decode transactions, may handle conflicts, may interface with the snoop filter 204, and may process transactions. In one embodiment, the protocol logic 202 may comprise distributed protocol logic (DPL) 210 and centralized protocol logic (CPL) 212. In one embodiment, each port 200 has an associated DPL 210 to locally implement portions of the protocol logic 202 for the respective port 200. In particular, the DPL 210 may comprise decode logic to decode incoming transactions and may comprise one or more buffers or queues to store data and/or other information associated with incoming and outgoing transactions while being processed by the protocol logic 202 and/or awaiting responses from cache nodes 102, 104.")

- [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

In addition, the claimed interconnection controller comprises protocol engines for processing transactions in accordance with a cache coherence protocol. *See*, *e.g.*, '206 patent claim 1.4. A cache coherence protocol facilitates cache coherency, thus the claimed interconnection controller is necessarily operable to facilitate cache coherency in the computer system.

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and Exhibits E-1 – E-14 to include an interconnection controller operable to facilitate cache coherency across the computer system, at least under Memory Integrity's apparent infringement

theories. *See*, *e.g.*, Exhibits D-1 – D-14, claim 15.1. For example, it would have been obvious for an interconnection controller that processes coherence transactions to process those transactions so as to facilitate cache coherency across the computer system as described above with respect to the "cache coherence controller."

8. *"Cache Coherence Controller" "Constructed to Act As An Aggregate Remote Cache"*

Some of the Asserted Claims are directed to a "cache coherence controller" "constructed to act as an aggregate remote cache." For example, claim 18.1 of the '409 patent recites "the cache coherence controller is constructed to act as an aggregate remote cache." *See also*, *e.g.*, '409 patent claim 47.1; and '636 patent claim 29.1. At least under Memory Integrity's apparent infringement theories, cache coherence controllers "constructed to act as an aggregate remote cache" were well-known in the art before the priority dates of the Asserted Patents. *See*, *e.g.*, Exhibits A-1 – A-9, claims 18.1 and 47.1; and Exhibits B-1 – B-19, claim 29.1. The following discussion further shows that, at least under Memory Integrity's apparent infringement theory, it was well known and conventional to implement cache coherence controllers "constructed to act as an aggregate remote cache" in multiprocessor systems.

At least under Memory Integrity's apparent infringement theories, there are many examples of prior art references that disclose implementing cache coherence controllers "constructed to act as an aggregate remote cache." Examples of prior art references that disclose and further demonstrate that such was well known include:

- "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor." Lenoski (1990): *See*, *e.g.*, p. 1, para. 2 ("We are currently building a prototype of a scalable shared memory multiprocessor. The system provides high processor performance and scalability though the use of coherent caches and a directory-based coherence protocol. The high-level organization of the prototype, called DASH (Directory Architecture for SHared memory) [17]. is shown in Figure 1. The architecture consists of a number of processing nodes connected through a high-bandwidth low-latency interconnection network. The physical memory in the

machine is distributed among the nodes of the multiprocessor, with all memory accessible to each node. Each processing node, or cluster, consists of a small number of high-performance processors with their individual caches, a portion of the shared-memory, a common cache for pending remote accesses, and a directory controller interfacing the cluster to the network. A bus-based snoopy scheme is used to keep caches coherent within a cluster, while inter-node cache consistency is maintained using a distributed directory-based coherence protocol.")

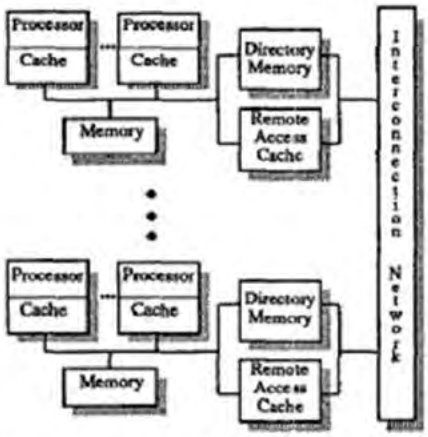

Figure 1: General architecture of DASH.

Lenoski (1990), Figure 1

- Lenoski (1990): *See, e.g.*, p. 1, para. 4 ("In DASH, each processing node has a directory memory corresponding to its portion of the shared physical memory. For each memory block, the directory memory stores the identities of all remote nodes caching that block. Using the directory memory, a node writing a location can send point-to-point invalidation or update messages to those processors that are actually caching that block.")

- Lenoski (1990): *See, e.g.*, p. 3, para. 5 ("The directory controller (DC) contains the directory memory corresponding to the portion of main memory present within the cluster. It also initiates out-bound network requests and replies. The pseudo-CPU (PCPU) is responsible for buffering incoming requests and issuing such requests on the cluster bus. It mimics a CPU on this bus on behalf of remote processors except that responses from the bus are sent out by the directory controller. The reply controller (RC) tracks outstanding requests made by the local processors and receives and buffers the corresponding replies from remote clusters. It acts as memory when the local processors are allowed to retry their remote requests. The network interface and the local portion of the network itself reside on the directory card. The interconnection network consists of a pair of meshes. One mesh is dedicated to the request messages while the other handles replies. These meshes utilize wormhole routing [9] to minimize latency. Finally, the board contains hardware monitoring logic and miscellaneous control and status registers. The monitoring logic samples a variety of directory board and bus events from which usage and performance statistics can be derived.")

- Lenoski (1990): *See, e.g.*, p. 4.5, para. 4 ("In the protocol, invalidation acknowledges are sent to the local cluster that initiated the memory request. An alternative would be for the home cluster to gather the acknowledges, and, when all have been received, send a message to the requesting cluster indicating that the request has been completed. We chose the former because it reduces the waiting time for completion of a subsequent fence operation by the requesting cluster and reduces the potential of a hot spot developing at the memory.")

- "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor." Lenoski (1992): *See, e.g.*, p. 150 ("A DASH system consists of a number of modified 4D/240 systems that have been supplemented with a directory controller board. This directory controller board is responsible for maintaining the cache coherence across the nodes and serving as the interface to the interconnection network.")

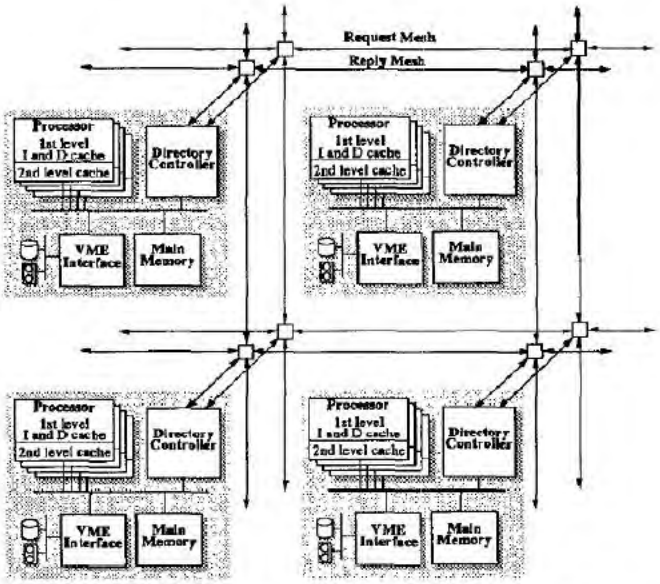

Figure 2: Block diagram of sample 2 x 2 DASH system.

Lenoski (1992), Figure 2

- U.S. Patent No. 6,055,610 to Smith: *See, e.g.*, 11:44-55 ("A flow chart of the basic method M1 of handling a data request is flow charted in FIG. 3. At step S1, processor P11 issues a read request of data stored in main memory MM0. At step S2, caches C10-C13 of requester cell MC1 are examined to determine if the request can be met locally. First, associated cache C11 is checked. A hit allows the request to be met locally. A miss refers the request to the requestor's coherency controller CC1. Coherency controller CC1 initiates a local snoop while referring the request to owner cell MC0. If the snoop results in a hit, the request can be met locally. If the data is held privately by another local processor, e.g., processor P12, coherency controller requests that the data be made public so that the request can be met. Only if the local snoop misses is involvement of the owner cell MC0 required.")

- Smith: *See*, *e.g.*, 11:56-63 ("At step S3, coherency controller CC0 of owner cell MC0 initiates a local snoop of its caches, accesses fast directory FD0, and initiates access of main memory MM0. Coherency controller CC0 determines whether or not the fast-directory data calls for a recall and whether the directory cache data is consistent with the local snoop results. If the directory data is consistent with the snoop results and if a recall is indicated, it is initiated at step S4.")

- Smith: *See*, *e.g.*, 12:3-8 ("Once the recall process is complete, the requested data is transferred to the requester cell MC1, coherency controller CC1, cache C11, and processor P11, at step S6. State information in cache C11, fast directory FD0, and the coherency directory of main memory MM0 is updated as necessary. This completes method M1.")

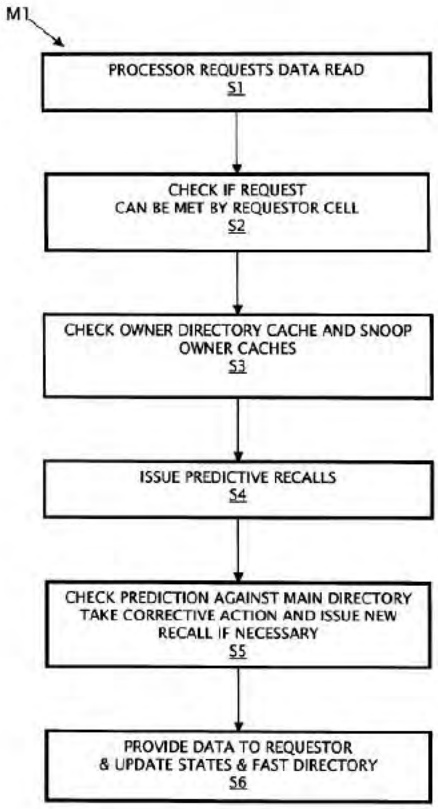

Figure 3

Smith, Figure 3

- U.S. Patent No, 6,085,295 to Ekanadham: *See*, *e.g.*, 2:25-33 ("In a node where a remote line is brought into the cache of a processor, but not into the node's memory, the adapter acts as a proxy memory representing the remote memory that the line is mapped onto. More specifically, when a memory command is issued from a local processor to a remote memory, the memory command is directed to the adapter which is responsible for insuring that the command is executed at that remote memory.")

- Ekanadham: *See, e.g.*, 3:37-45 ("The preferred embodiment of our system that is based on a network of switch-based SMP nodes with an adapter attached to each node. FIG. 1 illustrates a high-level diagram of such a multiprocessing system. Each node has a plurality of processors P1, P2, . . . , PN interconnected to each other by a switch (SW). The switch also interconnects the memory modules M1, M2, . . . , MN and adapters A. The nodes in turn, are connected to each other through a network as shown.")

- Ekanadham: *See, e.g.*, 3:49-56 ("The adapter connects to the switch and plays the role of either a memory or a processor. The behavior of the adapter is different for different memory lines. When a line is homed at the local memory of the node, the adapter behaves as a proxy processor for that line. When a line is homed at the memory of a remote node, the adapter behaves as a proxy memory for that line. These roles are illustrated in FIGS. 3A-3C and are elaborated further below.")

- Ekanadham: *See, e.g.*, 4:7-24 ("In a node in which a line is homed in the local memory, the adapter plays the role of a proxy processor representing the accesses to the line made by the processors in other nodes of the system. In this role, the adapter maintains a state for the line and the list of all nodes sharing that line. The state can be I (indicating that no other node has this line), E (indicating that some other node has exclusive copy of this line) or S (indicating that this line is shared by this and other nodes). As a proxy processor, the adapter receives requests from other adapters and performs the reads and writes in this node on their behalf. Whenever a local processor requires exclusive access to the line while it is in shared state, it communicates with other adapters and invalidates the line in all other nodes. When another node requests for exclusive copy of the line. The adapter only invalidates the copies in all other nodes, but also requests the local memory to grant the exclusive access. The memory controller treats the adapter as another processor.")

- Ekanadham: *See, e.g.*, 4:26-41 ("In a node in which a line is homed at a remote memory, the adapter acts as a proxy memory. It captures all the transactions for the corresponding address and runs the memory protocol. In this role, the adapter maintains a state for the line and the list of local caches sharing the line. The state can be I (indicating that no local cache this line), E (indicating that some local cache has exclusive copy of this line) or S (indicating that this line is shared by this and other nodes). As a proxy memory, the adapter responds to all requests to the line and obtains the contents of the line from the remote node (where that line is backed by memory) and supplies the contents to the local caches. It performs the usual coherence control operations in the node and coordinates with other adapters. In order to maintain global coherence, it may have to issue some bus transactions as a master, as illustrated later.")

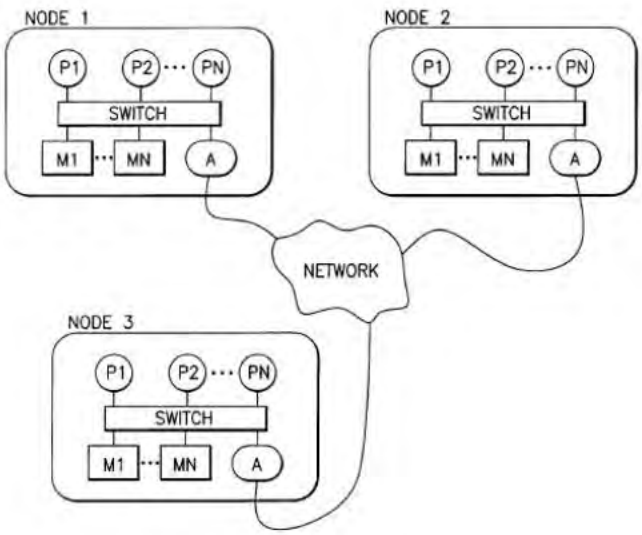

FIG.1
PRIOR ART

Ekanadham, Figure 1

- U.S. Patent No, 6,141,692 to Loewenstein: *See*, *e.g.*, 6:29-33 ("Requests for data and responses to those requests are exchanged between nodes by the respective HA, SA, and RA of each global interface (i.e., 415, 425, 435, and 445) in the form of data/control packets, thereby enabling each node to keep track of the status of all data cached therein.")

- Loewenstein: *See*, *e.g.*, 11:1-9 ("When the system is operating in NUMA mode, a typical read request (e.g., a Read-- To-- Share request) by processor 411a of node 410 occurs in the following manner. To initiate the request, processor 411a presents a virtual address (VA) to MMU 412a, which converts the VA into a GA and presents the GA to cache 413a. If there is a valid copy of the data line of interest in cache 413a (e.g., a shared or owned copy), then cache 413a provides the data to processor 411a via MMU 412a, thereby completing the read request.")

- Loewenstein: *See*, *e.g.*, 11:10-15 ("However, if cache 413a does not have a valid copy, then cache 413a presents the GA to the local interconnect 419 of its associated node. If the GA is not part of the node 410's local address space (i.e., node 410 is not the home node for the requested address), then the request is forwarded to the appropriate home node (i.e., node 420).")

- Loewenstein: *See*, *e.g.*, 11:40-50 ("If the home node is determined to have a valid copy of the requested data line, then the home node provides the data to the requesting node. In the case where the requesting node is also the home node, only an internal data transfer is required. Alternatively, where the home node is not the requesting node, then the global interface of the home node (global interface 425 in the above example) responds by retrieving the requested data line from the main memory 424 or from a

cache line which is owned by a processor within node 420, and sends the data line to the global interface 415 of the requesting node 410 via global interconnect 450.")

- Loewenstein: *See*, *e.g.*, 11:60-64 ("Upon receiving the data line, global interface 415 forwards the data line to cache 413a, which provides the data to the requesting processor 411a.")

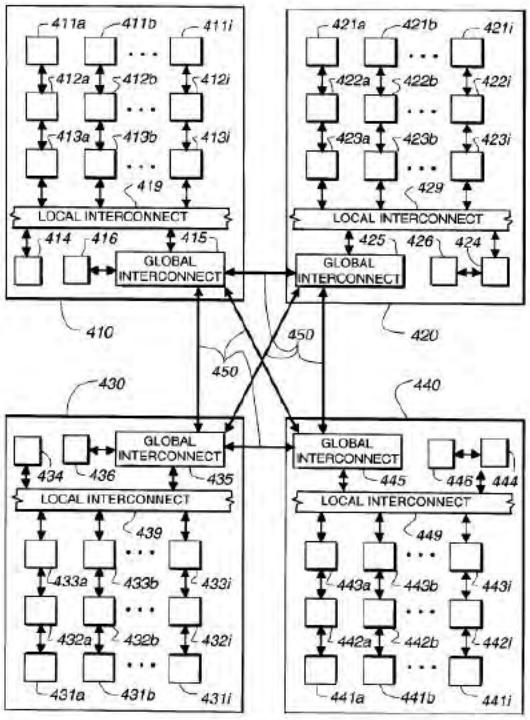

Fig. 4

Loewenstein, Figure 4

- U.S. Patent No, 6,751,721 to Webb: *See*, *e.g.*, Abstract ("A directory-based multiprocessor cache control scheme for distributing invalidate messages to change the state of shared data in a computer system. The plurality of processors are grouped into a plurality of clusters. A directory controller tracks copies of shared data sent to processors in the clusters. Upon receiving an exclusive request from a processor requesting permission to modify a shared copy of the data, the directory controller generates invalidate messages requesting that other processors sharing the same data invalidate that data. These invalidate messages are sent via a point-to-point transmission only to master processors in clusters actually containing a shared copy of the data. Upon receiving the invalidate message, the master processors broadcast the invalidate message in an ordered fan-in/fan-out process to each processor in the cluster. All processors within the cluster invalidate a local copy of the shared data if it exists and once the master processor receives acknowledgements from all processors in the cluster, the master processor sends an invalidate acknowledgment message to the processor that originally requested the exclusive rights to the shared data. The cache coherency is scalable and may be implemented using the hybrid

point-to-point/broadcast scheme or a conventional point-to-point only directory-based invalidate scheme.")

As illustrated by the prior art references above, it was well known before the priority dates of the Asserted Patents to implement a "cache coherence controller" "constructed to act as an aggregate remote cache" in multiprocessor systems, at least under Memory Integrity's apparent infringement theories. Indeed, a person of ordinary skill would have been motivated to implement a "cache coherence controller" "constructed to act as an aggregate remote cache" in a multiprocessor system as described below:

- Ekanadham: *See*, *e.g.*, 1:23-35 ("Technology considerations limit the size of an SMP node to a small number of processors. A method for building a shared-memory multiprocessor with a larger number of processors is to connect a number of SMP nodes with a network, and provide an adapter to extend the SMP's memory across the SMP nodes (see FIG. 1). Existing adapter designs plug into the memory bus of bus-based SMP nodes and collectively provide shared memory across the system, so that any processor in any node can access any location in any memory module in the system. Resources within a node are termed local and resources on other nodes are termed remote.")

- Ekanadham: *See*, *e.g.*, 2:37-41 ("By appearing as either a local processor or a local memory, the adapter uses the local SMP coherence protocol within a node to accomplish the above tasks, without any changes to the memory controllers.")

- Loewenstein: *See*, *e.g.*, 5:1-8 ("Since global interface 115 is also responsible for maintaining global cache coherency, global interface 115 includes a hardware and/or software implemented cache-coherency mechanism for maintaining coherency between the respective caches and main memories of nodes 110, 120, . . . 180. Cache coherency is essential in order for the system 100 to properly execute shared-memory programs correctly.")

Accordingly, it would have been obvious to implement a "cache coherence controller" "constructed to act as an aggregate remote cache" in a multiprocessor system having multiple clusters of processors while maintaining coherency with a reasonable expectation of success. Furthermore, it would have been obvious to implement a "cache coherence controller" "constructed to act as an aggregate remote cache" because such a modification would simply be the use of a known technique (*e.g.*, a cache coherence controller "constructed to act as an aggregate

remote cache") to improve similar devices (*e.g.*, multiprocessor systems) in the same way (*e.g.*, improve performance and scalability while maintaining coherency).

9. *"Cache Coherence Controller" "Constructed to Act As A Probing Agent Pair"*

Some of the Asserted Claims are directed to a "cache coherence controller" "constructed to act as a probing agent pair." For example, claim 19.1 of the '409 patent recites "the cache coherence controller is constructed to act as a probing agent pair." *See also*, *e.g.*, '409 patent claim 48.1; and '636 patent claim 30.1. At least under Memory Integrity's apparent infringement theories, cache coherence controllers "constructed to act as a probing agent pair" were well-known in the art before the priority dates of the Asserted Patents. *See*, *e.g.*, Exhibits A-1 – A-9, claims 19.1 and 48.1; and Exhibits B-1 – B-19, claim 30.1. The following discussion further shows that, at least under Memory Integrity's apparent infringement theory, it was well known and conventional to implement cache coherence controllers "constructed to act as a probing agent pair" in multiprocessor systems.

