±5% of the actual system. When the tool is used to project relative performance, e.g. modifying one or two components of the system, the error bar is much less: ±1%.

Mean Value Analysis works for applications such as OLTP because it is a throughput-oriented workload where the average behavior over a long run of the application is indicative of the total behavior of the application. This also applies to applications such as SpecWeb, etc.

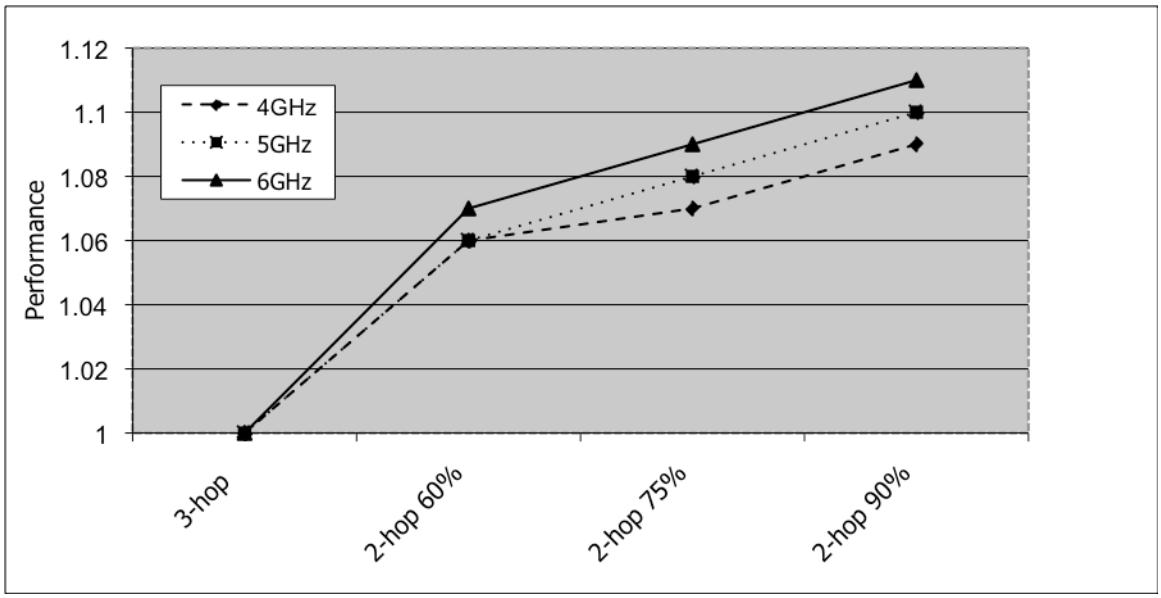
In the following table, we list the important parameters used in the projections. The cache specified is the largest cache in a node and is shared by the multiple cores in the node. It is listed in the table because it has the most impact on the performance of 2-hop versus 3-hop protocols. Other caches closer to the core have less impact.

Parameters	**Value**
Core frequency	4GHz, 5GHz and 6GHz
Threads per core	2
Cores per node	2
Number of sockets	4
Cache line size	64 bytes

Cache size for node	12MB, 12-way associative
Cache hit latency	~40ns (serial tag data access)
Cache miss that hit in another node's cache	60%, 75%, 90%
Aggregate memory size	32GB (8GB per socket)
Main memory latency	~100ns
Main memory bandwidth per socket	6.4GB/s
Interconnect	Fully connected, 2 uni-directional links per port
Interconnect bandwidth	6.4GB/s per link, 12.8GB/s aggregate per port
1-hop Interconnect latency	20ns

In the row labeled "Cache miss that hit in another node's cache", we vary the percentage of requests leaving a node that can be satisfied by another node's cache. Cache-to-cache transfers are inherently faster than requests satisfied by main memory and are another major component in determining the goodness of a 2-hop protocol.

The performance data of varying the core frequency and varying the percentage of last level cache misses that hit in a last level cache in another socket is shown below.



Our analysis indicates a clear advantage in all cases for the MESIF over the 3-hop protocol. For a 4GHz clock, with 4 nodes (four threads—two processors--per node), performance will be

enhanced by 6 to 11%, depending on cache-to-cache to main memory-to-cache ratio. Obviously, as more requests can be satisfied from another node's cache, the more benefit of a 2-hop protocol. For faster cores, the improvement is slightly larger because the impact of going off-node is larger; any improvement there will have a commensurately larger impact on performance.