At least under Memory Integrity's apparent infringement theories, there are many examples of prior art references that disclose implementing cache coherence controllers "constructed to act as a probing agent pair." Examples of prior art references that disclose and further demonstrate that such was well known include:

- "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor." Lenoski (1990): *See*, *e.g.*, p. 1, para. 2 ("We are currently building a prototype of a scalable shared memory multiprocessor. The system provides high processor performance and scalability though the use of coherent caches and a directory-based coherence protocol. The high-level organization of the prototype, called DASH (Directory Architecture for SHared memory) [17]. is shown in Figure 1. The architecture consists of a number of processing nodes connected through a high-bandwidth low-latency interconnection network. The physical memory in the machine is distributed among the nodes of the multiprocessor, with all memory accessible to each node. Each processing node, or cluster, consists of a small number of high-performance processors with their individual caches, a portion of the shared-memory, a common cache for pending remote accesses, and a directory

controller interfacing the cluster to the network. A bus-based snoopy scheme is used to keep caches coherent within a cluster, while inter-node cache consistency is maintained using a distributed directory-based coherence protocol.")

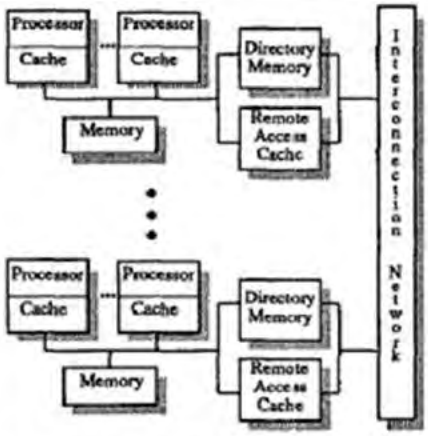

Figure 1: General architecture of DASH.

Lenoski (1990), Figure 1

- Lenoski (1990): *See, e.g.*, p. 1, para. 4 ("In DASH, each processing node has a directory memory corresponding to its portion of the shared physical memory. For each memory block, the directory memory stores the identities of all remote nodes caching that block. Using the directory memory, a node writing a location can send point-to-point invalidation or update messages to those processors that are actually caching that block.")

- Lenoski (1990): *See, e.g.*, p. 3, para. 5 ("The directory controller (DC) contains the directory memory corresponding to the portion of main memory present within the cluster. It also initiates out-bound network requests and replies. The pseudo-CPU (PCPU) is responsible for buffering incoming requests and issuing such requests on the cluster bus. It mimics a CPU on this bus on behalf of remote processors except that responses from the bus are sent out by the directory controller. The reply controller (RC) tracks outstanding requests made by the local processors and receives and buffers the corresponding replies from remote clusters. It acts as memory when the local processors are allowed to retry their remote requests. The network interface and the local portion of the network itself reside on the directory card. The interconnection network consists of a pair of meshes. One mesh is dedicated to the request messages while the other handles replies. These meshes utilize wormhole routing [9] to minimize latency. Finally, the board contains hardware monitoring logic and miscellaneous control and status registers. The monitoring logic samples a variety of directory board and bus events from which usage and performance statistics can be derived.")

- Lenoski (1990): *See, e.g.*, p. 4.5, para. 4 ("In the protocol, invalidation acknowledges are sent to the local cluster that initiated the memory request. An alternative would be for the home cluster to gather the acknowledges, and, when all have been received,

send a message to the requesting cluster indicating that the request has been completed. We chose the former because it reduces the waiting time for completion of a subsequent fence operation by the requesting cluster and reduces the potential of a hot spot developing at the memory.")

- "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor." Lenoski (1992): *See, e.g.*, p. 150 ("A DASH system consists of a number of modified 4D/240 systems that have been supplemented with a directory controller board. This directory controller board is responsible for maintaining the cache coherence across the nodes and serving as the interface to the interconnection network.")

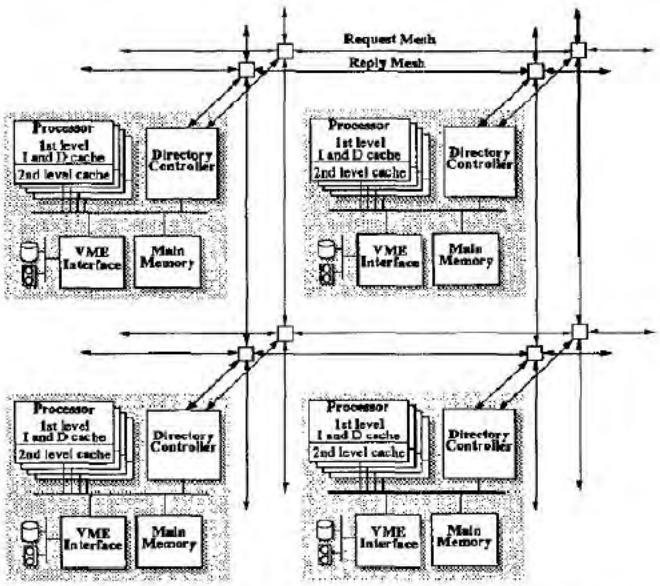

Figure 2: Block diagram of sample 2 x 2 DASH system.

Lenoski (1992), Figure 2

- U.S. Patent No. 6,055,610 to Smith: *See, e.g.*, 11:44-55 ("A flow chart of the basic method M1 of handling a data request is flow charted in FIG. 3. At step S1, processor P11 issues a read request of data stored in main memory MM0. At step S2, caches C10-C13 of requester cell MC1 are examined to determine if the request can be met locally. First, associated cache C11 is checked. A hit allows the request to be met locally. A miss refers the request to the requestor's coherency controller CC1. Coherency controller CC1 initiates a local snoop while referring the request to owner cell MC0. If the snoop results in a hit, the request can be met locally. If the data is held privately by another local processor, e.g., processor P12, coherency controller requests that the data be made public so that the request can be met. Only if the local snoop misses is involvement of the owner cell MC0 required.")

- Smith: *See, e.g.*, 11:56-63 ("At step S3, coherency controller CC0 of owner cell MC0 initiates a local snoop of its caches, accesses fast directory FD0, and initiates access of main memory MM0. Coherency controller CC0 determines whether or not the

fast-directory data calls for a recall and whether the directory cache data is consistent with the local snoop results. If the directory data is consistent with the snoop results and if a recall is indicated, it is initiated at step S4.")

- Smith: *See, e.g.*, 12:3-8 ("Once the recall process is complete, the requested data is transferred to the requester cell MC1, coherency controller CC1, cache C11, and processor P11, at step S6. State information in cache C11, fast directory FD0, and the coherency directory of main memory MM0 is updated as necessary. This completes method M1.")

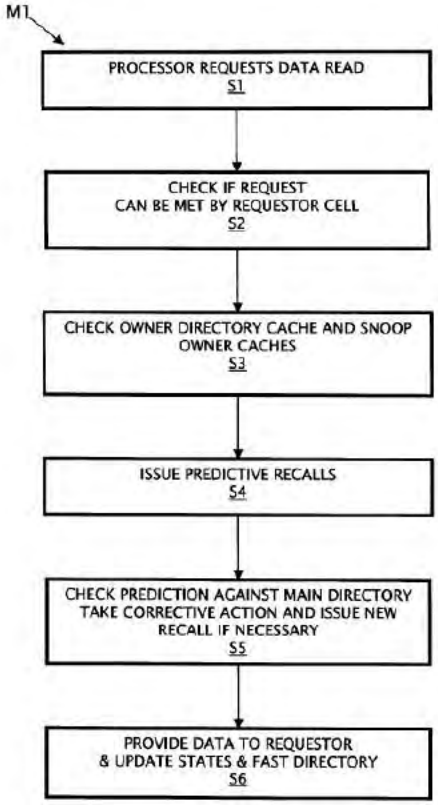

Figure 3

Smith, Figure 3

- U.S. Patent No, 6,085,295 to Ekanadham: *See, e.g.*, 2:25-33 ("In a node where a remote line is brought into the cache of a processor, but not into the node's memory, the adapter acts as a proxy memory representing the remote memory that the line is mapped onto. More specifically, when a memory command is issued from a local processor to a remote memory, the memory command is directed to the adapter which is responsible for insuring that the command is executed at that remote memory.")

- Ekanadham: *See, e.g.*, 3:37-45 ("The preferred embodiment of our system that is based on a network of switch-based SMP nodes with an adapter attached to each node. FIG. 1 illustrates a high-level diagram of such a multiprocessing system. Each node has a plurality of processors P1, P2, . . . , PN interconnected to each other by a switch (SW).

The switch also interconnects the memory modules M1, M2, . . . , MN and adapters A. The nodes in turn, are connected to each other through a network as shown.")

- Ekanadham: *See, e.g.*, 3:49-56 ("The adapter connects to the switch and plays the role of either a memory or a processor. The behavior of the adapter is different for different memory lines. When a line is homed at the local memory of the node, the adapter behaves as a proxy processor for that line. When a line is homed at the memory of a remote node, the adapter behaves as a proxy memory for that line. These roles are illustrated in FIGS. 3A-3C and are elaborated further below.")

- Ekanadham: *See, e.g.*, 4:7-24 ("In a node in which a line is homed in the local memory, the adapter plays the role of a proxy processor representing the accesses to the line made by the processors in other nodes of the system. In this role, the adapter maintains a state for the line and the list of all nodes sharing that line. The state can be I (indicating that no other node has this line), E (indicating that some other node has exclusive copy of this line) or S (indicating that this line is shared by this and other nodes). As a proxy processor, the adapter receives requests from other adapters and performs the reads and writes in this node on their behalf. Whenever a local processor requires exclusive access to the line while it is in shared state, it communicates with other adapters and invalidates the line in all other nodes. When another node requests for exclusive copy of the line. The adapter only invalidates the copies in all other nodes, but also requests the local memory to grant the exclusive access. The memory controller treats the adapter as another processor.")

- Ekanadham: *See, e.g.*, 4:26-41 ("In a node in which a line is homed at a remote memory, the adapter acts as a proxy memory. It captures all the transactions for the corresponding address and runs the memory protocol. In this role, the adapter maintains a state for the line and the list of local caches sharing the line. The state can be I (indicating that no local cache this line), E (indicating that some local cache has exclusive copy of this line) or S (indicating that this line is shared by this and other nodes). As a proxy memory, the adapter responds to all requests to the line and obtains the contents of the line from the remote node (where that line is backed by memory) and supplies the contents to the local caches. It performs the usual coherence control operations in the node and coordinates with other adapters. In order to maintain global coherence, it may have to issue some bus transactions as a master, as illustrated later.")

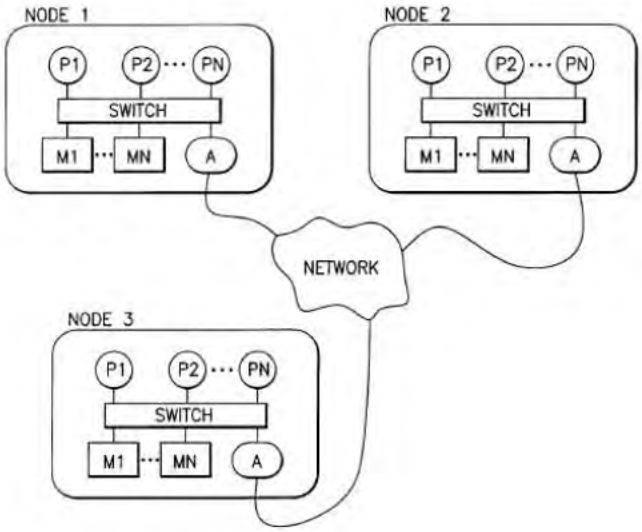

FIG.1
PRIOR ART

Ekanadham, Figure 1

- U.S. Patent No, 6,141,692 to Loewenstein: *See*, *e.g.*, 6:29-33 ("Requests for data and responses to those requests are exchanged between nodes by the respective HA, SA, and RA of each global interface (i.e., 415, 425, 435, and 445) in the form of data/control packets, thereby enabling each node to keep track of the status of all data cached therein.")

- Loewenstein: *See*, *e.g.*, 11:1-9 ("When the system is operating in NUMA mode, a typical read request (e.g., a Read-- To-- Share request) by processor 411a of node 410 occurs in the following manner. To initiate the request, processor 411a presents a virtual address (VA) to MMU 412a, which converts the VA into a GA and presents the GA to cache 413a. If there is a valid copy of the data line of interest in cache 413a (e.g., a shared or owned copy), then cache 413a provides the data to processor 411a via MMU 412a, thereby completing the read request.")

- Loewenstein: *See*, *e.g.*, 11:10-15 ("However, if cache 413a does not have a valid copy, then cache 413a presents the GA to the local interconnect 419 of its associated node. If the GA is not part of the node 410's local address space (i.e., node 410 is not the home node for the requested address), then the request is forwarded to the appropriate home node (i.e., node 420).")

- Loewenstein: *See*, *e.g.*, 11:40-50 ("If the home node is determined to have a valid copy of the requested data line, then the home node provides the data to the requesting node. In the case where the requesting node is also the home node, only an internal data transfer is required. Alternatively, where the home node is not the requesting node, then the global interface of the home node (global interface 425 in the above example) responds by retrieving the requested data line from the main memory 424 or from a

cache line which is owned by a processor within node 420, and sends the data line to the global interface 415 of the requesting node 410 via global interconnect 450.")

- Loewenstein: *See*, *e.g.*, 11:60-64 ("Upon receiving the data line, global interface 415 forwards the data line to cache 413a, which provides the data to the requesting processor 411a.")

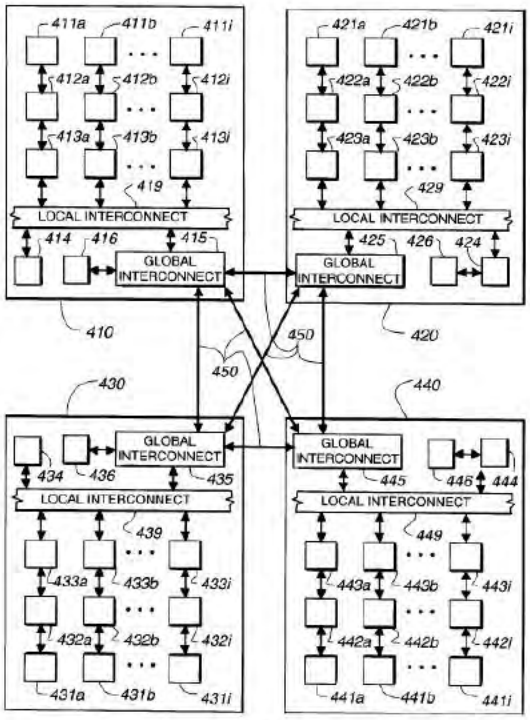

Fig. 4

Loewenstein, Figure 4

- U.S. Patent No, 6,751,721 to Webb: *See*, *e.g.*, Abstract ("A directory-based multiprocessor cache control scheme for distributing invalidate messages to change the state of shared data in a computer system. The plurality of processors are grouped into a plurality of clusters. A directory controller tracks copies of shared data sent to processors in the clusters. Upon receiving an exclusive request from a processor requesting permission to modify a shared copy of the data, the directory controller generates invalidate messages requesting that other processors sharing the same data invalidate that data. These invalidate messages are sent via a point-to-point transmission only to master processors in clusters actually containing a shared copy of the data. Upon receiving the invalidate message, the master processors broadcast the invalidate message in an ordered fan-in/fan-out process to each processor in the cluster. All processors within the cluster invalidate a local copy of the shared data if it exists and once the master processor receives acknowledgements from all processors in the cluster, the master processor sends an invalidate acknowledgment message to the processor that originally requested the exclusive rights to the shared data. The cache coherency is scalable and may be implemented using the hybrid

point-to-point/broadcast scheme or a conventional point-to-point only directory-based invalidate scheme.")

As illustrated by the prior art references above, it was well known before the priority dates of the Asserted Patents to implement a "cache coherence controller" "constructed to act as a probing agent pair" in multiprocessor systems. Indeed, a person of ordinary skill would have been motivated to implement a "cache coherence controller" "constructed to act as a probing agent pair" in a multiprocessor system as described below:

- Ekanadham: *See, e.g.*, 1:23-35 ("Technology considerations limit the size of an SMP node to a small number of processors. A method for building a shared-memory multiprocessor with a larger number of processors is to connect a number of SMP nodes with a network, and provide an adapter to extend the SMP's memory across the SMP nodes (see FIG. 1). Existing adapter designs plug into the memory bus of bus-based SMP nodes and collectively provide shared memory across the system, so that any processor in any node can access any location in any memory module in the system. Resources within a node are termed local and resources on other nodes are termed remote.")

- Ekanadham: *See, e.g.*, 2:37-41 ("By appearing as either a local processor or a local memory, the adapter uses the local SMP coherence protocol within a node to accomplish the above tasks, without any changes to the memory controllers.")

- Loewenstein: *See, e.g.*, 5:1-8 ("Since global interface 115 is also responsible for maintaining global cache coherency, global interface 115 includes a hardware and/or software implemented cache-coherency mechanism for maintaining coherency between the respective caches and main memories of nodes 110, 120, . . . 180. Cache coherency is essential in order for the system 100 to properly execute shared-memory programs correctly.")

Accordingly, it would have been obvious to implement a "cache coherence controller" "constructed to act as a probing agent pair" in a multiprocessor system having multiple clusters of processors while maintaining coherency with a reasonable expectation of success. Furthermore, it would have been obvious to implement a "cache coherence controller" "constructed to act as a probing agent pair" because such a modification would simply be the use of a known technique (*e.g.*, a cache coherence controller "constructed to act as a probing agent pair") to improve similar

devices (*e.g.*, multiprocessor systems) in the same way (*e.g.*, improve performance and scalability while maintaining coherency).

10. *"Cache Coherence Controller" "Constructed to Act As A Remote Memory"*

Some of the Asserted Claims are directed to a "cache coherence controller" "constructed to act as a remote memory." For example, claim 20.1 of the '409 patent recites "the cache coherence controller is constructed to act as a remote memory." *See also*, *e.g.*, '409 patent claim 49.1; and '636 patent claim 31.1. At least under Memory Integrity's apparent infringement theories, "cache coherence controllers" "constructed to act as a remote memory" were well-known in the art before the priority dates of the Asserted Patents. *See*, *e.g.*, Exhibits A-1 – A-9, claims 20.1 and 49.1; and Exhibits B-1 – B-19, claim 31.1. The following discussion further shows that, at least under Memory Integrity's apparent infringement theory, it was well known and conventional to implement "cache coherence controllers" "constructed to act as a remote memory" in multiprocessor systems.

At least under Memory Integrity's apparent infringement theories, there are many examples of prior art references that disclose implementing "cache coherence controllers" "constructed to act as a remote memory." Examples of prior art references that disclose and further demonstrate that such was well known include:

- "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor." Lenoski (1990): *See*, *e.g.*, p. 1, para. 2 ("We are currently building a prototype of a scalable shared memory multiprocessor. The system provides high processor performance and scalability though the use of coherent caches and a directory-based coherence protocol. The high-level organization of the prototype, called DASH (Directory Architecture for Shared memory) [17]. is shown in Figure 1. The architecture consists of a number of processing nodes connected through a high-bandwidth low-latency interconnection network. The physical memory in the machine is distributed among the nodes of the multiprocessor, with all memory accessible to each node. Each processing node, or cluster, consists of a small number of high-performance processors with their individual caches, a portion of the shared-memory, a common cache for pending remote accesses, and a directory

controller interfacing the cluster to the network. A bus-based snoopy scheme is used to keep caches coherent within a cluster, while inter-node cache consistency is maintained using a distributed directory-based coherence protocol.")

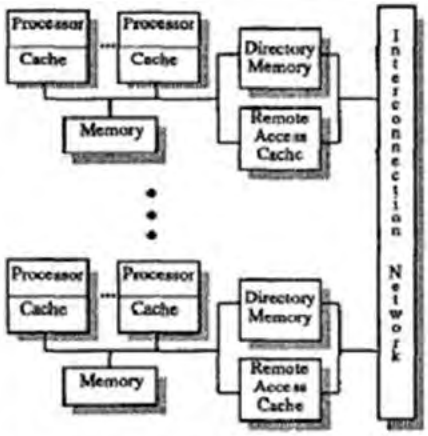

Figure 1: General architecture of DASH.

Lenoski (1990), Figure 1

- Lenoski (1990): *See, e.g.*, p. 1, para. 4 ("In DASH, each processing node has a directory memory corresponding to its portion of the shared physical memory. For each memory block, the directory memory stores the identities of all remote nodes caching that block. Using the directory memory, a node writing a location can send point-to-point invalidation or update messages to those processors that are actually caching that block.")

- Lenoski (1990): *See, e.g.*, p. 3, para. 5 ("The directory controller (DC) contains the directory memory corresponding to the portion of main memory present within the cluster. It also initiates out-bound network requests and replies. The pseudo-CPU (PCPU) is responsible for buffering incoming requests and issuing such requests on the cluster bus. It mimics a CPU on this bus on behalf of remote processors except that responses from the bus are sent out by the directory controller. The reply controller (RC) tracks outstanding requests made by the local processors and receives and buffers the corresponding replies from remote clusters. It acts as memory when the local processors are allowed to retry their remote requests. The network interface and the local portion of the network itself reside on the directory card. The interconnection network consists of a pair of meshes. One mesh is dedicated to the request messages while the other handles replies. These meshes utilize wormhole routing [9] to minimize latency. Finally, the board contains hardware monitoring logic and miscellaneous control and status registers. The monitoring logic samples a variety of directory board and bus events from which usage and performance statistics can be derived.")

- Lenoski (1990): *See, e.g.*, p. 4.5, para. 4 ("In the protocol, invalidation acknowledges are sent to the local cluster that initiated the memory request. An alternative would be for the home cluster to gather the acknowledges, and, when all have been received,

send a message to the requesting cluster indicating that the request has been completed. We chose the former because it reduces the waiting time for completion of a subsequent fence operation by the requesting cluster and reduces the potential of a hot spot developing at the memory.")

- "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor." Lenoski (1992): *See, e.g.*, p. 150 ("A DASH system consists of a number of modified 4D/240 systems that have been supplemented with a directory controller board. This directory controller board is responsible for maintaining the cache coherence across the nodes and serving as the interface to the interconnection network.")

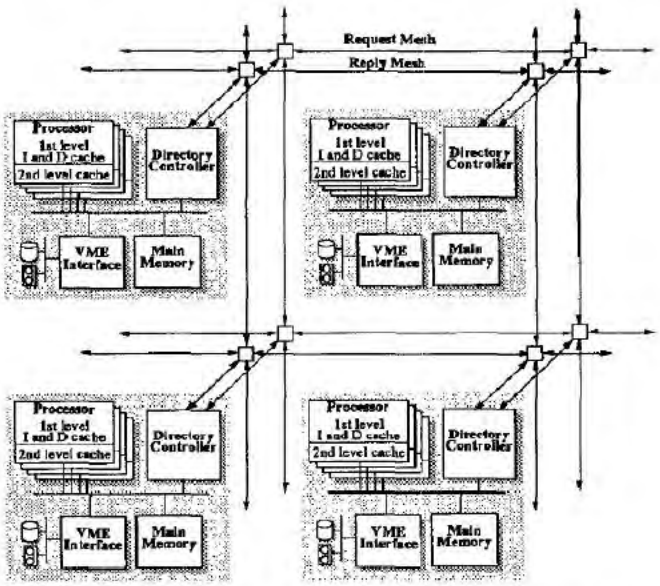

Figure 2: Block diagram of sample 2 x 2 DASH system.

Lenoski (1992), Figure 2

- U.S. Patent No. 6,055,610 to Smith: *See, e.g.*, 11:44-55 ("A flow chart of the basic method M1 of handling a data request is flow charted in FIG. 3. At step S1, processor P11 issues a read request of data stored in main memory MM0. At step S2, caches C10-C13 of requester cell MC1 are examined to determine if the request can be met locally. First, associated cache C11 is checked. A hit allows the request to be met locally. A miss refers the request to the requestor's coherency controller CC1. Coherency controller CC1 initiates a local snoop while referring the request to owner cell MC0. If the snoop results in a hit, the request can be met locally. If the data is held privately by another local processor, e.g., processor P12, coherency controller requests that the data be made public so that the request can be met. Only if the local snoop misses is involvement of the owner cell MC0 required.")

- Smith: *See, e.g.*, 11:56-63 ("At step S3, coherency controller CC0 of owner cell MC0 initiates a local snoop of its caches, accesses fast directory FD0, and initiates access of main memory MM0. Coherency controller CC0 determines whether or not the

fast-directory data calls for a recall and whether the directory cache data is consistent with the local snoop results. If the directory data is consistent with the snoop results and if a recall is indicated, it is initiated at step S4.")

- Smith: *See, e.g.*, 12:3-8 ("Once the recall process is complete, the requested data is transferred to the requester cell MC1, coherency controller CC1, cache C11, and processor P11, at step S6. State information in cache C11, fast directory FD0, and the coherency directory of main memory MM0 is updated as necessary. This completes method M1.")

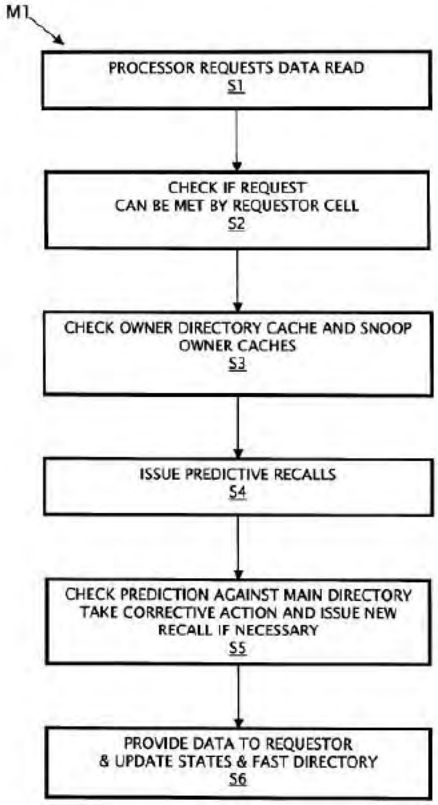

Figure 3

Smith, Figure 3

- U.S. Patent No, 6,085,295 to Ekanadham: *See, e.g.*, 2:25-33 ("In a node where a remote line is brought into the cache of a processor, but not into the node's memory, the adapter acts as a proxy memory representing the remote memory that the line is mapped onto. More specifically, when a memory command is issued from a local processor to a remote memory, the memory command is directed to the adapter which is responsible for insuring that the command is executed at that remote memory.")

- Ekanadham: *See, e.g.*, 3:37-45 ("The preferred embodiment of our system that is based on a network of switch-based SMP nodes with an adapter attached to each node. FIG. 1 illustrates a high-level diagram of such a multiprocessing system. Each node has a plurality of processors P1, P2, . . . , PN interconnected to each other by a switch (SW).

The switch also interconnects the memory modules M1, M2, . . . , MN and adapters A. The nodes in turn, are connected to each other through a network as shown.")

- Ekanadham: *See, e.g.*, 3:49-56 ("The adapter connects to the switch and plays the role of either a memory or a processor. The behavior of the adapter is different for different memory lines. When a line is homed at the local memory of the node, the adapter behaves as a proxy processor for that line. When a line is homed at the memory of a remote node, the adapter behaves as a proxy memory for that line. These roles are illustrated in FIGS. 3A-3C and are elaborated further below.")

- Ekanadham: *See, e.g.*, 4:7-24 ("In a node in which a line is homed in the local memory, the adapter plays the role of a proxy processor representing the accesses to the line made by the processors in other nodes of the system. In this role, the adapter maintains a state for the line and the list of all nodes sharing that line. The state can be I (indicating that no other node has this line), E (indicating that some other node has exclusive copy of this line) or S (indicating that this line is shared by this and other nodes). As a proxy processor, the adapter receives requests from other adapters and performs the reads and writes in this node on their behalf. Whenever a local processor requires exclusive access to the line while it is in shared state, it communicates with other adapters and invalidates the line in all other nodes. When another node requests for exclusive copy of the line. The adapter only invalidates the copies in all other nodes, but also requests the local memory to grant the exclusive access. The memory controller treats the adapter as another processor.")

- Ekanadham: *See, e.g.*, 4:26-41 ("In a node in which a line is homed at a remote memory, the adapter acts as a proxy memory. It captures all the transactions for the corresponding address and runs the memory protocol. In this role, the adapter maintains a state for the line and the list of local caches sharing the line. The state can be I (indicating that no local cache this line), E (indicating that some local cache has exclusive copy of this line) or S (indicating that this line is shared by this and other nodes). As a proxy memory, the adapter responds to all requests to the line and obtains the contents of the line from the remote node (where that line is backed by memory) and supplies the contents to the local caches. It performs the usual coherence control operations in the node and coordinates with other adapters. In order to maintain global coherence, it may have to issue some bus transactions as a master, as illustrated later.")

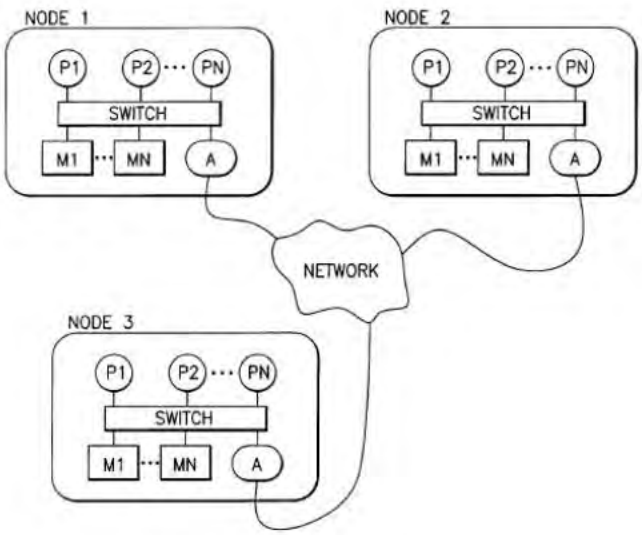

FIG.1
PRIOR ART

Ekanadham, Figure 1

- U.S. Patent No, 6,141,692 to Loewenstein: *See*, *e.g.*, 6:29-33 ("Requests for data and responses to those requests are exchanged between nodes by the respective HA, SA, and RA of each global interface (i.e., 415, 425, 435, and 445) in the form of data/control packets, thereby enabling each node to keep track of the status of all data cached therein.")

- Loewenstein: *See*, *e.g.*, 11:1-9 ("When the system is operating in NUMA mode, a typical read request (e.g., a Read-- To-- Share request) by processor 411a of node 410 occurs in the following manner. To initiate the request, processor 411a presents a virtual address (VA) to MMU 412a, which converts the VA into a GA and presents the GA to cache 413a. If there is a valid copy of the data line of interest in cache 413a (e.g., a shared or owned copy), then cache 413a provides the data to processor 411a via MMU 412a, thereby completing the read request.")

- Loewenstein: *See*, *e.g.*, 11:10-15 ("However, if cache 413a does not have a valid copy, then cache 413a presents the GA to the local interconnect 419 of its associated node. If the GA is not part of the node 410's local address space (i.e., node 410 is not the home node for the requested address), then the request is forwarded to the appropriate home node (i.e., node 420).")

- Loewenstein: *See*, *e.g.*, 11:40-50 ("If the home node is determined to have a valid copy of the requested data line, then the home node provides the data to the requesting node. In the case where the requesting node is also the home node, only an internal data transfer is required. Alternatively, where the home node is not the requesting node, then the global interface of the home node (global interface 425 in the above example) responds by retrieving the requested data line from the main memory 424 or from a

cache line which is owned by a processor within node 420, and sends the data line to the global interface 415 of the requesting node 410 via global interconnect 450.")

- Loewenstein: *See, e.g.*, 11:60-64 ("Upon receiving the data line, global interface 415 forwards the data line to cache 413a, which provides the data to the requesting processor 411a.")

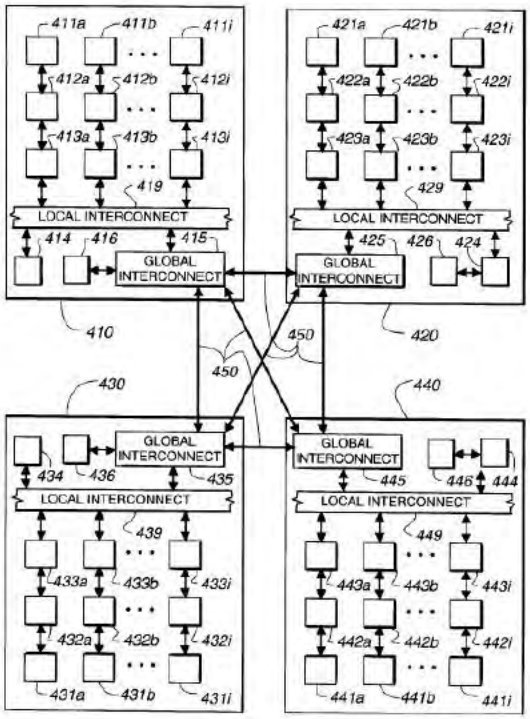

Fig. 4

Loewenstein, Figure 4

- U.S. Patent No, 6,751,721 to Webb: *See, e.g.*, Abstract ("A directory-based multiprocessor cache control scheme for distributing invalidate messages to change the state of shared data in a computer system. The plurality of processors are grouped into a plurality of clusters. A directory controller tracks copies of shared data sent to processors in the clusters. Upon receiving an exclusive request from a processor requesting permission to modify a shared copy of the data, the directory controller generates invalidate messages requesting that other processors sharing the same data invalidate that data. These invalidate messages are sent via a point-to-point transmission only to master processors in clusters actually containing a shared copy of the data. Upon receiving the invalidate message, the master processors broadcast the invalidate message in an ordered fan-in/fan-out process to each processor in the cluster. All processors within the cluster invalidate a local copy of the shared data if it exists and once the master processor receives acknowledgements from all processors in the cluster, the master processor sends an invalidate acknowledgment message to the processor that originally requested the exclusive rights to the shared data. The cache

coherency is scalable and may be implemented using the hybrid point-to-point/broadcast scheme or a conventional point-to-point only directory-based invalidate scheme.")

As illustrated by the prior art references above, it was well known before the priority dates of the Asserted Patents to implement a "cache coherence controller" "constructed to act as a remote memory" in a multiprocessor system, at least under Memory Integrity's apparent infringement theories. A person of ordinary skill would have been motivated to implement a "cache coherence controller" "constructed to act as a remote memory" in a multiprocessor system as described below:

- Ekanadham: *See, e.g.*, 1:23-35 ("Technology considerations limit the size of an SMP node to a small number of processors. A method for building a shared-memory multiprocessor with a larger number of processors is to connect a number of SMP nodes with a network, and provide an adapter to extend the SMP's memory across the SMP nodes (see FIG. 1). Existing adapter designs plug into the memory bus of bus-based SMP nodes and collectively provide shared memory across the system, so that any processor in any node can access any location in any memory module in the system. Resources within a node are termed local and resources on other nodes are termed remote.")

- Ekanadham: *See, e.g.*, 2:37-41 ("By appearing as either a local processor or a local memory, the adapter uses the local SMP coherence protocol within a node to accomplish the above tasks, without any changes to the memory controllers.")

- Loewenstein: *See, e.g.*, 5:1-8 ("Since global interface 115 is also responsible for maintaining global cache coherency, global interface 115 includes a hardware and/or software implemented cache-coherency mechanism for maintaining coherency between the respective caches and main memories of nodes 110, 120, . . . 180. Cache coherency is essential in order for the system 100 to properly execute shared-memory programs correctly.")

Accordingly, it would have been obvious to implement a "cache coherence controller" "constructed to act as a remote memory" in a multiprocessor system having multiple clusters of processors while maintaining coherency with a reasonable expectation of success. Furthermore, it would have been obvious to implement a "cache coherence controller" "constructed to act as a remote memory" because such a modification would simply be the use of a known technique (*e.g.*,

a cache coherence controller "constructed to act as a remote memory") to improve similar devices (*e.g.*, multiprocessor systems) in the same way (*e.g.*, improve performance and scalability while maintaining coherency).

11. *"Shared Memory Address Space"*

Some of the Asserted Claims are directed to a shared memory address space. For example, claim 9.1 of the '409 patent recites "the plurality of local processors in the local cluster share a memory address space with a plurality of non-local processors in the non-local cluster." *See also*, *e.g.*, '409 patent claims 25.6, 27.1, 34.6, and 36.1; '636 patent claims 13.1 and 24.1; and '254 patent claims 1.5-1.11 and 8.1. Claim 42.4 of '409 patent also recites "the first and second processors sharing a common virtual address space." At least under Memory Integrity's apparent infringement theories, sharing an address space and/or sharing a common virtual address space between processors in a multiprocessor system was well-known in the art before the priority dates of the Asserted Patents. *See*, *e.g.*, Exhibits A-1 – A-9, claims 9.1, 25.6, 27.1, 34.6, 36.1, and 42.4; Exhibits B-1 – B-19, claims 13.1 and 24.1; and Exhibits E-1 – E-14, claims 1.5-1.11 and 8.1. The following discussion further shows that, at least under Memory Integrity's apparent infringement theory, it was well known and conventional to share an address space and/or share a common virtual address space between processors in multiprocessor systems.

As an initial matter, the Asserted Patents acknowledge that a shared memory address space was well known. *See*, *e.g.*, '409 patent at 2:18-3:40 ("Background of the Invention … Although, cache coherency mechanisms such as bus arbitration are effective, using a shared bus limits the number of processors that can be implemented in a single system with a single memory space. … Performance limitations have led to the development of a point-to-point architecture for connecting processors in a system with a single memory space. … However, using a point-to-point architecture to connect multiple processors in a multiple cluster system sharing a single memory

space presents its own problems."); '636 patent at 1:33-2:59; '121 patent at 1:20-2:38; '206 patent at 1:13-38; and '254 patent at 1:16-41.

Indeed, under Memory Integrity's apparent infringement theories, there are many examples of prior art references that disclose sharing an address space and/or a common virtual address space between processors in a multiprocessor system and further demonstrate that such features were well known include:

- "Computer Organization & Design," Patterson et al. (1998): *See*, *e.g.*, p. 713 ("Single address space multiprocessors come in two styles. The first takes the same time to access main memory no matter which processor requests it and no matter which word is asked. Such machines are called *uniform memory access (UMA)* multiprocessors or *symmetric multiprocessors (SMP)*. In the second style, some memory accesses are faster than others depending on which processor asks for which word. Such machines are called *nonuniform memory access (NUMA)* multiprocessors. As you might expect, there are more programming challenges to get highest performance from a NUMA multiprocessor than a UMAmultiprocessor, but NUMAmachines can scale to larger sizes and hence are potentially higher performance.")

- Patterson: *See*, *e.g.*, p. 713 ("The alternative model to shared memory for communicating uses *message passing* for communicating among processors. Message passing is required for machines with *private memories*, in contrast to shared memory. As an extreme example, processors in different desktop computers communicate by passing messages over a local area network. Provided the system has routines to send and receive messages, coordination is built in with message passing since one processor knows when a message is sent and the receiving processor knows when a message arrives. The receiving processor can then send a message back to the sender saying the message has arrived, if the sender needs that confirmation.")

Name	Maximum number of processors	Processor name	Processor clock rate	Maximum memory size/ system	Communi-cations BW/link	Node	Topology
Cray Research T3E	2048	Alpha 21164	450 MHz	524,288 MB	1200 MB/sec	4-way SMP	3-D torus
HP/Convex Exemplar X-class	64	PA-8000	180 MHz	65,536 MB	980 MB/sec	2-way SMP	8-way crossbar + ring
Sequent NUMA-Q	32	Pentium Pro	200 MHz	131,072 MB	1024 MB/sec	4-way SMP	Ring
SGI Origin2000	128	MIPS R10000	195 MHz	131,072 MB	800 MB/sec	2-way SMP	6-cube
Sun Enterprise 10000	64	UltraSPARC 1	250 MHz	65,536 MB	1600 MB/sec	4-way SMP	16-way crossbar

FIGURE 9.9 Characteristics of multiprocessor computers connected by a network that are for sale in 1997. All these machines have a shared address space with nonuniform memory access time except for the Sun Enterprise 10000, which offers a shared address with uniform memory access time. And all these machines except the Cray Research T3E are cache coherent, with the HP, Sequent, and SGI using directories. The Sun machine uses buses for addresses and a switch for data, so it supports coherency with conventional snooping on the address buses. Communication bandwidth is peak per link, counting all bytes sent including network headers. The bisection bandwidth typically scales with the number of processors. (See *www.mkp.com/cod2e.htm* for pointers to these and more recent network-connected multiprocessors.)

Patterson, Figure 9.9

- Patterson: *See, e.g.*, p. 579 ("7.4 Virtual Memory In the previous section, we saw how caches served as a method for providing fast access to recently used portions of a program's code and data. Similarly, the main memory can act as a 'cache' for the secondary storage, usually implemented with magnetic disks. This technique is called virtual memory. There are two major motivations for virtual memory: to allow efficient and safe sharing of memory among multiple programs and to remove the programming burdens of a small, limited amount of main memory.")