4. MESIF and QPI

The Intel® Quick-Path Interconnect™ (QPI) was recently introduced to replace the Front-Side Bus (FSB) for Desktop, Xeon and Itanium platforms, first appearing in the Core i7 desktop processor and the X58 chipset. Like, MESIF from which it is derived, QPI is a source snoop protocol. It differs from MESIF primarily in that responses to a broadcast request are sent to the Home node rather than the requesting node. When data is required in a response, it is sent directly, in a separate packet, to the requesting node.

This approach was considered for MESIF, but rejected because it imposed greater responsibility on the Home node in resolving conflicts and required ordering of messages delivered to the Home node. The ordering of messages within a given channel imposed a dependence on the routing fabric which MESIF strived to avoid. In current products, the QPI protocol requires strict ordering of the HOM class of messages pertaining to a single cache line – those sent to the Home requesting access to that cache line and those sent to the Home in response to snoop requests pertaining to the cache line. In this section we give examples showing how the two protocols differ.

We first consider the simple case of a RdShr(RS) to an uncached line. Figure 5(a) shows the paths and sequences for a three-processor system using MESIF when node C attempts to read from a memory location that is uncached. Figure 5(b) shows the situation using QPI. MESIF requires one additional message, but the latency will be the same assuming that all messages take

the same time and that the Home node sees a memory latency greater than or equal to two message times.

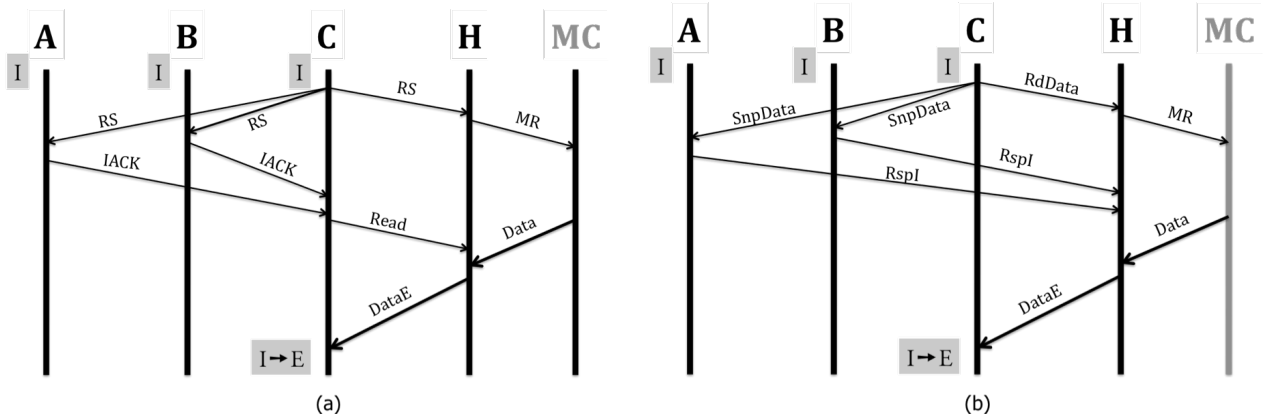


Fig. 5. Read to an uncached line in a three-node system for MESIF (a) and QPI (b). In QPI responses are sent directly to the Home node, resulting in one fewer message. *(QPI source: Maddox, Singh & Safranek)*

Next we compare the protocols when node C attempts to read for ownership a cache line already present in another cache in modified state (figure 6). In both cases, node B may use the data as soon as it is received from C (even retiring the instruction), though it may not expose the modified cache line to the rest of the system until it received a completion signal (ACK) from the Home node. Again assuming that all messages take the same time, the completion signal is delayed by one message time in arriving at B.