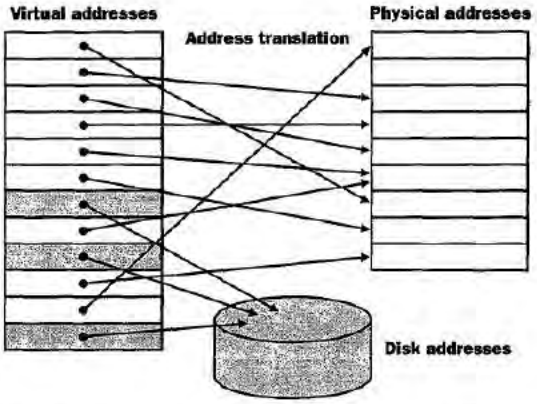

FIGURE 7.20 In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses). The processor generates virtual addresses while the memory is accessed using physical addresses. Both the virtual memory and the physical memory are broken into pages, so that a virtual page is really mapped to a physical page. Of course, it is also possible for a virtual page to be absent from main memory and not be mapped to a physical address, residing instead on disk. Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.

Patterson, Figure 7.20

- "Computer Organization," Hamacher (2001): *See, e.g.*, p. 637 ("In Chapter 5 we saw that the organization of the memory in a uniprocessor system has a large impact on performance. The same is true in multiprocessor systems. To exploit the locality of reference phenomenon, each processor usually includes a primary cache and a secondary cache. If the organization in Figure 12.2 is used, then each processor module can be connected to the communication network as shown in Figure 12.12. Only the secondary cache is shown in the figure since the primary cache is assumed to be a part of the processor chip. The memory modules are accessed using a single global address space, where a range of physical addresses is assigned to each memory module. In such a shared memory system, the processors access all memory modules in the same way. From the software standpoint, this is the simplest use of the address space.")

- Hamacher: *See, e.g.*, p. 637-638 ("In NUMA-organized multiprocessors, shown in Figure 12.3, each node contains a processor and a portion of the memory. A natural way of implementing the node is illustrated in Figure 12.13. In this case, it is also convenient to use a single global address space. Again, the processor accesses all memory modules in the same way, but the accesses to the local memory component of the global address space take less time to complete than accesses to remote memory modules.")

- Hamacher: *See, e.g.*, p. 638 ("In the organization of Figure 12.4, each processor accesses directly only its own local memory. Thus, each memory module constitutes the private address space of one processor; there is no global address space. Any interaction among programs or processes running on different processors is implemented by sending messages from one processor to another. In this form of communication, each processor views the interconnection network as an I/O device. In effect, each node in such a system behaves as a computer in the same manner as discussed in previous chapters for uniprocessor machines. For this reason, systems of this type are referred to also as multicomputers. This organization, provides the easiest way to connect a number of computers into a large system. Communication between tasks running on different computers is relatively slow because the exchange of messages requires software intervention. We consider this type of system in Section 12.7.")

- "Parallel Computer Architecture," Culler et al. (1998): *See, e.g.*, p. 28 ("One of the most important classes of parallel machines is shared memory multiprocessors. The key property of this class is that communication occurs implicitly as a result of conventional memory access instructions (i.e., loads and stores). This class has a long history, dating at least to precursors of mainframes in the early 1960s,[3] and today it has a role in almost every segment of the computer industry. Shared memory multiprocessors serve to provide better throughput on multiprogramming workloads, as well as to support parallel programs. Thus, they are naturally found across a wide range of scale, from a few processors to perhaps hundreds. This section examines the communicator architecture of shared memory machines and the key organizational issues for small-scale designs and large configurations.")

- U.S. Patent No. 5,796,605 to Hagersten: *See*, *e.g.*, Abstract ("A technique for system memory space address mapping in a multiprocessor computer system is provided. The disclosed mapping architecture may be applied to a multiprocessor computer system having multiple processing nodes (SMP nodes), where each processing node may include multiple processors. The system memory address space is split into different regions such that each of the n SMP nodes is assigned 1/n of the total address space. Cache coherency state information is stored for the memory in each SMP node.")

- U.S. Patent No. 6,336,177 to Stevens: *See*, *e.g.*, 6:59-65 ("The light shaded sections, 402, 406, 410, and 414 indicate that these areas of the virtual address space are shared by at least two threads. Note that in this example, the virtual address space 304 is represented in a wrap-around fashion so that the shared section 402 appears both the top and at the bottom of the virtual address space 304."); *See*, *e.g.*, 7:49-57 ("FIG. 5 depicts an example of memory placement that could occur, if the known memory access patterns, as described above with reference to FIG. 4, are not taken into account for the purpose of maximizing memory placement locality. Accordingly, in this example, the threads 302 a and 302 b are randomly mapped to the node 502, in one comer of the system 316, while the threads 302 c and 302 d are randomly mapped to the node 504, in the opposite comer of the system 316.")

- U.S. Patent No. 6,490,671 to Frank: *See*, *e.g.*, 1:46-56 ("In a multi-processor machine, a process may be divided into threads of execution, some threads executing on different CPUs. All of a process's threads share a common virtual address space. Each of the CPUs, however, maintains its own copy of the TLB. When any of the CPUs in the machine invalidates an entry in the TLB, each CPU is notified, traditionally by means of a hardware interrupt, that there has been a change to the TLB and refreshes its copy. Invalidating TLB entries, however, is very expensive, because all of the CPUs on the machine stop their processing to perform the refresh.")

- U.S. Patent No. 6,631,448 to Weber: *See*, *e.g.*, 1:34-51 ("A distributed, shared-memory system is known as a distributed shared-memory (DSM) or a non-uniform memory access (NUMA) architecture." "DSM architecture provides a single shared address space to the programmer where all memory locations may be accessed by every processor. As there is no need to distribute data or explicitly communicate data between the processors in software, the burden of programming a parallel machine is simpler in a DSM model. In addition, by dynamically partitioning the work, DSM architecture makes it easier to balance the computational load between processors. Finally, as shared memory is the model provided on small-scale multiprocessors, DSM architecture facilitates the portability of programs parallelized for a small system to a larger shared-memory system. In contrast, in a message-passing system, the programmer is responsible for partitioning all shared data and managing communication of any updates.")

- U.S. Patent Application No. 2003/0009640 to Arimilli: *See*, *e.g.*, para. [0043] ("To illustrate the present invention, an embodiment will be described in which instruction sequencing unit 52 and LSU 68 of each CPU 20 in NUMA data processing system 10 reference instructions and data utilizing 32-bit effective addresses, meaning that CPUs

20 have a 4 Gbyte (2^{32}) effective address space. This effective address space is a subset of a much larger virtual address space referenced by 52-bit virtual addresses. This virtual address space, which is shared by all CPUs 20 in NUMA data processing system 10, is partitioned into a number of (e.g., 4 Kbyte) memory pages, which each have an Page Table Entry (PTE) address descriptor that associates the base virtual address of the memory page with the corresponding physical address of the memory page in one of system memories 26.")

- U.S. Patent No. 6,684,305 to Deneau: *See*, *e.g.*, 7:21-36 ("The main processor operating system 206 creates and maintains both the main processor page tables 200 and the co-processor page tables 202, and embodies virtual-to-physical address translation mechanisms of both the main processor 102 and the co-processor, 104. The main processor page tables 200 are used by the main processor 102 to translate virtual addresses generated within the main processor 102 to physical addresses within the shared memory 106. The co-processor page tables 202 are used by the co-processor 104 to translate virtual addresses generated within the co-processor 104 to physical addresses within the shared memory 106. The main processor 102 and the co-processor 104 share a common physical address space within the shared memory 106. For simplicity of operation, the main processor 102 and the co-processor 104 may also share a common virtual address space.")

As illustrated by the prior art references above, it was well known before the priority dates of the Asserted Patents to share an address space and/or a common virtual address space between processors in a multiprocessor system, at least under Memory Integrity's apparent infringement theories. Indeed, a person of ordinary skill would have been motivated to implement a shared an address space and/or a common virtual address space as described below:

- Hamacher: *See*, *e.g.*, p. 637 ("The memory modules are accessed using a single global address space, where a range of physical addresses is assigned to each memory module. In such a shared memory system, the processors access all memory modules in the same way. From the software standpoint, this is the simplest use of the address space.")

- Weber: *See*, *e.g.*, 1:38-51 ("DSM architecture provides a single shared address space to the programmer where all memory locations may be accessed by every processor. As there is no need to distribute data or explicitly communicate data between the processors in software, the burden of programming a parallel machine is simpler in a DSM model. In addition, by dynamically partitioning the work, DSM architecture makes it easier to balance the computational load between processors. Finally, as shared memory is the model provided on small-scale multiprocessors, DSM architecture facilitates the portability of programs parallelized for a small system to a larger shared-memory system. In contrast, in a message-passing system, the programmer is responsible for partitioning all shared data and managing communication of any updates.")

- Patterson: *See*, *e.g.*, p. 579 ("7.4 Virtual Memory … This technique is called virtual memory. There are two major motivations for virtual memory: to allow efficient and safe sharing of memory among multiple programs and to remove the programming burdens of a small, limited amount of main memory.")

Accordingly, it would have been obvious to implement a shared address and/or a common virtual address space between processors in a multiprocessor system because doing so would simply be an obvious engineering design choice by selecting one of a finite number of known options (*e.g.*, shared address space or message passing and physical mapping or virtual address mapping) according to the desired multiprocessor system implementation with a reasonable expectation of success. Furthermore, it would have been obvious to implement a shared address space and/or a common virtual address space between processors in a multiprocessor system because such a modification would simply be the use of a known technique (*e.g.*, shared address space and/or a common virtual address space) to improve similar devices (*e.g.*, multiprocessor systems) in the same way (*e.g.*, simplified programming, improved computational load balancing, efficient and safe sharing of memory among multiple programs, etc.).

12. *"Protocol Engines"*

Some of the Asserted Claims are directed to one or more protocol engines. For example, some of the Asserted claims are directed to a cache coherence controller comprising a protocol engine coupled to interface circuitry. *See*, *e.g.*, '409 patent claims 7.4, 8.1, 10.1, 12.1, and 22.1; '636 patent claims 22.3. 22.4, 23.1, 27.1, and 33.1. Some of the Asserted Claims are directed to an interconnection controller including one or more protocol engines for processing transactions, for example, in accordance with a cache coherence protocol, or for processing interrupts. *See*, *e.g.*, '206 patent claims 1.4, 1.5, 1.6, 14.2, 19.2, 19.3, 21.4, 21.5, 21.6, 30.3, 30.4, 30.5, 35.1, 38.2, 39.4, 39.5, 39.7, 39.8, 41.1, and 44.2; '254 patent claims 1.4, 1.5, 1.6, 1.8, 1.10, 1.11, 6.2, and 7.1. At least under Memory Integrity's apparent infringement theories, an interconnection or cache

coherence controller including one or more protocol engines coupled to interface circuitry, for processing transactions in accordance with a cache coherence protocol, etc. was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits A-1 – A-9 claims 7.4, 8.1, 10.1, 12.1, and 22.1; Exhibits B-1 – B-19 claims 22.3. 22.4, 23.1, 27.1, and 33.1; Exhibits D-1 – D-14, claims 1.4, 1.5, 1.6, 14.2, 19.2, 19.3, 21.4, 21.5, 21.6, 30.3, 30.4, 30.5, 35.1, 38.2, 39.4, 39.5, 39.7, 39.8, 41.1, and 44.2; and Exhibits E-1 – E-14 claims 1.4, 1.5, 1.6, 1.8, 1.10, 1.11, 6.2, and 7.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose an interconnection or cache coherence controller including a plurality of protocol engines coupled to interface circuitry, for processing transactions in accordance with a cache coherence protocol, etc. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
  - *Page* 6 ("The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers."),
  - Figure 1,
  - Page 7 ("Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-bytewide buses, one on and one off, operating at one-third processor frequency."),
  - Page 15 ("The unified second-level cache is shared across the two processors on the POWER4 chip. Figure 5 shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers."),

- Figure 5,
- Page 16 ("The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two processors (for either an L1 data-cache reload or an instruction fetch) or from one of the store queues."),
- Page 16 ("Included in each L2 controller are four snoop processors responsible for managing coherency operations snooped from the fabric."),
- Page 16 ("The L2 cache implements an enhanced version of the MESI coherency protocol . . .").

- *See*, *e.g.*, Behling et al., "The POWER4 Processor Introduction and Tuning Guide," at
  - Page 15 ("Stores can be sent to the L2 cache at a maximum rate of one store per cycle. Store data is directed to the proper L2 controller (through a hashing function) by way of the storage slice queue (SSQ) and the L2 store queue (STQ)."),
  - Page 17 ("Each POWER4 chip has an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree pseudo-LRU algorithm. The L2 cache is a unified cache: it caches instructions, data, and page table entries. The L2 cache is also shared by the processors on the chip."),
  - Page 18 ("Memory coherency in the system is enforced primarily at the L2 cache level by L2 cache controllers. Each L2 has associated command queues, known as coherency processors. Snoop processors within each controller observe all transactions in the system and respond accordingly, providing responses or delivering cache lines if the situation merits."),
  - Page 30 ("The size of the L2 cache is 1440 KB per POWER4 chip, and this is shared between the two processors in the chip. As with the L1 data cache, the cache line size is 128 bytes. The replacement policy is pseudo-LRU (least recently used) so frequently accessed cache lines should be readily maintained in the cache. The L2 cache is a combined data and instruction cache. Instruction caching aspects of the L2 cache are not considered here. The L2 cache is divided into three equal parts, each under control of a separate L2 cache controller. The particular portion a line is stored is in is determined from the real memory address using a hashing algorithm. Sixteen consecutive double-precision Fortran array elements (138 bytes) are held in the same cache line, and therefore under control of the same cache controller. The 17th element will be in a different cache line and the hashing algorithm guarantees it will be stored under control of a different cache controller.").
- *See*, *e.g.*, IBM POWER4 processor (as disclosed at least in the above references)
- *See*, *e.g.*, Intel 870 Chipset [redacted]

[redacted]

- *See, e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108."),
  - Figure 1
  - [4:20-32] ("In one embodiment, the snoop filter 204 may be divided into four snoop filter (SF) interleaves 208. While in one embodiment the snoop filter 204 maintains coherency data for all lines of the caches 114, each SF interleave 208 may maintain coherency data for a unique subset of the cache lines. For example, two bits of a line's memory address may be used to identify which of the SF interleaves 208 maintains the coherency data for the particular line. Further, by assigning unique subsets to each of the SF interleaves 208, the SF interleaves 208 may operate in parallel and may increase the overall performance of the coherent switch 106 in comparison to a non-interleaved snoop filter 204 that may process a single request at time."),
  - Figure 2
  - [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

As acknowledged by the Applicant during prosecution of U.S. Patent No. 8,815,602, to which both the '206 and '254 patents claim priority, U.S. Patent Publication No. 2002/0007443 to Gharachorloo discloses an interconnection controller including two protocol engines for processing transactions in accordance with a cache coherence protocol, specifically a local protocol engine and a remote protocol engine. *See, e.g.*, U.S. Patent No. 8,815,602 Prosecution History, Applicant Response 2-13-2008 at Pages 8-9, Appeal Brief 10-3-2008 at Pages 7-8, 10. For example, Applicant stated that "Gharachorloo clearly teaches that each multi-processor node includes only a *single* protocol engine for processing local memory transactions, i.e., the HPE, and a *single* protocol engine for processing remote memory transaction, i.e., the RPE." *See, e.g.*, U.S. Patent No. 8,815,602 Prosecution History, Applicant Response 2-13-2008 at Page 9, Appeal Brief 10-3-2008 at Page 8. Furthermore, the Board of Patent Appeals and Interferences found that U.S. Patent No. 6,370,585 to Hagersten in view of Gharachorloo teaches or suggests a plurality of remote protocol engines for processing transactions targeting remote memory and a plurality of local protocol engines for processing transactions targeting local memory. *See, e.g.*, U.S. Patent No. 8,815,602 Prosecution History, Decision on Appeal 10-27-2011 at Pages 7-9.

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and Exhibits E-1 – E-14 to include a plurality of protocol engines coupled to interface circuitry and for processing transactions in accordance with a cache coherence protocol, at least under Memory Integrity's apparent infringement theories. *See, e.g.*, Exhibits A-1 – A-9 claims 7.4, 8.1, 10.1, 12.1, and 22.1; Exhibits B-1 – B-19 claims 22.3. 22.4, 23.1, 27.1, and 33.1; Exhibits D-1 – D-14, claims 1.4, 1.5, 1.6, 14.2, 19.2, 19.3, 21.4, 21.5, 21.6, 30.3, 30.4, 30.5, 35.1, 38.2, 39.4, 39.5, 39.7,

39.8, 41.1, and 44.2; and Exhibits E-1 – E-14 claims 1.4, 1.5, 1.6, 1.8, 1.10, 1.11, 6.2, and 7.1. Multiple protocol engines may improve interconnection controller performance by allowing multiple coherence transactions to be processed in parallel, thereby improving throughput. *See*, *e.g.*, *id.* The improved throughput allows a cluster corresponding to an interconnection controller to have more processors. *See*, *e.g.*, *id.*

a) Each protocol engine assigned a subset of global memory space

Some of the Asserted Claims are further directed to each protocol engine being assigned a subset of global memory space. *See*, *e.g.*, '206 claims 1.5, 1.6, 14.2, 19.2, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1; '254 claims 1.5, 1.6, 1.8, 1.10, 1.11, 6.2, and 7.1. At least under Memory Integrity's apparent infringement theories, assigning each protocol engine a subset of global memory space was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.5, 1.6, 14.2, 19.2, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1; Exhibits E-1 – E-14 claims 1.5, 1.6, 1.8, 1.10, 1.11, 6.2, and 7.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose an interconnection controller including a plurality of protocol engines for processing transactions in accordance with a cache coherence protocol, where each protocol engine of the plurality of protocol engines is assigned a subset of global memory space. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
    - Page 6 ("The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers."),

- Figure 1,
- Page 7 ("Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-bytewide buses, one on and one off, operating at one-third processor frequency."),
- Page 15 ("The unified second-level cache is shared across the two processors on the POWER4 chip. Figure 5 shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers."),
- Figure 5,
- Page 16 ("The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two processors (for either an L1 data-cache reload or an instruction fetch) or from one of the store queues."),
- Page 16 ("Included in each L2 controller are four snoop processors responsible for managing coherency operations snooped from the fabric."),
- Page 16 ("The L2 cache implements an enhanced version of the MESI coherency protocol . . .").

- *See*, *e.g.*, Behling et al., "The POWER4 Processor Introduction and Tuning Guide," at

  - Page 15 ("Stores can be sent to the L2 cache at a maximum rate of one store per cycle. Store data is directed to the proper L2 controller (through a hashing function) by way of the storage slice queue (SSQ) and the L2 store queue (STQ)."),
  - Page 17 ("Each POWER4 chip has an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree pseudo-LRU algorithm. The L2 cache is a unified cache: it caches instructions, data, and page table entries. The L2 cache is also shared by the processors on the chip."),
  - Page 18 ("Memory coherency in the system is enforced primarily at the L2 cache level by L2 cache controllers. Each L2 has associated command queues, known as coherency processors. Snoop processors within each controller observe all transactions in the system and respond accordingly, providing responses or delivering cache lines if the situation merits."),

- Page 30 ("The size of the L2 cache is 1440 KB per POWER4 chip, and this is shared between the two processors in the chip. As with the L1 data cache, the cache line size is 128 bytes. The replacement policy is pseudo-LRU (least recently used) so frequently accessed cache lines should be readily maintained in the cache. The L2 cache is a combined data and instruction cache. Instruction caching aspects of the L2 cache are not considered here. The L2 cache is divided into three equal parts, each under control of a separate L2 cache controller. The particular portion a line is stored is in is determined from the real memory address using a hashing algorithm. Sixteen consecutive double-precision Fortran array elements (138 bytes) are held in the same cache line, and therefore under control of the same cache controller. The 17th element will be in a different cache line and the hashing algorithm guarantees it will be stored under control of a different cache controller.").

- *See, e.g.*, IBM POWER4 processor (as disclosed at least in the above references)

- *See, e.g.*, Intel 870 Chipset [redacted]

- *See, e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108."),
  - Figure 1
  - [4:20-32] ("In one embodiment, the snoop filter 204 may be divided into four snoop filter (SF) interleaves 208. While in one embodiment the snoop filter 204 maintains coherency data for all lines of the caches 114, each SF interleave 208 may maintain coherency data for a unique subset of the cache lines. For example, two bits of a line's memory address may be used to identify which of the SF interleaves 208 maintains the coherency data for the particular line. Further, by assigning unique subsets to each of the SF interleaves 208, the SF interleaves 208 may operate in parallel and may increase the overall performance of the coherent switch 106 in comparison to a non-interleaved snoop filter 204 that may process a single request at time."),

- Figure 2
- [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

As acknowledged by the Applicant during prosecution of U.S. Patent No. 8,815,602, to which both the '206 and '254 patents claim priority, U.S. Patent Publication No. 2002/0007443 to Gharachorloo discloses a local protocol engine, which is assigned a subset of global memory space consisting of local memory addresses, and a remote protocol engine, which is assigned a subset of global memory space consisting of remote memory addresses. (U.S. Patent No. 8,815,602 Prosecution History, Applicant Response 2-13-2008 at Pages 8-9, Appeal Brief 10-3-2008 at Pages 7-8, 10.) For example, Applicant stated that "Gharachorloo clearly teaches that each multi-processor node includes only a *single* protocol engine for processing local memory transactions, i.e., the HPE, and a *single* protocol engine for processing remote memory transaction, i.e., the RPE." (U.S. Patent No. 8,815,602 Prosecution History, Applicant Response 2-13-2008 at Page 9, Appeal Brief 10-3-2008 at Page 8.) Furthermore, the Board of Patent Appeals and Interferences found that U.S. Patent No. 6,370,585 to Hagersten in view of Gharachorloo teaches or suggests a plurality of remote protocol engines for processing transactions targeting remote memory and a plurality of local protocol engines for processing transactions targeting local

memory. (U.S. Patent No. 8,815,602 Prosecution History, Decision on Appeal 10-27-2011 at Pages 7-9.)

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and Exhibits E-1 – E-14 so that each protocol engine of the plurality of protocol engines is assigned a subset of global memory space, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.5, 1.6, 14.2, 19.2, 19.2, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1; Exhibits E-1 – E1-4 claims 1.5, 1.6, 1.8, 1.10, 1.11, 6.2, and 7.1. Assigning subsets of global memory space to protocol engines may be simple to implement relative to other methods of assigning transactions to protocol engines. *See*, *e.g.*, *id.* For example, assigning addresses to protocol engines by interleaving is simple to implement and may provide good load balance across the protocol engines. *See*, *e.g.*, Nguyen, "High-Throughput Coherence Controllers" at Pages 29-30.

b) Remote protocol engine, local protocol engine

Some of the Asserted Claims are further directed to a remote protocol engine for processing transactions targeting remote memory and a local protocol engine for processing transactions targeting local memory. *See*, *e.g.*, '206 patent claims 1.5, 1.6, 14.2, 19.2, 19.3, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1, and 44.2; '254 patent claims 1.5-1.11, 6.2, and 7.1. At least under Memory Integrity's apparent infringement theories, a remote protocol engine for processing transactions targeting remote memory and a local protocol engine for processing transactions targeting local memory were well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.5, 1.6, 14.2, 19.2, 19.3, 21.5, 21.6,

30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1, and 44.2; Exhibits E-1 – E-14 claims 1.5-1.11, 6.2, and 7.1.

As acknowledged by the Applicant during prosecution of U.S. Patent No. 8,815,602, to which both the '206 and '254 patents claim priority, U.S. Patent Publication No. 2002/0007443 to Gharachorloo discloses a remote protocol engine for processing transactions targeting remote memory and a local protocol engine for processing transactions targeting local memory. *See, e.g.*,U.S. Patent No. 8,815,602 Prosecution History, Applicant Response 2-13-2008 at Pages 8-9, Appeal Brief 10-3-2008 at Pages 7-8, 10. For example, Applicant stated that "Gharachorloo clearly teaches that each multi-processor node includes only a *single* protocol engine for processing local memory transactions, i.e., the HPE, and a *single* protocol engine for processing remote memory transaction, i.e., the RPE." *See, e.g.*, U.S. Patent No. 8,815,602 Prosecution History, Applicant Response 2-13-2008 at Page 9, Appeal Brief 10-3-2008 at Page 8. Furthermore, the Board of Patent Appeals and Interferences found that U.S. Patent No. 6,370,585 to Hagersten in view of Gharachorloo teaches or suggests a plurality of remote protocol engines for processing transactions targeting remote memory and a plurality of local protocol engines for processing transactions targeting local memory. *See, e.g.*, U.S. Patent No. 8,815,602 Prosecution History, Decision on Appeal 10-27-2011 at Pages 7-9.

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and Exhibits E-1 – E-14 so that the plurality of protocol engines includes a remote protocol engine for processing transactions targeting remote memory and a local protocol engine for processing transactions targeting local memory, at least under Memory Integrity's apparent infringement

theories. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.5, 1.6, 14.2, 19.2, 19.3, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1, and 44.2; Exhibits E-1 – E-14 claims 1.5-1.11, 6.2, and 7.1. Assigning transactions targeting local versus remote memory to different protocol engines is beneficial when processing of transactions differs based on whether the transaction targets local or remote memory. *See*, *e.g.*, *id*. For example, if transactions targeting local memory require a directory look-up, but transactions targeting remote memory do not, then having a remote protocol engine allows transactions targeting remote memory to be processed faster, without having to wait for access to the directory. *See*, *e.g.*, Nguyen, "High-Throughput Coherence Controllers" at Pages 30-31. Separating local and remote protocol engines can also simplify protocol engine design and gives designers another dimension to vary the number of protocol engines by, for example, having a higher number of one type of protocol engine over the other. *See*, *e.g.*, *id*. at Pages 31-32, Pragaspathy et al, "Address Partitioning in DSM Clusters with Parallel Coherence Controllers" at 50 ("Because, only the home FSMs access the directory and only the remote FSMs access the remote cache, home-based partitioning reduces the sharing and contention in resources by a factor of two. As such, home-based partitioning reduces the hardware complexity of the resources (*e.g.*, by obviating the need for multiporting for two-engine designs) and the FSMs managing access to the resources, making this partitioning scheme the least expensive in hardware complexity and cost.").

c) Selecting or mapping a transaction to a protocol engine

Some of the Asserted Claims are further directed to selecting or mapping a transaction to a protocol engine using destination information associated with the transaction. *See*, *e.g.*, '206 claims 1.6, 2.1, 14.1, 19.2, 19.3, 21.6, 30.5, 34.1, 35.1, 39.7, 40.1, and 41.1; '254 claims 1.10, 1.11, 5.1, 6.1, and 6.2. At least under Memory Integrity's apparent infringement theories, selecting or mapping a transaction to a protocol engine using destination information associated with the

transaction was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.6, 2.1, 14.1, 19.2, 19.3, 21.6, 30.5, 34.1, 35.1, 39.7, 40.1, and 41.1; Exhibits E-1 – E-14 claims 1.10, 1.11, 5.1, 6.1, and 6.2. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose selecting or mapping a transaction to a protocol engine using destination information associated with the transaction. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
  - Page 6 ("The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers."),
  - Figure 1,
  - Page 7 ("Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-bytewide buses, one on and one off, operating at one-third processor frequency."),
  - Page 15 ("The unified second-level cache is shared across the two processors on the POWER4 chip. Figure 5 shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers."),
  - Figure 5,
  - Page 16 ("The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two processors (for either an L1 data-cache reload or an instruction fetch) or from one of the store queues."),

- Page 16 ("Included in each L2 controller are four snoop processors responsible for managing coherency operations snooped from the fabric."),
- Page 16 ("The L2 cache implements an enhanced version of the MESI coherency protocol . . .").

- *See*, *e.g.*, Behling et al., "The POWER4 Processor Introduction and Tuning Guide," at
  - Page 15 ("Stores can be sent to the L2 cache at a maximum rate of one store per cycle. Store data is directed to the proper L2 controller (through a hashing function) by way of the storage slice queue (SSQ) and the L2 store queue (STQ)."),
  - Page 17 ("Each POWER4 chip has an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree pseudo-LRU algorithm. The L2 cache is a unified cache: it caches instructions, data, and page table entries. The L2 cache is also shared by the processors on the chip."),
  - Page 18 ("Memory coherency in the system is enforced primarily at the L2 cache level by L2 cache controllers. Each L2 has associated command queues, known as coherency processors. Snoop processors within each controller observe all transactions in the system and respond accordingly, providing responses or delivering cache lines if the situation merits."),
  - Page 30 ("The size of the L2 cache is 1440 KB per POWER4 chip, and this is shared between the two processors in the chip. As with the L1 data cache, the cache line size is 128 bytes. The replacement policy is pseudo-LRU (least recently used) so frequently accessed cache lines should be readily maintained in the cache. The L2 cache is a combined data and instruction cache. Instruction caching aspects of the L2 cache are not considered here. The L2 cache is divided into three equal parts, each under control of a separate L2 cache controller. The particular portion a line is stored is in is determined from the real memory address using a hashing algorithm. Sixteen consecutive double-precision Fortran array elements (138 bytes) are held in the same cache line, and therefore under control of the same cache controller. The 17th element will be in a different cache line and the hashing algorithm guarantees it will be stored under control of a different cache controller.").
- *See*, *e.g.*, IBM POWER4 processor (as disclosed at least in the above references)
- *See, e.g.*, Intel 870 Chipset [redacted]

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at

- [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108."),
- Figure 1
- [4:20-32] ("In one embodiment, the snoop filter 204 may be divided into four snoop filter (SF) interleaves 208. While in one embodiment the snoop filter 204 maintains coherency data for all lines of the caches 114, each SF interleave 208 may maintain coherency data for a unique subset of the cache lines. For example, two bits of a line's memory address may be used to identify which of the SF interleaves 208 maintains the coherency data for the particular line. Further, by assigning unique subsets to each of the SF interleaves 208, the SF interleaves 208 may operate in parallel and may increase the overall performance of the coherent switch 106 in comparison to a non-interleaved snoop filter 204 that may process a single request at time."),
- Figure 2
- [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

Circuitry or other means for selecting or mapping a transaction to a protocol engine using destination information associated with the transaction may be in the interconnection controller or in a processing node, at least under Memory Integrity's apparent infringement theories. *See, e.g.*,

Exhibits D-1 – D-14, claims 1.6, 2.1, 14.1, 19.2, 19.3, 21.6, 30.5, 34.1, 35.1, 39.7, 40.1, and 41.1; Exhibits E-1 – E-14 claims 1.10, 1.11, 5.1, 6.1, and 6.2.

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include selecting or mapping a transaction to a protocol engine using destination information associated with the transaction, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.6, 2.1, 14.1, 19.2, 19.3, 21.6, 30.5, 34.1, 35.1, 39.7, 40.1, and 41.1; Exhibits E-1 – E-14 claims 1.10, 1.11, 5.1, 6.1, and 6.2. Destination information may include a target address associated with the transaction, in which case selecting or mapping a transaction to a protocol engine using destination information results in assigning subsets of global memory space to protocol engines. *See*, *e.g.*, '206 claims 2.1, 7.1, 14.1, 14.2, 19.3, 34.1, 35.1, 38.1, 38.2, 40.1, 41.1, 43.1, 44.1, 44.2; '254 claim 5.1. As discussed above, assigning subsets of global memory space to protocol engines may be simple to implement relative to other methods of assigning transactions to protocol engines. *See*, *e.g.*, Exhibits D-1 – D-14, claims 1.5, 1.6, 14.2, 19.2, 19.2, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1; Exhibits E-1 – E-14 claims 1.5, 1.6, 1.8, 1.10, 1.11, 6.1, 6.2, and 7.1. For example, interleaving – selecting a protocol engine based on destination information or specific bits of a target address corresponding to a transaction – is simple to implement and may provide good load balancing across the protocol engines. *See*, *e.g.*, Nguyen, "High-Throughput Coherence Controllers" at Pages 29-30.

d) Destination information comprises a target address associated with the transaction

Some of the Asserted Claims are further directed to the destination information comprising a target address associated with the transaction. *See*, *e.g.*, '206 claims 2.1, 7.1, 14.1, 14.2, 19.3,

34.1, 35.1, 38.1, 38.2, 40.1, 41.1, 43.1, 44.1, 44.2; '254 claim 5.1. At least under Memory Integrity's apparent infringement theories, destination information comprising a target address associated with the transaction was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 2.1, 7.1, 14.1, 14.2, 19.3, 34.1, 35.1, 38.1, 38.2, 40.1, 41.1, 43.1, 44.1, 44.2; Exhibits E-1 – E-14 claim 5.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose destination information comprising a target address associated with the transaction. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
    - Page 6 ("The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers."),
    - Figure 1,
    - Page 7 ("Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-bytewide buses, one on and one off, operating at one-third processor frequency."),
    - Page 15 ("The unified second-level cache is shared across the two processors on the POWER4 chip. Figure 5 shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers."),
    - Figure 5,
    - Page 16 ("The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two

[redacted]

- *See*, *e.g.*, Behling et al., "The POWER4 Processor Introduction and Tuning Guide," at
  - Page 15 ("Stores can be sent to the L2 cache at a maximum rate of one store per cycle. Store data is directed to the proper L2 controller (through a hashing function) by way of the storage slice queue (SSQ) and the L2 store queue (STQ)."),
  - Page 17 ("Each POWER4 chip has an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree pseudo-LRU algorithm. The L2 cache is a unified cache: it caches instructions, data, and page table entries. The L2 cache is also shared by the processors on the chip."),
  - Page 18 ("Memory coherency in the system is enforced primarily at the L2 cache level by L2 cache controllers. Each L2 has associated command queues, known as coherency processors. Snoop processors within each controller observe all transactions in the system and respond accordingly, providing responses or delivering cache lines if the situation merits."),
  - Page 30 ("The size of the L2 cache is 1440 KB per POWER4 chip, and this is shared between the two processors in the chip. As with the L1 data cache, the cache line size is 128 bytes. The replacement policy is pseudo-LRU (least recently used) so frequently accessed cache lines should be readily maintained in the cache. The L2 cache is a combined data and instruction cache. Instruction caching aspects of the L2 cache are not considered here. The L2 cache is divided into three equal parts, each under control of a separate L2 cache controller. The particular portion a line is stored is in is determined from the real memory address using a hashing algorithm. Sixteen consecutive double-precision Fortran array elements (138 bytes) are held in the same cache line, and therefore under control of the same cache controller. The 17th element will be in a different cache line and the hashing algorithm guarantees it will be stored under control of a different cache controller.").
- *See*, *e.g.*, IBM POWER4 processor (as disclosed at least in the above references)
- *See*, *e.g.*, Intel 870 Chipset [redacted]

[redacted]

[redacted]

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108."),
  - Figure 1
  - [4:20-32] ("In one embodiment, the snoop filter 204 may be divided into four snoop filter (SF) interleaves 208. While in one embodiment the snoop filter 204 maintains coherency data for all lines of the caches 114, each SF interleave 208 may maintain coherency data for a unique subset of the cache lines. For example, two bits of a line's memory address may be used to identify which of the SF interleaves 208 maintains the coherency data for the particular line. Further, by assigning unique subsets to each of the SF interleaves 208, the SF interleaves 208 may operate in parallel and may increase the overall performance of the coherent switch 106 in comparison to a non-interleaved snoop filter 204 that may process a single request at time."),
  - Figure 2
  - [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and Exhibits E-1 – E-14 to include destination information comprising a target address associated with the transaction, at least under Memory Integrity's apparent infringement theories. *See, e.g.*, Exhibits D-1 – D-14, claims 2.1, 7.1, 14.1, 14.2, 19.3, 34.1, 35.1, 38.1, 38.2, 40.1, 41.1, 43.1, 44.1, 44.2; Exhibits E-1 – E-14 claim 5.1. Using a target address associated with the transaction as the destination information used to select or map a transaction to a protocol engine results in assigning subsets of global memory space to protocol engines. As discussed above, assigning subsets of global memory space to protocol engines may be simple to implement relative to other methods of assigning transactions to protocol engines. *See, e.g.*, Exhibits D-1 – D-14, claims 1.5, 1.6, 14.2, 19.2, 19.2, 21.5, 21.6, 30.4, 30.5, 35.1, 38.2, 39.5, 39.7, 39.8, 41.1; Exhibits E-1 – E-14 claims 1.5, 1.6, 1.8, 1.10, 1.11, 6.1, 6.2, and 7.1. For example, interleaving – selecting a protocol engine based on specific bits of a target address, or destination information, corresponding to a transaction – is simple to implement and may provide good load balance across the protocol engines. *See, e.g.*, Nguyen, "High-Throughput Coherence Controllers" at Pages 29-30.

e) Packet associated with a transaction

Some of the Asserted Claims are further directed to associating a packet with a transaction. *See, e.g.*, '206 claims 2.1, 7.1, 19.1, 19.2, 34.1, 37.1, 40.1, 43.1; '254 claims 5.1, 6.2. At least under Memory Integrity's apparent infringement theories, associating a packet with a transaction was well-known in the art at the time of the alleged invention of the Asserted Claims. *See, e.g.*, Exhibits D-1 – D-14, claims 2.1, 7.1, 19.1, 19.2, 34.1, 37.1, 40.1, 43.1; Exhibits E-1 – E-14, 5.1, 6.2. At least under Memory Integrity's apparent infringement theories, there are many additional

exemplary prior art references that disclose associating a packet with a transaction. Some examples include:

- *See, e.g.*, Intel 870 Chipset [redacted]

- *See*, *e.g.*, Alan Charlesworth, "Starfire:Extending the SMP Envelope," IEEE Micro, January/February 1998, pp. 39-49 at
  - Page 40 ("Changed from a circuit-switched protocol to a packet-switched protocol. In a circuit-switched organization, each processor's bus request must complete before the next can begin. Packet switching separates the requests from the replies, letting bus transactions from several processors overlap.")
  - Page 40 ("All current Sun workstations and servers use Sun's Ultra Port Architecture.3 The UPA provides writeback MOESI (exclusive modified, shared modified, exclusive clean, shared clean, and invalid) coherency on 64-byte-wide cache blocks. The UPA uses packet-switched transactions with separate address and 18-byte-wide data lines, including two bytes of error-correcting code (ECC).")
  - Table 1
  - Pages 41-42 ("Table 3 characterizes the data interconnect. Data packets take four cycles. In the case of a load-miss, the missed-upon 16 bytes are sent first. The Starfire data buffer ASICs provide temporary storage for packets that are waiting their turn to be moved across the centerplane. The local and global routers are not buffered, and transfers take a fixed eight clocks from the data buffer on the sending board to the data buffer on the receiving board.")
  - Table 4
  - Page 45 ("In addition to the ECC for data that is generated and checked by the processor module, Starfire ASICs also generate and check ECC for address packets. To help isolate faults, the Starfire data-buffer chips check data-packet ECC along the way through the interconnect.")
- *See*, *e.g.*, Alan Charlesworth, Nicholas Aneshansley, Mark Haakmeester, Dan Drogichen, Gary Gilbert, Ricki Williams, Andrew Phelps, "The Starfire SMP Interconnect" Proceedings of the 1997 ACM/IEEE conference on Supercomputing, November 1997, pp. 1-20

  - Page 3 ("All Sun workstations and servers use the same Ultra Port Architecture (UPA) [7] interconnect protocol. This is Sun's third-generation SMP interconnect architecture, following the second-generation XDBus and the first-generation MBus. The UPA does writeback MOESI (exclusively modified, shared modified, exclusively clean, shared clean, and invalid) coherency on 64-byte-wide cache blocks. It is a packet-switched protocol, with separate address and 18-byte-wide data lines, which includes two bytes of error-correcting code (ECC). The highest system clock rate so far is 100 MHz, which yields a peak UPA_databus rate of 1,600 MBps.")
  - Page 6 ("Data packets take four cycles on the data interconnect.")
  - Page 11 ("In addition to the ECC for data that is generated and checked by the processor module, the Starfire ASICs also generate and check ECC for address packets. The Starfire Data Buffer chips check data-packet ECC along the way through the interconnect to help isolate faults.")
- *See*, *e.g.*, Alan Charlesworth, Andy Phelps, Ricki Williams, Gary Gilbert "Gigaplane-XB: Extending the Ultra Enterprise Family" July 30, 1997, pp. 1-16
  - Page 2 ("To allow scaling beyond traditional snoopy buses, the UPA defines a point-to-point packet-switched protocol between the distributed system control function.")
  - Page 6 ("Address packets take two cycles to convey address information over one of the address buses. As explained above, the address buses act logically as buses, but are physically implemented with point-to-point connections and broadcast router ASICs.")
- *See*, *e.g.*, Alan Charlesworth, "The Sun Fireplane Interconnect in the Mid- Range Sun Fire Servers," Presentation at Symposium on High Performance Interconnects (Hot Interconnects), August 22-24, 2001 at Slide 5 (disclosing XDBus, UPA, Fireplane are packet switched
- *See*, *e.g.*, Alan Charlesworth, "The Sun Fireplane System Interconnect," Proceedings of the 2001 ACM/IEEE conference on Supercomputing, November 2001 at Table 1, Section 7
- *See*, *e.g.*, Alan Charlesworth, "SMP Interconnect @ Sun," SUPerG conference, Spring 2002 at
  - Page 1 ("This dimension is called horizontal scaling. Multiple computers are connected together by a network switch that routes network packets between the computers. The cluster interconnect is usually based on open-standard technology. These systems are more cost and power efficient, because their interconnect is simpler than in large SMPs, and they are built in higher volumes. Each computer runs its own instance of the operating system, and the whole ensemble is orchestrated by cluster-management software.")

- Table 1
- Page 6 ("The protocol is improved so that address and data packets need only to traverse the shortest path between source and destination. In the previous generation, packets always had to go all the way to the outermost level of the interconnect, even if they were going to a destination on the same board. This enhancement lowers the latency for "close" transfers.")