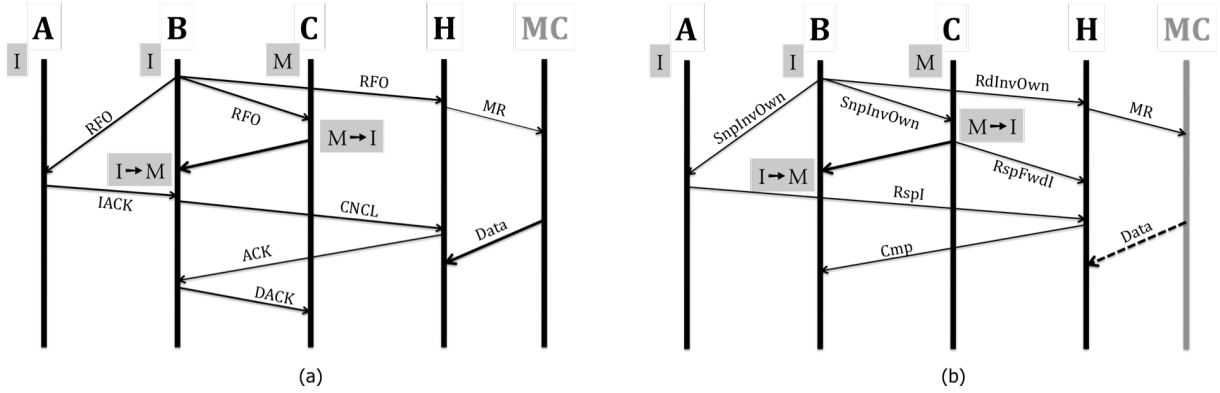


Fig. 6. RFO to a modified cache line in a three-node system for MESIF (a) and QPI (b). In QPI, node C sends two responses, data to the requester and a forwarding notification to the Home node. MESIF still requires one additional message, however, because node B must acknowledge receipt of the data from node C, which must stall snoop requests to the same address until acknowledged. *(QPI source: Maddox, Singh & Safranek)*

Finally, we compare the protocols in the context of contention. In MESIF, one or both of the conflicting nodes will detect the conflict (only node B in the example), so both will know after the response, and including the conflict information in the confirming request for the data to the Home node. Assuming each initial request has arrived when the confirming request arrives, the first confirming request receives the data as soon as it is available from memory. Even when messages are delayed, as in the example, the first confirming request (from A) to arrive at the Home node (assuming the initial request preceded it) wins, resulting in delivery of the data as soon as it is available from memory. On arrival of the second confirming request (from B), a forwarding directive is sent to A. Thus the data is delivered directly to node A. Assuming that the winning confirming request arrives at the Home node before memory supplies the data, the system delivers the cache line optimally, first sending it to node A, with instructions to modify it once and forward it to node B.

QPI performs worse in the conflict case, requiring more messages but also a much longer sequence of messages. To avoid deadlock in the protocol, an extra handshake is required at the end, a total of 9 messages being required (7 after data is supplied by the memory) before the G.O. point is reached for the losing node.

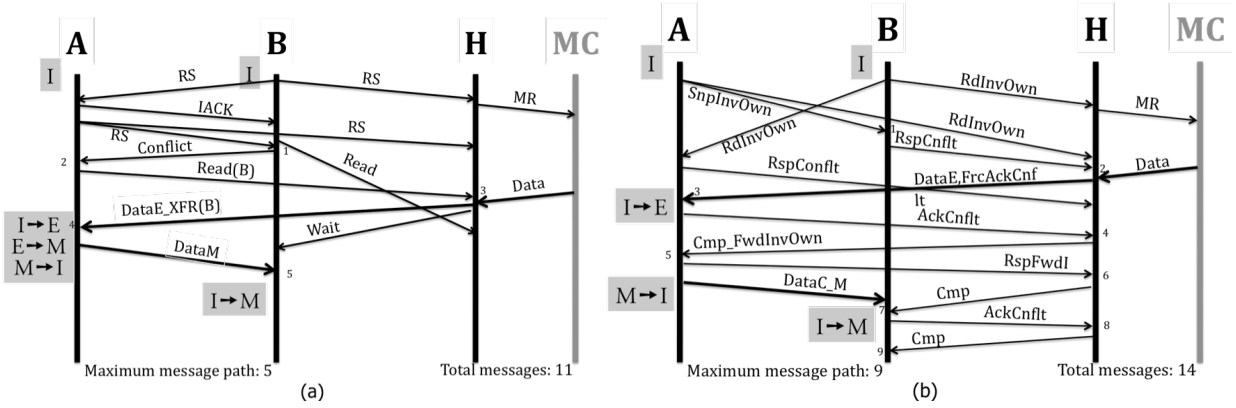


Fig. 7. Conflicting RFOs to an uncached line in a two-node system for MESIF (a) and QPI (b). *(QPI source: Maddox, Singh & Safranek)*

This comparison shows that 2-hop protocols gain the benefits of lower latencies for the more common, non-conflicting requests. QPI opted to forgo the conflict list accumulation at the

requesting node at the expense of requiring an in-order HOM channel and a conflict resolution flow that requires extra messages. Conflicts are rare, so these tradeoffs can be justified.