- *See*, *e.g.*, Sun architectures, including Sun XDBus, UltraPort Architecture (UPA), and Fireplane, as disclosed in the above references

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include associating a packet with a transaction, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits D-1 – D-14, claims 2.1, 7.1, 19.1, 19.2, 34.1, 37.1, 40.1, 43.1; Exhibits D-1 – D-14, 5.1, 6.2. For example, packet-based transmissions for communications between, for example, processors and cache coherence controllers, allow multiple transactions to overlap. *See*, *e.g.*, Alan Charlesworth, "Starfire: Extending the SMP Envelope," IEEE Micro, January/February 1998 at page 40.

f) Target address corresponds to or is in a packet

Some of the Asserted Claims are further directed to a target address that corresponds to or is in a packet. *See*, *e.g.*, '206 claims 2.1, 7.1, 34.1, 37.1, 40.1, 43.1; '254 claim 5.1. At least under Memory Integrity's apparent infringement theories, a target address that corresponds to or is in a packet was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 2.1, 7.1, 34.1, 37.1, 40.1, 43.1; Exhibits E-1 – E-14 claim 5.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a target address that corresponds to or is in a packet. Some examples include:

- *See, e.g.*, Intel 870 Chipset [redacted]

- *See, e.g.*, Alan Charlesworth, "Starfire:Extending the SMP Envelope," IEEE Micro, January/February 1998, pp. 39-49 at
  - Page 40 ("Changed from a circuit-switched protocol to a packet-switched protocol. In a circuit-switched organization, each processor's bus request must complete before the next can begin. Packet switching separates the requests from the replies, letting bus transactions from several processors overlap.")
  - Page 40 ("All current Sun workstations and servers use Sun's Ultra Port Architecture.3 The UPA provides writeback MOESI (exclusive modified, shared modified, exclusive clean, shared clean, and invalid) coherency on 64-byte-wide cache blocks. The UPA uses packet-switched transactions with separate address and 18-byte-wide data lines, including two bytes of error-correcting code (ECC).")
  - Table 1
  - Pages 41-42 ("Table 3 characterizes the data interconnect. Data packets take four cycles. In the case of a load-miss, the missed-upon 16 bytes are sent first. The Starfire data buffer ASICs provide temporary storage for packets that are waiting their turn to be moved across the centerplane. The local and global routers are not buffered, and transfers take a fixed eight clocks from the data buffer on the sending board to the data buffer on the receiving board.")
  - Table 4
  - Page 45 ("In addition to the ECC for data that is generated and checked by the processor module, Starfire ASICs also generate and check ECC for address packets. To help isolate faults, the Starfire data-buffer chips check data-packet ECC along the way through the interconnect.")
- *See, e.g.*, Alan Charlesworth, Nicholas Aneshansley, Mark Haakmeester, Dan Drogichen, Gary Gilbert, Ricki Williams, Andrew Phelps, "The Starfire SMP Interconnect" Proceedings of the 1997 ACM/IEEE conference on Supercomputing, November 1997, pp. 1-20
  - Page 3 ("All Sun workstations and servers use the same Ultra Port Architecture (UPA) [7] interconnect protocol. This is Sun's third-generation SMP interconnect architecture, following the second-generation XDBus and the first-generation

MBus. The UPA does writeback MOESI (exclusively modified, shared modified, exclusively clean, shared clean, and invalid) coherency on 64-byte-wide cache blocks. It is a packet-switched protocol, with separate address and 18-byte-wide data lines, which includes two bytes of error-correcting code (ECC). The highest system clock rate so far is 100 MHz, which yields a peak UPA_databus rate of 1,600 MBps.")

- Page 6 ("Data packets take four cycles on the data interconnect.")
- Page 11 ("In addition to the ECC for data that is generated and checked by the processor module, the Starfire ASICs also generate and check ECC for address packets. The Starfire Data Buffer chips check data-packet ECC along the way through the interconnect to help isolate faults.")

- *See, e.g.*, Alan Charlesworth, Andy Phelps, Ricki Williams, Gary Gilbert "Gigaplane-XB: Extending the Ultra Enterprise Family" July 30, 1997, pp. 1-16

  - Page 2 ("To allow scaling beyond traditional snoopy buses, the UPA defines a point-to-point packet-switched protocol between the distributed system control function.")
  - Page 6 ("Address packets take two cycles to convey address information over one of the address buses. As explained above, the address buses act logically as buses, but are physically implemented with point-to-point connections and broadcast router ASICs.")

- *See, e.g.*, Alan Charlesworth, "The Sun Fireplane Interconnect in the Mid- Range Sun Fire Servers," Presentation at Symposium on High Performance Interconnects (Hot Interconnects), August 22-24, 2001 at Slide 5 (disclosing XDBus, UPA, Fireplane are packet switched)

- *See, e.g.*, Alan Charlesworth, "The Sun Fireplane System Interconnect," Proceedings of the 2001 ACM/IEEE conference on Supercomputing, November 2001 at Table 1, Section 7

- *See, e.g.*, Alan Charlesworth, "SMP Interconnect @ Sun," SUPerG conference, Spring 2002 at

  - Page 1 ("This dimension is called horizontal scaling. Multiple computers are connected together by a network switch that routes network packets between the computers. The cluster interconnect is usually based on open-standard technology. These systems are more cost and power efficient, because their interconnect is simpler than in large SMPs, and they are built in higher volumes. Each computer runs its own instance of the operating system, and the whole ensemble is orchestrated by cluster-management software.")
  - Table 1
  - Page 6 ("The protocol is improved so that address and data packets need only to traverse the shortest path between source and destination. In the previous

generation, packets always had to go all the way to the outermost level of the interconnect, even if they were going to a destination on the same board. This enhancement lowers the latency for "close" transfers.")

- *See, e.g.*, Sun architectures, including Sun XDBus, UltraPort Architecture (UPA), and Fireplane, as disclosed in the above references

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a target address that corresponds to or is in a packet, at least under Memory Integrity's apparent infringement theories. *See, e.g.*, Exhibits D-1 – D-14, claims 2.1, 7.1, 34.1, 37.1, 40.1, 43.1; Exhibits E-1 – E-14 claim 5.1. By placing a target address in a packet or having a target address correspond to a packet, circuitry may use the target address to determine which protocol engine has been assigned the target address.

g) Distributing transactions over protocol engines by interleaving target addresses, selecting protocol engines with reference to at least one bit in a target address, and assigning mutually exclusive subsets of memory space to protocol engines

Some of the Asserted Claims are further directed to distributing transactions over protocol engines by interleaving target addresses. *See, e.g.*, '206 claims 14.1, 38.2, 44.2. Interleaving assigns an address to a protocol engine based on the values of specific bits of a target address. Interleaving thus allows protocol engines to be selected with reference to at least one bit in a target address, as recited in some Asserted Claims. *See, e.g.*, '206 claims 35.1, 41.1; '254 claim 6.2. Interleaving results in the subsets of memory space assigned to protocol engines being mutually exclusive, as recited in some Asserted Claims. *See, e.g.*, '254 claim 8.1. At least under Memory Integrity's apparent infringement theories, distributing transactions over protocol engines by interleaving target addresses, selecting protocol engines with reference to at least one bit in a target address, and assigning mutually exclusive subsets of memory space to protocol engines were

well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 14.1, 35.1, 38.1, 41.1, 44.2; Exhibits E-1 – E-14 claims 6.2, 8.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose distributing transactions over protocol engines by interleaving target addresses, selecting protocol engines with reference to at least one bit in a target address, and assigning mutually exclusive subsets of memory space to protocol engines. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
  - Page 6 ("The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers."),
  - Figure 1,
  - Page 7 ("Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-bytewide buses, one on and one off, operating at one-third processor frequency."),
  - Page 15 ("The unified second-level cache is shared across the two processors on the POWER4 chip. Figure 5 shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers."),
  - Figure 5,
  - Page 16 ("The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two processors (for either an L1 data-cache reload or an instruction fetch) or from one of the store queues."),

- Page 16 ("Included in each L2 controller are four snoop processors responsible for managing coherency operations snooped from the fabric."),
- Page 16 ("The L2 cache implements an enhanced version of the MESI coherency protocol . . .").

- *See*, *e.g.*, Behling et al., "The POWER4 Processor Introduction and Tuning Guide," at
  - Page 15 ("Stores can be sent to the L2 cache at a maximum rate of one store per cycle. Store data is directed to the proper L2 controller (through a hashing function) by way of the storage slice queue (SSQ) and the L2 store queue (STQ)."),
  - Page 17 ("Each POWER4 chip has an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree pseudo-LRU algorithm. The L2 cache is a unified cache: it caches instructions, data, and page table entries. The L2 cache is also shared by the processors on the chip."),
  - Page 18 ("Memory coherency in the system is enforced primarily at the L2 cache level by L2 cache controllers. Each L2 has associated command queues, known as coherency processors. Snoop processors within each controller observe all transactions in the system and respond accordingly, providing responses or delivering cache lines if the situation merits."),
  - Page 30 ("The size of the L2 cache is 1440 KB per POWER4 chip, and this is shared between the two processors in the chip. As with the L1 data cache, the cache line size is 128 bytes. The replacement policy is pseudo-LRU (least recently used) so frequently accessed cache lines should be readily maintained in the cache. The L2 cache is a combined data and instruction cache. Instruction caching aspects of the L2 cache are not considered here. The L2 cache is divided into three equal parts, each under control of a separate L2 cache controller. The particular portion a line is stored is in is determined from the real memory address using a hashing algorithm. Sixteen consecutive double-precision Fortran array elements (138 bytes) are held in the same cache line, and therefore under control of the same cache controller. The 17th element will be in a different cache line and the hashing algorithm guarantees it will be stored under control of a different cache controller.").
- *See*, *e.g.*, IBM POWER4 processor (as disclosed at least in the above references)
- *See*, *e.g.*, Intel 870 Chipset [redacted]

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at

- [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108."),
- Figure 1
- [4:20-32] ("In one embodiment, the snoop filter 204 may be divided into four snoop filter (SF) interleaves 208. While in one embodiment the snoop filter 204 maintains coherency data for all lines of the caches 114, each SF interleave 208 may maintain coherency data for a unique subset of the cache lines. For example, two bits of a line's memory address may be used to identify which of the SF interleaves 208 maintains the coherency data for the particular line. Further, by assigning unique subsets to each of the SF interleaves 208, the SF interleaves 208 may operate in parallel and may increase the overall performance of the coherent switch 106 in comparison to a non-interleaved snoop filter 204 that may process a single request at time."),
- Figure 2
- [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include distributing transactions over protocol engines by interleaving target

addresses, selecting protocol engines with reference to at least one bit in a target address, and assigning mutually exclusive subsets of memory space to protocol engines, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits D-1 – D-14, claims 14.1, 35.1, 38.1, 41.1, 44.2; Exhibits E-1 – E-14 claims 6.2, 8.1. For example, methods for selecting protocol engines with reference to at least one bit in a target address, such as interleaving – selecting a protocol engine based on specific bits of a target address, or destination information, corresponding to a transaction – are simple to implement and may provide good load balance across the protocol engines. *See*, *e.g.*, Nguyen, "High-Throughput Coherence Controllers" at Pages 29-30. Assigning mutually exclusive subsets of memory space to protocol engines simplifies maintenance of cache coherency, as each protocol engine has exclusive access to a set of addresses. *See*, *e.g.*, Nguyen, "High-Throughput Coherence Controllers" at Pages 29-30, Page 30 n.1.

h) A packet corresponding to a transaction includes a node identifier corresponding to a protocol engine, where the transaction is mapped to the protocol engine by a node generating the node identifier with reference to a target address associated with the transaction

Some of the Asserted Claims are further directed to a packet corresponding to a transaction that includes a node identifier corresponding to a protocol engine, where the transaction is mapped to the protocol engine by a node generating the node identifier with reference to a target address associated with the transaction. *See*, *e.g.*, '206 claims 19.2, 19.3. At least under Memory Integrity's apparent infringement theories, a packet corresponding to a transaction that includes a node identifier corresponding to a protocol engine, where the transaction is mapped to the protocol engine by a node generating the node identifier with reference to a target address associated with the transaction, was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits D-1 – D-14, claims 19.2, 19.3. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that

disclose a packet corresponding to a transaction that includes a node identifier corresponding to a protocol engine, where the transaction is mapped to the protocol engine by a node generating the node identifier with reference to a target address associated with the transaction. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
  - Page 6 ("The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers."),
  - Figure 1,
  - Page 7 ("Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-bytewide buses, one on and one off, operating at one-third processor frequency."),
  - Page 15 ("The unified second-level cache is shared across the two processors on the POWER4 chip. Figure 5 shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers."),
  - Figure 5,
  - Page 16 ("The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two processors (for either an L1 data-cache reload or an instruction fetch) or from one of the store queues."),
  - Page 16 ("Included in each L2 controller are four snoop processors responsible for managing coherency operations snooped from the fabric."),
  - Page 16 ("The L2 cache implements an enhanced version of the MESI coherency protocol . . .").

- *See*, *e.g.*, Behling et al., "The POWER4 Processor Introduction and Tuning Guide," at
  - Page 15 ("Stores can be sent to the L2 cache at a maximum rate of one store per cycle. Store data is directed to the proper L2 controller (through a hashing function) by way of the storage slice queue (SSQ) and the L2 store queue (STQ)."),
  - Page 17 ("Each POWER4 chip has an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree pseudo-LRU algorithm. The L2 cache is a unified cache: it caches instructions, data, and page table entries. The L2 cache is also shared by the processors on the chip."),
  - Page 18 ("Memory coherency in the system is enforced primarily at the L2 cache level by L2 cache controllers. Each L2 has associated command queues, known as coherency processors. Snoop processors within each controller observe all transactions in the system and respond accordingly, providing responses or delivering cache lines if the situation merits."),
  - Page 30 ("The size of the L2 cache is 1440 KB per POWER4 chip, and this is shared between the two processors in the chip. As with the L1 data cache, the cache line size is 128 bytes. The replacement policy is pseudo-LRU (least recently used) so frequently accessed cache lines should be readily maintained in the cache. The L2 cache is a combined data and instruction cache. Instruction caching aspects of the L2 cache are not considered here. The L2 cache is divided into three equal parts, each under control of a separate L2 cache controller. The particular portion a line is stored is in is determined from the real memory address using a hashing algorithm. Sixteen consecutive double-precision Fortran array elements (138 bytes) are held in the same cache line, and therefore under control of the same cache controller. The 17th element will be in a different cache line and the hashing algorithm guarantees it will be stored under control of a different cache controller.").
- *See*, *e.g.*, IBM POWER4 processor (as disclosed at least in the above references)
- *See*, *e.g.*, Intel 870 Chipset [redacted]

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102

may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108."),

- Figure 1
- [4:20-32] ("In one embodiment, the snoop filter 204 may be divided into four snoop filter (SF) interleaves 208. While in one embodiment the snoop filter 204 maintains coherency data for all lines of the caches 114, each SF interleave 208 may maintain coherency data for a unique subset of the cache lines. For example, two bits of a line's memory address may be used to identify which of the SF interleaves 208 maintains the coherency data for the particular line. Further, by assigning unique subsets to each of the SF interleaves 208, the SF interleaves 208 may operate in parallel and may increase the overall performance of the coherent switch 106 in comparison to a non-interleaved snoop filter 204 that may process a single request at time."),
- Figure 2
- [4:61-5:13] ("The CPL 212 may provide global functions for processing transactions regardless of which port 200 the transaction originated. For example, the CPL 212 for each port 200 may check for-transaction conflicts, may prevent transaction starvation, and may maintain data coherency in accordance with a snoop-based protocol. In particular, the CPL 212 in response to processing a read, snoop, or invalidate request may check the state of the line of the request in the caches 114 and may issue requests to one or more cache nodes 102, 104 based upon the state of the line. In one embodiment, the CPL 212 may use the coherency data of the snoop filter 204 to reduce the number of requests sent to the cache nodes 102, 104 and may update coherency data in the snoop filter 204 based upon the transaction type and snoop responses received from the cache nodes 102, 104. The CPL 212 may further comprise logic to bypass the snoop filter 204 and to maintain data coherency without using the coherency data of the snoop filter 204. Moreover, the CPL 212 may be divided into four interleaves 214, and a separate CPL interleave 214 may be associated with each of the four SF interleaves 208.").

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a packet corresponding to a transaction that includes a node identifier corresponding to a protocol engine, where the transaction is mapped to the protocol engine by a node generating the node identifier with reference to a target address associated with the

transaction, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits D-1 – D-14, claims 19.2, 19.3. For example, it is well-known that a packet may include destination information such as a node identifier to allow the packet to be routed to the node identified by the node identifier. A transaction may thus be mapped to a protocol engine by generating the node identifier for the protocol engine based on whether the target address corresponding to the packet is assigned to that protocol engine, for example, by generating the node identifier for a remote protocol engine if the target address is in remote memory and generating the node identifier for a local protocol engine if the target address is in local memory.

13. *Protocol engine configured to process interrupts*

Some of the Asserted Claims are directed to a protocol engine configured to process interrupts. *See*, *e.g.*, '254 claim 7.1. At least under Memory Integrity's apparent infringement theories, a protocol engine configured to process interrupts was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, E-1 – E-14, claim 7.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a protocol engine configured to process interrupts. Some examples include:

- *See*, *e.g.*, Tendler et al., "POWER4 system microarchitecture," at
    - Page 15 ("The L1 caches are parity-protected. A parity error detected in the L1 instruction cache forces the line to be invalidated and reloaded from the L2. Errors encountered in the L1 data cache are reported as a synchronous machine-check interrupt. To support error recovery, the machine-check interrupt handler is implemented in system-specific firmware code. When the interrupt occurs, the firmware saves the processor-architected states and examines the processor registers to determine the recovery and error status. If the interrupt is recoverable, the system firmware removes the error by invalidating the L1 data-cache line and incrementing an error counter. If the L1 data-cache error counter is greater than a predefined threshold, which is an indication of a solid error, the system firmware disables the failing portion of the L1 data cache. The system firmware then restores the processor-architected states and "calls back" the operating system machine-check handler with the "fully recovered" status. The operating system

checks the return status from firmware and resume execution. With the L1 data-cache line invalidated, data is now reloaded from the L2. All data stored in the L1 data cache is available in the L2 cache, guaranteeing no data loss.")

- *See, e.g.*, Intel 870 Chipset [redacted]
- *See, e.g.*, Intel 870 Chipset [redacted]

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a protocol engine configured to process interrupts, at least under Memory Integrity's apparent infringement theories. *See, e.g.*, E-1 – E-14 claim 7.1. For example, interrupts are a well-known non-coherent transaction that traverse through interconnection controllers in multiprocessor systems.

14. *Integrated circuit, computer-readable medium, and semiconductor processing masks*

Some of the Asserted Claims are directed to:

- an integrated circuit comprising a probe filtering unit or interconnection controller (*see*, *e.g.*, '121 claim 17.1; '206 claim 22.1),
- non-transitory computer-readable medium having data structures stored therein representative of a probe filtering unit or interconnection controller (*see*, *e.g.*, '121 claim 19.1; '206 claim 24.1),
- the data structures comprising a simulatable representation of a probe filtering unit or interconnection controller (*see*, *e.g.*, '121 claim 20.1; '206 claim 25.1),
- the simulatable representation comprising a netlist (see, e.g.,'121 claim 21.1; '206 claim 26.1),
- the data structures comprising a code description of a probe filtering unit or interconnection controller (*see*, *e.g.*, '121 claim 22.1; '206 claim 27.1),
- the code description corresponding to a hardware description language (*see*, *e.g.*, '121 claim 23.1; '206 claim 28.1), and
- a set of semiconductor processing masks representative of at least a portion of a probe filtering unit or interconnection controller (*see*, *e.g.*, '121 claim 24.1; '206 claim 29.1).

At least under Memory Integrity's apparent infringement theories, all of the above were well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claims 17.1, 19.1, 20.1, 21.1, 22.1, 23.1.24.1; Exhibits D-1 – D-14, claims 22.1, 24.1, 25.1, 26.1, 27.1, 28.1, 29.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose the above limitations. Some examples include:

- *See*, *e.g.*, Michael John Sebastian Smith, "Application-Specific Integrated Circuits," 1997 at
  - Page 1 ("Figure 1.1 (a) shows an Ie package (this is a pin-grid array, or PGA, shown upside down; the pins will go through holes in a printed-circuit board). People often call the package a chip, but, as you can see in Figure 1.1 (b), the silicon chip itself (more properly called a die) is mounted in the cavity under the sealed lid. A PGA package is usually made from a ceramic material, but plastic packages are also common.")
  - Figure 1.1
  - Page 3 ("With the advent of VLSI in the 1980s engineers began to realize the advantages of designing an IC that was customized or tailored to a particular

system or application rather than using standard ICs alone. Microelectronic system design then becomes a matter of defining the functions that you can implement using standard ICs and then implementing the remaining logic functions (sometimes called glue logic) with one or more custom ICs. As VLSI became possible you could build a system from a smaller number of components by combining many standard ICs into a few custom ICs. Building a microelectronic system with fewer ICs allows you to reduce cost and improve reliability.")

- Page 4 ("ICs are made on a thin (a few hundred microns thick), circular silicon wafer, with each wafer holding hundreds of die (sometimes people use dies or dice for the plural of die). The transistors and wiring are made from many layers (usually between 10 and 15 distinct layers) built on top of one another. Each successive mask layer has a pattern that is defined using a mask similar to a glass photographic slide. The first half-dozen or so layers define the transistors. The last half-dozen or so layers define the metal wires between the transistors (the interconnect). A full-custom IC includes some (possibly all) logic cells that are customized and all mask layers that are customized.")
- Pages 16-18 (Section 1.2 Design Flow)
- Page 28 ("If you complete an ASIC design using a cell library that you bought, you also own the masks (the tooling) that are used to manufacture your ASIC.")
- Pages 49-55 (Section 2.2 The CMOS Process)

- *See*, *e.g.*, J. D. Warnock, J. M. Keaty, J. Petrovick, J. G. Clabes, C. J. Kircher, B. L. Krauter, P. J. Restle, B. A. Zoric, C. J. Anderson, "The circuit and physical design of the POWER4 microprocessor," IBM Journal of Research and Development, Vol. 46 No. 1, January 2002, pp. 27-51 at

  - Page 27 ("The IBM POWER4 processor is a 174-million transistor chip that runs at a clock frequency of greater than 1.3 GHz. It contains two microprocessor cores, high-speed buses, and an on-chip memory subsystem. The complexity and size of POWER4, together with its high operating frequency, presented a number of significant challenges for its multisite design team. This paper describes the circuit and physical design of POWER4 and gives results that were achieved. Emphasis is placed on aspects of the design methodology, clock distribution, circuits, power, integration, and timing that enabled the design team to meet the project goals and to complete the design on schedule.")
  - Page 27 ("The POWER4 chip provides the processing power for eServer p690, the recently introduced high-end, IBM 64-bit POWER-architecture, 8-to-32-way server system [1]. The chip, shown in Figure 1, includes two microprocessors, 1.44 MB of shared L2 cache memory plus the directory for a 32MB off-chip cache, a 500-MHz interconnection fabric, high-bandwidth buses and I/O designed to allow building an eight-way system on a single multi-chip module, and the logic needed to support large SMPs [2]. The microprocessor core is an out-of-order, speculative, eight-issue superscalar design containing eight execution units, a 64KB L1 instruction cache, and a 32KB, dual-ported data cache [1, 3]. The POWER4 chips

were fabricated in the state-of-the-art IBM 0.18-m CMOS 8S3 SOI (silicon-on-insulator) technology with seven levels of copper wiring [4]. Some of the features of this technology are given in Table 1. Characteristics of the POWER4 chip fabricated in this technology are given in Table 2. Using these chips, a 32-way SMP system has been operated in our laboratory at clock frequencies exceeding 1.3 GHz. Work is in progress to release the POWER4 design in CMOS 9S technology, which will significantly reduce the chip area as well as improve performance and decrease power dissipation.")

- Figure 1 POWER4 chip photograph showing the principal functional units in the microprocessor core and in the memory subsystem.
- Table 1 Features of the IBM CMOS 8S3 SOI technology.
- Table 2 Characteristics of the POWER4 chip fabricated in CMOS 8S3 SOI.
- Pages 29-30 (Design phases)
- Pages 30-31 (Design flow and tools)
- Figure 3 Design flow used during high-level design. The rectangular shapes represent tools used to complete a portion of the design. The cylinders represent design data, and the circle the verification of the VHDL logic.
- Figure 4 Design flow used during schematic design. The rectangular shapes represent tools used to complete a portion of the design. The cylinders represent design data and the circle the verification of the VHDL logic.
- Figure 5 Design flow used during physical design. The rectangular shapes represent tools used to complete a portion of the design. The cylinders represent design data and the circle the verification of the VHDL logic.

- Jeffrey Kuskin, David Ofelt, Mark Heinrich, John Heinlein, Richard Simoni, Kourosh Gharachorloo, John Chapin, David Nakahira, Joel Baxter, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John Hennessy, "The Stanford FLASH Multiprocessor," Proceedings of the 21st International Symposium on Computer Architecture, 1994, pp. 302-313 at

  - Page 302 ("The FLASH multiprocessor efficiently integrates support for cache-coherent shared memory and high-performance message passing, while minimizing both hardware and software overhead. Each node in FLASH contains a microprocessor, a portion of the machine's global memory, a port to the interconnection network, an U0 interface, and a custom node controller called MAGIC. The MAGIC chip handles all communication both within the node and among nodes, using hardwired data paths for efficient data movement and a programmable processor optimized for executing protocol operations. The use of the protocol processor makes FLASH very flexible - it can support a variety of different communication mechanisms - and simplifies the design and implementation. This paper presents the architecture of FLASH and MAGIC, and discusses the base cache-coherence and message-passing protocols. Latency and

occupancy numbers, which are derived from our system-level simulator and our Verilog code, are given for several common protocol operations. The paper also describes our software strategy and FLASH'S current status.")

- Pages 302-303 ("To accomplish these goals we are designing a custom node controller. This controller, called MAGIC (Memory And General Interconnect Controller), is a highly integrated chip that implements all data transfers both within the node and between the node and the network. To deliver high performance, the MAGIC chip contains a specialized data path optimized to move data between the memory, network, processor, and I/O ports in a pipelined fashion without redundant copying. To provide the flexible control needed to support a variety of DSM and message-passing protocols, the MAGIC chip contains an embedded processor that controls the data path and implements the protocol. The separate data path allows the processor to update the protocol data structures (e.g., the directory for cache coherence) in parallel with the associated data transfers.")
- Page 303 ("First, MAGIC includes a programmable protocol processor for flexibility.")
- Page 309 ("To demonstrate that MAGIC can achieve competitive performance, we present the latency for servicing a processor read miss to local memory. Table 4.1 lists the latency through each stage of the data transfer logic and control macropipeline for this operation, assuming that the MAGIC chip was initially idle. The cycle counts are based on a 100 MHz (10 ns) MAGIC clock rate and are derived from the current Verilog models of the various units.")
- Page 311 ("FLASH will use the MIPS TS, a follow-on to the R4000, as its primary processor. Like the R4000, the TS manages its own second-level cache. The target speed of the node board and the MAGIC chip is 100 MHz. The multiply-banked memory system is designed to match the node's bandwidth requirements and is optimized for 128-byte transfers, the system cache line size. FLASH will implement the PCI standard bus for its I/O subsystem and will use next-generation Intel routers for the interconnection network. 'The initial FLASH prototype will contain 256 processing nodes. We plan to collaborate with the Intel Corporation and Silicon Graphics on the design and construction of the prototype machine. We currently have a detailed system-level simulator up and running. The simulator is written in C++ as a multithreaded memory simulator for Tango-Lite [Golds93]. The entire system, called FlashLite, runs real applications and enables us to verify protocols, analyze system performance, and identify architectural bottlenecks. We have coded the entire base cache-coherence and base block-transfer protocols for FlashLite, and have run complete simulations of several SPLASH applications. The FlashLite code is structured identically to the actual hardware, with each hardware block corresponding to a FlashLite thread. To aid the debugging of protocols implemented in FlashLite we have developed a random test case generator, the FLASH Protocol Verifier. On the hardware design front we are busily coding the Verilog description of the MAGIC chip. To verify our hardware description we plan to have FlashLite provide test vectors for a Verilog run, and to run real N processor applications with N-1 FlashLite nodes and one Verilog node. Since the

> FlashLite code is structured like the Verilog description, we also plan to replace a single FlashLite thread with the appropriate hardware block description to allow more efficient and accurate verification. Software tools for the protocol processor are another major effort. We are porting the GNU C compiler [Stall93] to generate code for the 64-bit PP. We have also ported a superscalar instruction scheduler and an assembler from the Torch project [SHL92]. Finally, we have a PP instruction set emulator ported from Mable. This emulator will help us verify the actual PP code sequences by becoming the PP thread in FlashLite simulations. Operating system development is proceeding concurrently with the hardware design. Hive's implementation is based on IRIX (UNIX SVR4 from Silicon Graphics), with extensive modifications in progress to the virtual memory, I/O, and process management subsystems.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include the above limitations, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claims 17.1, 19.1, 20.1, 21.1, 22.1, 23.1, 24.1; D-1 – D-14, claims 22.1, 24.1, 25.1, 26.1, 27.1, 28.1, 29.1. It was well-known and routine at the time of the alleged invention of the Asserted Claims for components that communicate with processors, such as a probe filtering unit or interconnection controller, to be an integrated circuit, represented in a set of semiconductor processing masks for manufacturing an integrated circuit, or represented by data structures stored on a non-transitory computer readable medium, where the data structures are a code description, such as a hardware description language, or used to simulate the component, such as a netlist. For example, an integrated circuit was the most common implementation of a computer component, and integrated circuits were known to have been created with a set of semiconductor processing masks. Also, simulations were commonly performed for purposes of testing performance, and the design of a component was routinely described in code by, for example, a hardware description language.

15. *Probe filtering unit corresponds to an additional node interconnected with the processing nodes, additional node comprises a cache coherence controller, cache coherence controller comprises the probe filtering unit*

Some of the Asserted Claims are directed to a probe filtering unit that corresponds to an additional node interconnected with the processing nodes (*see, e.g.*, '121 claim 2.1), where the additional node comprises a cache coherence controller (*see, e.g.*, '121 claim 3.1). Some of the Asserted Claims are directed to a cache coherence controller that comprises the probe filtering unit. *See, e.g.*, '121 claim 6.1. At least under Memory Integrity's apparent infringement theories, a probe filtering unit that corresponds to an additional node interconnected with the processing nodes, where the additional node comprises a cache coherence controller, or a cache coherence controller comprising the probe filtering unit was well-known in the art at the time of the alleged invention of the Asserted Claims. *See, e.g.*, Exhibits C-1 – C-8 claims 2.1, 3.1, 6.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a probe filtering unit that corresponds to an additional node interconnected with the processing nodes, where the additional node comprises a cache coherence controller, or a cache coherence controller comprising the probe filtering unit. Some examples include:

- *See, e.g.*, U.S. Patent No. 7,093,079 to Quach at
    - [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108.")
    - Figure 1

- [3:38-41] ("The coherent switches 106 may couple the processor cache nodes 102 and the I/O cache nodes 104 together and may help maintain data consistency or coherency among the cache nodes 102, 104.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a probe filtering unit that corresponds to an additional node interconnected with the processing nodes, where the additional node comprises a cache coherence controller, or a cache coherence controller comprising the probe filtering unit, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claims 2.1, 3.1, 6.1. A node controller for interconnecting and maintaining coherence between processing nodes and that included cache coherence directories for reducing coherence traffic was a well-known, common component for building multiprocessor systems having a relatively high number of processors as described above with respect to the "cache coherence controller."

16. *Probe filtering information comprises a cache coherence directory which includes entries corresponding to memory lines stored in the selected cache memories*

Some of the Asserted Claims are directed to probe filtering information comprising a cache coherence directory which includes entries corresponding to memory lines stored in the selected cache memories. *See*, *e.g.*, '121 claims 3.2, 6.1. At least under Memory Integrity's apparent infringement theories, probe filtering information comprising a cache coherence directory which includes entries corresponding to memory lines stored in the selected cache memories was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claims 3.2, 6.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose probe filtering

information comprising a cache coherence directory which includes entries corresponding to memory lines stored in the selected cache memories. Some examples include:

- C. K. Tang, "Cache System Design in the Tightly Coupled Multiprocessor System," Proceedings of the National Computer Conference and Exposition, June 1976, pp. 749-753
- Lucien M. Censier and Paul Feautrier, "A New Solution to Coherence Problems in Multicache Systems," IEEE Transactions on Computers, Vol. C-27, No. 12, December 1978, pp. 1112-1118
- James Archibald and Jean-Loup Baer, "An Economical Solution to the Cache Coherence Problem," Proceedings of the 11th Annual International Symposium on Computer Architecture, Vol. 12, No. 3, June 1984, pp. 355-362
- Anant Agarwal et al., "An Evaluation of Directory Schemes for Cache Coherence," Proceedings of the 15th Annual International Symposium on Computer Architecture, May 30-June 2, 1988, pp. 280-289
- Per Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," IEEE Computer, Vol. 23, No. 6, June 1990, pp. 12-24
- Daniel Lenoski et al., "The Stanford Dash Multiprocessor," IEEE Computer, Vol. 25, No. 3, March 1992, pp. 63-79
- John Hennessy et al., "Cache-Coherent Distributed Shared Memory: Perspectives on Its Development and Future Challenges," Proceedings of the IEEE, Special Issue on Distributed Shared Memory, Vol. 87, No. 3, March 1999, pp. 418-429
- U.S. Patent No. 6,085,295 to Ekanadham
- U.S. Publication No. 2001/0034816 to Michael
- Ashwini Nanda, Anthony-Trung Nguyen, Maged Michael, Doug Joseph, "High-throughput Coherence Control and Hardware Messaging in Everest," IBM Journal of Research and Development, Vol. 45, No. 2, March 2001, pp. 229-243

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include probe filtering information comprising a cache coherence directory which includes entries corresponding to memory lines stored in the selected cache memories, at

least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claims 3.2, 6.1. For example, a cache coherence directory includes information regarding the data stored in caches in the system, which can obviate the need to send probes to those caches, thus reducing coherence traffic in the system. Such directories are a well-known, routine mechanism for reducing coherence traffic that have been known and used for decades.

17. *Each of the processing nodes is operable to transmit the probes only to the probe filtering unit*

Some of the Asserted Claims are further directed to each of the processing nodes being operable to transmit the probes only to the probe filtering unit. *See*, *e.g.*, '121 claim 8.1. At least under Memory Integrity's apparent infringement theories, processing nodes operable to transmit the probes only to the probe filtering unit were well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claim 8.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose processing nodes operable to transmit the probes only to the probe filtering unit. Some examples include:

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - [2:10-22] ("Referring now to FIG. 1, an example computing device 100 may comprise one or more processor cache nodes 102, one or more input/output (I/O) cache nodes 104, and one or more coherent switches 106 that interconnect the processor nodes 102 and the I/O cache nodes 104. Each processor cache node 102 may comprise one or more processors 108, a node controller (SNC) 110 and memory 112. The processors 108 may execute code or instructions of the memory 112 and may process data of the memory 112 in response to executing such instructions. Further, the processors 108 may have associated caches 114 in which lines of the memory 112 may be stored and accessed more quickly by the associated processors 108.")
  - Figure 1
  - [3:38-41] ("The coherent switches 106 may couple the processor cache nodes 102 and the I/O cache nodes 104 together and may help maintain data consistency or coherency among the cache nodes 102, 104.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include processing nodes operable to transmit the probes only to the probe filtering unit, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claim 8.1. For example, processing nodes operable to transmit the probes only to the probe filtering unit may reduce the amount of coherence traffic in the system. Instead of transmitting probes to the other processing nodes in the system, a processing node transmits probes only to the probe filtering unit, which uses probe filtering information to forward the probe only to selected processing nodes in the system. For example, processing nodes that are connected to each other only via a switch that includes the probe filtering unit necessarily transmit probes only to the probe filtering unit. Using a switch to connect processing nodes was a well-known, common interconnect mechanism as described above.

18. *Each of the processing nodes is programmed to complete a memory transaction after receiving a first number of responses to a first probe, the first number being fewer than the number of processing nodes*

Some of the Asserted Claims are directed to each of the processing nodes being programmed to complete a memory transaction after receiving a first number of responses to a first probe, the first number being fewer than the number of processing nodes. *See*, *e.g.*, '121 claim 11.1. At least under Memory Integrity's apparent infringement theories, processing nodes programmed to complete a memory transaction after receiving a first number of responses to a first probe, the first number being fewer than the number of processing nodes, were well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claim 11.1.

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include processing nodes programmed to complete a memory transaction after receiving a first number of responses to a first probe, the first number being fewer than the number of processing nodes, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claim 11.1. For example, by sending probes to only selected processing nodes, instead of all processing nodes, a memory transaction may be completed after receiving responses only from those selected processing nodes, namely a number of responses equal to the number of selected processing nodes, which is fewer than the total number of processing nodes. Waiting for fewer responses generally may reduce the time needed to complete a transaction.

a) Probe filtering unit having temporary storage associated therewith for holding read response data from one of the cache memories, where the first number is one

Some of the Asserted Claims are further directed to a probe filtering unit having temporary storage associated therewith for holding read response data from one of the cache memories, where the first number is one. *See*, *e.g.*, '121 claim 12.1. At least under Memory Integrity's apparent infringement theories, a probe filtering unit having temporary storage associated therewith for holding read response data from one of the cache memories, where the first number is one, was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claim 12.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a probe filtering unit having temporary storage associated therewith for holding read response data from one of the cache memories, where the first number is one. Some examples include:

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at

- [6:16-36] ("The snoop completion detector 312 may determine based upon data of the snoop pending table 310 whether a final snoop response may be obtained for a transaction. In one embodiment, the snoop completion detector 312 may determine that a final snoop response may be obtained if the snoop pending table 310 indicates that all snoop requests associated with the transaction have completed (e.g. no snoop requests still in pending state). In another embodiment, the snoop completion detector 312 may determine that the final snoop response has been obtained prior to the snoop pending table 310 indicating that all snoop requests associated with the transaction have completed. For example, in one embodiment, the coherency protocol requires that a line modified in one cache node 102, 104 be in the invalid state in all other cache nodes 102, 104. Accordingly, if the snoop pending table 310 indicates a modified state for one of the cache nodes 102, 104, the snoop completion detector 312 may determine that the final snoop response for the transaction is the modified snoop response even if other snoop requests have yet to complete since the other cache nodes should have the line in the invalid state.")
- [8:17-37] ("Accordingly, the snoop completion detector 312 in the non-bypass mode may determine the final snoop response after receiving a snoop response from a single non-requesting cache node. However, in one embodiment, the coherent switch 106 may receive multiple types of snoop responses from the non-requesting cache nodes when in bypass-mode. Accordingly, the snoop completion detector 312 in one embodiment may simply determine the final snoop response after all snoop requests for the originating request have completed. In another embodiment, the snoop completion detector 312 may determine the final snoop response after receiving only a subset of the snoop responses for the issued snoop requests. For example, in one embodiment, the coherency protocol ensures that if one cache node has the line in the modified state, the other cache nodes should have the line in the invalid state. Accordingly, the snoop completion detector 312 may determine that the final snoop response is a modified snoop response after the CPL interleave 214 has received a modified snoop response and prior to receiving all the invalid snoop responses from the other non-requesting cache nodes.")