5. Related Work

Destination Set Prediction [9] and Token Coherence [10] are contemporary work of MESIF, independently developed. This work assumes an unordered interconnect and separates the protocol into a performance protocol, which is fast but does not always succeed, and a correctness substrate, which guarantees correct ordering of operations. Common operations can frequently be retrieved in two hops. This idea is compatible with MESIF, which could be extended to seek a shared or modified copy of a cache line by broadcasting to only some nodes, falling back on MESIF if it failed.

Timestamp snooping[11] is an alternate approach to creating a global ordering of events within a point-to-point network, by assigning a logical time to each operation and processing them in the logical order. Recent work has continued that approach [12].

6. Summary

We have described a novel protocol for point-to-point interconnects that garners the latency benefits of a bus-based protocol in terms of number of hops and retains the bandwidth advantages of a point-to-point fabric. We have shown the clear performance advantage of our 2-hop protocol versus known point-to-point protocols that require a minimum of three hops for a memory transaction. Indeed, we are unable to identify a single case where a 3-hop protocol is superior.

Within Intel Corporation, this work—completed in 2001—spawned numerous variants of the 2-hop protocol that have been formally validated, and analyzed on performance and implementability, including the QPI protocol now available in Intel products.

7. References

[1] R. Maddox, G. Singh, R. Safranek, "Weaving high performance multiprocessor fabric: architectural insights to the Intel® QuickPath Interconnect," Intel Press, 2009.

[2] A. Charlesworth, A. Phelps, R. Williams, and G. Gilbert. "Gigaplane-XB: extending the Ultra Enterprise family," *Proceedings of the Symposium on High Performance Interconnects V*, pages 97–112, August 1997.

[3] K.J. Getzlaff, B. Leppla, H.-W Tast, U. Wille, "Logical bus structure including plural physical busses for a multiprocessor system with a multi-level cache memory structure, IBM (Armonk, NY), US patent #5889969, 1997.

[4] E. Hagersten, A. Landin, and S. Haridi. "DDM - A cache-only memory architecture," *IEEE Computer*, September 1992, pp. 44-54.

[5] Institute of Electrical and Electronics Engineers, New York, NY, *IEEE Standard for the Scalable Coherent Interface (SCI),* August 1993. ANSI/IEEE Std. 1596-1992.

[6] "HyperTransport™ Technology I/O link," White paper, July 2001. Available at http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/AMD_HyperTransport_Technology-Based_System_Architecture.pdf

[7] M. Azimi, F. Briggs, M. Cekleov, M. Khare, A. Kumar, L.P. Looi, "Scalability port: a coherent interface for shared memory multiprocessors," *Proc. 10th Symposium on High Performance Interconnects HOT Interconnects (HotI'02)*, pp. 65-70, August 2002.

[8] P. Sweazey and A.J. Smith, "A class of compatible cache consistency protocols and their support by the IEEE Futurebus," Proceedings of the *13th Annual International Symposium on Computer Architecture*, pp. 414–423, June 1986.

[9] M.M. Martin, P.J. Harper, D.J. Sorin, M.D. Hill, and D.A. Wood, "Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors," Proceedings of the *30th Annual International Symposium on Computer Architecture*, Jun 9-11 2003, pp. 206-217.

[10] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token Coherence: Decoupling Performance and Correctness," Proceedings of the *30th Annual International Symposium on Computer Architecture*, San Diego, California, pp. 182-193, June 2003.

[11] M.M. Martin, D.J. Sorin, A. Ailamaki, A.R. Alameldeen, R.M. Dickson, C.J. Mauer, K.E. Moore, M. Plakal, M.D. Hill, and D.A. Wood, "Timestamp snooping: an approach for extending SMPs," Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 25-36, 2000.

[12] L.-S. Peh, N. Agarwal, N. Jha. "In-network snoop ordering (INSO): snoopy coherence on unordered interconnects," *International Symposium on High Performance Computer Architecture (HPCA)*, February, 2009, pp. 67-78.