- *See*, *e.g.*, U.S. Patent No. 6,185,662 to Beyerlein at
  - [6:28-35] ("In fast forward mode, the data component from the first reply that contains no error and is not in a transitional state is forwarded to the requester such as processor 110 immediately. The processing element interface chip 114 stores the reply, and ensures the set modes from all replies with the same transaction identifier form a valid combination and that there are no other errors.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a probe filtering unit having temporary storage associated therewith for

holding read response data from one of the cache memories, where the first number is one, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claim 12.1. For example, a probe filtering unit that provides responses, including read response data from one of the cache memories, to a requesting processing node has temporary storage associated therewith for holding the read response data. In addition, a memory transaction may be completed after receiving one response to a probe when, for example, probes are sent to one processing node as indicated by the probe filtering information. Being able to send a single probe, instead of multiple probes, reduces the amount of coherence traffic. Waiting for fewer responses generally may reduce the time needed to complete a transaction. In another example, a memory transaction may be completed after receiving one response to a probe when the first probe response indicates the probed cache had the requested data in a modified state, indicating that the remaining caches have the requested data in an invalid state. *See*, *e.g.*, Quach at 6:16-36, 8:13-17.

b) Probe filtering unit operable to forward read response data to a requesting node before accumulating all probe responses associated with the memory transaction, where the first number is two

Some of the Asserted Claims are further directed to a probe filtering unit operable to forward read response data to a requesting node before accumulating all probe responses associated with the memory transaction, where the first number is two. *See*, *e.g.*, '121 claim 13.1. At least under Memory Integrity's apparent infringement theories, a probe filtering unit operable to forward read response data to a requesting node before accumulating all probe responses associated with the memory transaction, where the first number is two, was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claim 13.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a probe filtering unit operable to forward read response

data to a requesting node before accumulating all probe responses associated with the memory transaction, where the first number is two. Some examples include:

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - [6:16-36] ("The snoop completion detector 312 may determine based upon data of the snoop pending table 310 whether a final snoop response may be obtained for a transaction. In one embodiment, the snoop completion detector 312 may determine that a final snoop response may be obtained if the snoop pending table 310 indicates that all snoop requests associated with the transaction have completed (e.g. no snoop requests still in pending state). In another embodiment, the snoop completion detector 312 may determine that the final snoop response has been obtained prior to the snoop pending table 310 indicating that all snoop requests associated with the transaction have completed. For example, in one embodiment, the coherency protocol requires that a line modified in one cache node 102, 104 be in the invalid state in all other cache nodes 102, 104. Accordingly, if the snoop pending table 310 indicates a modified state for one of the cache nodes 102, 104, the snoop completion detector 312 may determine that the final snoop response for the transaction is the modified snoop response even if other snoop requests have yet to complete since the other cache nodes should have the line in the invalid state.")
  - [8:17-37] ("Accordingly, the snoop completion detector 312 in the non-bypass mode may determine the final snoop response after receiving a snoop response from a single non-requesting cache node. However, in one embodiment, the coherent switch 106 may receive multiple types of snoop responses from the non-requesting cache nodes when in bypass-mode. Accordingly, the snoop completion detector 312 in one embodiment may simply determine the final snoop response after all snoop requests for the originating request have completed. In another embodiment, the snoop completion detector 312 may determine the final snoop response after receiving only a subset of the snoop responses for the issued snoop requests. For example, in one embodiment, the coherency protocol ensures that if one cache node has the line in the modified state, the other cache nodes should have the line in the invalid state. Accordingly, the snoop completion detector 312 may determine that the final snoop response is a modified snoop response after the CPL interleave 214 has received a modified snoop response and prior to receiving all the invalid snoop responses from the other non-requesting cache nodes.")
- *See*, *e.g.*, U.S. Patent No. 6,185,662 to Beyerlein at
  - [6:28-35] ("In fast forward mode, the data component from the first reply that contains no error and is not in a transitional state is forwarded to the requester such as processor 110 immediately. The processing element interface chip 114 stores the reply, and ensures the set modes from all replies with the same transaction identifier form a valid combination and that there are no other errors.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a probe filtering unit operable to forward read response data to a requesting node before accumulating all probe responses associated with the memory transaction, where the first number is two, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claim 13.1. For example, forwarding read response data before accumulating all probe responses associated with the memory transaction allows the requesting processing node to receive the requested data faster. *See*, *e.g.*, Beyerlein at 6:36-39. In addition, a memory transaction may be completed after receiving two responses to a probe when, for example, probes are sent to two processing nodes as indicated by the probe filtering information. Being able to send two probes, instead of more than two probes, reduces the amount of coherence traffic. Waiting for fewer responses generally may reduce the time needed to complete a transaction. In another example, a memory transaction may be completed after receiving two responses to a probe when the second probe response indicates the probed cache had the requested data in a modified state, indicating that the remaining caches have the requested data in an invalid state. *See*, *e.g.*, Quach at 6:16-36, 8:13-17.

19. *Probe filtering unit operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit*

At least one of the Asserted Claims is further directed to a probe filtering unit operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit. *See*, *e.g.*, '121 claim 14.1. At least under Memory Integrity's apparent infringement theories, a probe filtering unit operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit was well-known in the

art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claim 14.1. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a probe filtering unit operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit. Some examples include:

- *See*, *e.g.*, U.S. Patent No. 5,890,217 to Kabemoto at
  - Figure 39
  - [47:49-57] ("The inter-subsystem connection unit 513 of the subsystem #2 which received the bus command from the inter-subsystem connection unit 512 of the subsystem #1 changes the unit ID in the source field of the first word of the bus command to the designation of the unit ID #7 of itself. Further, the unit 513 changes the destination field to the designation of the unit ID #3 decoded from the second word and transmits to the system bus #2. The second word of the bus command is transmitted as it is.")
  - Figure 52
  - [47:65-48:3] ("When the read data can be prepared, as shown in (4), the reply bus command is transmitted. The reply bus command sets the subsystem extension identifier EX to the source field of the first word, further, sets the bus ID #2 and unit ID #3, and designates the unit ID #7 of the inter-subsystem connection unit 513 for the destination field.")
- *See*, *e.g.*, U.S. Patent No. 6,085,295 to Ekanadham at
  - Abstract ("A method of providing coherent shared memory access among a plurality of shared memory multiprocessor nodes. For each line of data in each of the nodes, a list of those processors of the node that have copies of the line in their caches is maintained. If a memory command is issued from a processor of one node, and if the command is directed to a line of memory of another node, then the memory command is sent directly to an adapter of the one node. When the adapter receives the command, it forwards the command from the one adapter to another adapter of the other node. When the other adapter receives the command, the command is forwarded to the local memory of the other node. The list of processors is then updated in the local memory of the other node to include or exclude the other adapter depending on the command. If the memory command is issued from one of the processors of one of the nodes, and if the command is directed to a line of memory of the one node, then the command is sent directly to local memory. When the local memory receives the command and if the adapter of the node is in the list of processors for a line associated with the command and if the command is a write command, then the command is forwarded to the adapter of the one node. When the

adapter receives the command, the command is forwarded to remote adapters in each of the remote nodes which have processors which have cache copies of the line. Finally, when the latter remote adapters receive the command, the command is forwarded to the processors having the cache copies of the line.")

- [1:13-2:2] ("A shared-memory multiprocessor system, comprised of a plurality of processing nodes with memory and caches, provides system-wide access to the memory in the system. It is imminent that each node of such parallel systems in the near future is a small cache-coherent multiprocessor, e.g, a symmetric multiprocessor (SMP), that consists of a small number (8 to 16) of slots connected by a bus or switch. Each slot can be occupied by a processor or a memory module. Each processor in the node can access any memory location in the node.
- Technology considerations limit the size of an SMP node to a small number of processors. A method for building a shared-memory multiprocessor with a larger number of processors is to connect a number of SMP nodes with a network, and provide an adapter to extend the SMP's memory across the SMP nodes (see FIG. 1). Existing adapter designs plug into the memory bus of bus-based SMP nodes and collectively provide shared memory across the system, so that any processor in any node can access any location in any memory module in the system. Resources within a node are termed local and resources on other nodes are termed remote.
- The adapter maintains a directory of all nodes sharing a line and monitors local accesses to the line in order to ensure coherence across nodes. On bus-based SMPs, the monitoring is straightforward. All address transactions appear on the bus and the adapter can snoop and respond to them. Thus, it is possible to design the adapter without having to make any changes to the SMP hardware, provided that the adapter can be connected to the bus as a master/slave device.
- However, as the size and speed of an SMP node grows, technology limitations force the transition from bus-based to switch-based interconnects for both address and data transactions within the node. The design of an adapter for switch-based systems is complicated by the fact that a switch-based system uses a point-to-point interconnection that, unlike a bus, does not allow an adapter to observe all address transactions. In a switch-based SMP, the memory M maintains a directory 26. For each line 25 the directory keeps a list 24-x of the processors within the node that have cached copies of the line (see FIG. 2), where x is an integer between 1 and n, where n is the number of lines 25. See for example one of the lines 25 and its list 24-4 at the bottom of FIG. 2. It is understood that memory M can be any one of the memories M1 through MN.
- Each processor communicates directly with the memory via the switch. In turn, the memory sends appropriate messages only to processors that need to be involved in the cache coherence protocol, and the memory has no knowledge of the adapter.
- There is, therefore, a need for an adapter which extends shared memory access across multiple switch-based multiprocessor nodes. Such an adapter must not rely on a broadcast of memory commands within a multiprocessor node, and must not require changes to the existing memory controller of a multiprocessor node.")

- [2:5-3:4] ("It is therefore an objective of this invention to provide a solution to the problem of providing shared-memory access across multiple switch-based SMP nodes.
- The invention comprises an adapter to extend cache-coherent memory access across SMP nodes, and a method for using the memory system of a switch-based SMP node to interface to the adapter.
- The key concepts in the adapter design are:
- All communications between the processors, the memories and the adapters are made point to point without the need for broadcasts within the SMP node.
- In a node where a line is mapped onto the node's memory, the adapter acts as a proxy processor representing all the processors outside the node that share the line. More specifically, when a remote processor issues a memory command to a local memory, the remote adapter at the remote processor's node, forwards the memory command to the local adapter, which is responsible for insuring that the command is executed at the local memory.
- In a node where a remote line is brought into the cache of a processor, but not into the node's memory, the adapter acts as a proxy memory representing the remote memory that the line is mapped onto. More specifically, when a memory command is issued from a local processor to a remote memory, the memory command is directed to the adapter which is responsible for insuring that the command is executed at that remote memory.
- The adapter is versatile enough to be used for either CC-NUMA (Cache Coherent Non Uniform Memory Access) and S-COMA (Simple Cache Only Memory Architecture) systems.
- By appearing as either a local processor or a local memory, the adapter uses the local SMP coherence protocol within a node to accomplish the above tasks, without any changes to the memory controllers.
- In situations where the memory controller is limited in the amount of storage it can use for the directory and must employ a dynamic allocation scheme for the directory entries, an extension of this invention involves a modification to the memory controller that overcomes the the storage limitation of the memory controller.
- Accordingly, this invention provides coherent shared memory access across a number of interconnected multiprocessor nodes. With this invention each line of data in each memory of the node maintains a list of processors of the node that have copies of the line in their caches. When a memory command is issued from one of the processors of a node to a memory of another node, the command is directed to an adapter of the issuing node. The adapter of the issuing node then receives the command and forwards the command to the adapter at the remote node. When the adapter at the remote node receives the command, it then forwards the command to its local memory, which then updates its list of processors to include or exclude the

adapter. However, if the memory command is issued from a processor to a local memory then the command is simply forwarded directly to that local memory. When that local memory receives the command and if an adapter is in the list of processors for the line in the memory command, then the command is forwarded to that adapter. The adapter then forwards the command to the other adapters of remote nodes which have cache copies of the line corresponding to the command. If the adapter is not found in the list, then the memory proceeds in accordance with the standard SMP protocol.")

- *See*, *e.g.*, Luiz A. Barroso, Michel Dubois, "Cache Coherence on a Slotted Ring," Proceedings of the 20th International Conference on Parallel Processing, August 1991 at

    - Page 6 ("In this class of protocols [14] every coherence message is directed to the block's home cluster, which keeps track of the state and location of all cached copies of the block. In directory protocols, the home cluster allows sharing of a block for read-only copies but enforces exclusive access for writable copies. The home cluster satisfies all misses if there is no writable (dirty) copy of the block, forwarding the request to the dirty cluster otherwise. When receiving an invalidation, the home cluster sends invalidation messages selectively only to the clusters sharing a particular block, and replies to the requesting cluster once all copies have been invalidated. The main difference between directory and snooping schemes is that directory-based protocols do not rely on the broadcasting of coherence information.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a probe filtering unit operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claim 14.1. For example, a probe filtering unit operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit may reduce the amount of coherence traffic in the system. Instead of each probe response being sent to both the requesting processing node and the probe filtering unit, to allow the probe filtering information to be updated, it may be sent to only the probe filtering unit, which can accumulate all the probe responses to send

a single response, in accordance with the accumulated responses, to the requesting processing node.

20. *Probe filtering unit operable to accumulate responses to each probe, and respond to requesting nodes in accordance with the accumulated responses*

Some of the Asserted Claims are further directed to a probe filtering unit operable to accumulate responses to each probe, and respond to requesting nodes in accordance with the accumulated responses. *See*, *e.g.*, '121 claims 15.1, 25.7, 25.8. At least under Memory Integrity's apparent infringement theories, a probe filtering unit operable to accumulate responses to each probe, and respond to requesting nodes in accordance with the accumulated responses was well-known in the art at the time of the alleged invention of the Asserted Claims. *See*, *e.g.*, Exhibits C-1 – C-8 claims 15.1, 25.7, 25.8. At least under Memory Integrity's apparent infringement theories, there are many additional exemplary prior art references that disclose a probe filtering unit operable to accumulate responses to each probe, and respond to requesting nodes in accordance with the accumulated responses. Some examples include:

- *See*, *e.g.*, U.S. Patent No. 7,093,079 to Quach at
  - Figures 1, 4
  - [3:38-41] ("The coherent switches 106 may couple the processor cache nodes 102 and the I/O cache nodes 104 together and may help maintain data consistency or coherency among the cache nodes 102, 104.")
  - [4:44-60] ("The protocol logic 202 may service transactions such as, for example, requests and responses received from the cache nodes 102, 104 and may issue transactions to the cache nodes 102, 104. In particular, the protocol logic 202 may decode transactions, may handle conflicts, may interface with the snoop filter 204, and may process transactions. In one embodiment, the protocol logic 202 may comprise distributed protocol logic (DPL) 210 and centralized protocol logic (CPL) 212. In one embodiment, each port 200 has an associated DPL 210 to locally implement portions of the protocol logic 202 for the respective port 200. In particular, the DPL 210 may comprise decode logic to decode incoming transactions and may comprise one or more buffers or queues to store data and/or other information associated with incoming and outgoing transactions while being

processed by the protocol logic 202 and/or awaiting responses from cache nodes 102, 104.")

- [6:16-36] ("The snoop completion detector 312 may determine based upon data of the snoop pending table 310 whether a final snoop response may be obtained for a transaction. In one embodiment, the snoop completion detector 312 may determine that a final snoop response may be obtained if the snoop pending table 310 indicates that all snoop requests associated with the transaction have completed (e.g. no snoop requests still in pending state). In another embodiment, the snoop completion detector 312 may determine that the final snoop response has been obtained prior to the snoop pending table 310 indicating that all snoop requests associated with the transaction have completed. For example, in one embodiment, the coherency protocol requires that a line modified in one cache node 102, 104 be in the invalid state in all other cache nodes 102, 104. Accordingly, if the snoop pending table 310 indicates a modified state for one of the cache nodes 102, 104, the snoop completion detector 312 may determine that the final snoop response for the transaction is the modified snoop response even if other snoop requests have yet to complete since the other cache nodes should have the line in the invalid state.")
- [7:50-8:46] ("In block 416, the ports 200 of the coherent switch 106 may receive responses from the non-requesting cache nodes 102, 104. In particular, the ports 200 may receive snoop responses or coherency state information for the line of the originating request and may receive a current copy of the line from the memory 112 of the home cache node or a cache 114 of the remote cache node 102, 104. The DPL 210 associated with the non-requesting ports 200 may forward the snoop responses to the CPL interleave 214 for further processing and may forward a current copy of the line to the DPL 210 associated with the requesting cache node and the DPL 210 associated with the home cache node. For example, one of the non-requesting cache nodes 102, 104 may have modified the line and may provide its associated non-requesting port 200 with the modified line. The DPL 210 associated with the port 200 that received the modified line may forward the modified line to the DPL 210 of the requesting port 200 for delivery to the requesting cache node and may forward the modified line to the DPL 210 of the home port 200 for writing the modified line to the memory 112 of the home cache node 102, 104.
- The CPL interleave 214 in block 418 may update the snoop pending table 310 with the received snoop response and may update the associated snoop request to a complete or not pending state. In block 420, the-snoop completion detector 312 may determine whether a final snoop response may be determined. In one embodiment, due to the coherency protocol used and the coherency data and presence vector of the snoop filter 204, the coherent switch 106 receives only one type of snoop response in response to the issued snoop requests. In particular all snoop responses received by the coherent request 106 for an originating request are invalid snoop responses, shared snoop responses, modified/shared snoop responses, or modified snoop responses. Accordingly, the snoop completion detector 312 in the non-bypass mode may determine the final snoop response after receiving a snoop response from a single non-requesting cache node. However, in one embodiment, the coherent switch 106 may receive multiple types of snoop

responses from the non-requesting cache nodes when in bypass-mode. Accordingly, the snoop completion detector 312 in one embodiment may simply determine the final snoop response after all snoop requests for the originating request have completed. In another embodiment, the snoop completion detector 312 may determine the final snoop response after receiving only a subset of the snoop responses for the issued snoop requests. For example, in one embodiment, the coherency protocol ensures that if one cache node has the line in the modified state, the other cache nodes should have the line in the invalid state. Accordingly, the snoop completion detector 312 may determine that the final snoop response is a modified snoop response after the CPL interleave 214 has received a modified snoop response and prior to receiving all the invalid snoop responses from the other non-requesting cache nodes.

- In block 422, the coherent switch 106 may complete the request. In particular, the DPL 210 associated with the requesting port 200 may provide the requesting cache node 102, 104 with the final snoop response and the requested line. Further, if the line was modified, the DPL 210 associated with the home cache node 102, 104 may write the modified line back to the memory 112 of the home cache node 102, 104.")

Furthermore, to the extent not disclosed, a person of ordinary skill in the art at the time of the alleged invention of the Asserted Claims would have been motivated to modify the prior art references identified in Section III and Exhibits A-1 – A-9; B-1 – B-19; C-1 – C-8; D-1 – D-14; and E-1 – E-14 to include a probe filtering unit operable to accumulate responses to each probe, and respond to requesting nodes in accordance with the accumulated responses, at least under Memory Integrity's apparent infringement theories. *See*, *e.g.*, Exhibits C-1 – C-8 claims 15.1, 25.7, 25.8. For example, a probe filtering unit operable to accumulate responses to each probe and respond to requesting nodes in accordance with the accumulated responses may reduce the amount of coherence traffic in the system. Instead of each probe response being sent to both the requesting processing node and the probe filtering unit, to allow the probe filtering information to be updated, it may be sent to only the probe filtering unit, which can accumulate all the probe responses to send a single response, in accordance with the accumulated responses, to the requesting processing node.

G. Contentions Under 35 U.S.C. § 112

Intel provides the following initial disclosures of invalidity contentions based on 35 U.S.C. § 112. Intel reserves the right to amend and/or supplement its invalidity contentions, including its contentions based on 35 U.S.C. § 112, as discovery proceeds. Memory Integrity failed to describe its Infringement Contentions with the required degree of particularity. Intel reserves the right to amend its invalidity contentions in the event that Memory Integrity narrows its asserted claims, and/or supplements or amends its Infringement Contentions. Intel also reserves the right to amend its invalidity contentions in the event that Memory Integrity contends that any item of prior art that Intel contends either anticipates or renders obvious an asserted claim does not describe or enable one or more of the elements of the asserted claim. Intel further reserves its right to amend its invalidity contentions in response to developments in fact and expert discovery.

1. *The Written Description and Enablement Requirements*

The written description and enablement requirements of patent law are provided in 35 U.S.C. § 112, first paragraph, which reads as follows:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same, and shall set forth the best mode contemplated by the inventor of carrying out his invention.

See Ariad Pharmaceuticals, Inc. v. Eli Lilly and Co., 598 F.3d 1336, 1344 (Fed. Cir. 2010) (en banc) (holding "that § 112, first paragraph, contains two separate description requirements: a written description [i] of the invention, and [ii] of the manner and process of making and using [the invention]").

2. *Lack of Written Description*

Under MI's infringement theories, MI's asserted patent specifications do not describe the full scope of its claims as required by 35 U.S.C § 112, first paragraph, and the asserted patent

claims are therefore invalid. The asserted patent claims fail to meet the written description requirement for multiple independent reasons.

First, for all of the asserted claims of all of the asserted patents, MI has alleged that the claim terms "cluster of processors" and "plurality of processing nodes" could be met by a single processor chip with multiple cores. But the specifications for the '409, '636, '121, '206, and '254 patents do not describe a single processor chip with multiple cores, and never suggest that the required "cluster of processors" or "plurality of processing nodes" could be a single processor chip with multiple cores. The difference between a cluster of separate processor chips and a single chip with multiple cores is important and material because multi-core chips require fundamentally different designs, protocols, and manufacturing process technology. The specifications of the asserted patents thus do not adequately describe the full scope of the claims under MI's apparent infringement theories because the asserted patent specifications do not show that MI was in possession of an embodiment of the invention in which a cluster included a plurality of cores and in which the cores and cache coherence controller were integrated in a single chip. In addition, the asserted patent specifications would not "clearly allow persons of ordinary skill in the art to recognize that [the inventors] invented" an implementation of the claimed invention where a cluster of processors is embodied in a single multi-core processor chip. *See Ariad*, 598 F.3d at 1351 ("[T]he test for sufficiency is whether the disclosure of the application relied upon reasonably conveys to those skilled in the art that the inventor had possession of the claimed subject matter as of the filing date."); *id.* ("[T]he description must clearly allow persons of ordinary skill in the art to recognize that [the inventor] invented what is claimed." (quotation marks omitted)).

Accordingly, under MI's infringement theories, all the asserted claims of all the asserted patents are invalid for failure to satisfy the written description requirement because the specifications do not describe implementing the claimed invention using a single processor chip with multiple cores as the "cluster of processors" or "plurality of processing nodes." *See Centocor Ortho Biotech, Inc. v. Abbott Labs.*, 636 F.3d 1341, 1353 (Fed. Cir. 2011) (finding plaintiff's patent invalid for lack of written description and explaining that "[t]he scope of [the patentee's] right to exclude cannot over-reach the scope of [its] contribution to the field of art as described in the patent specification." (internal quotation marks omitted)); *Abbvie Deutschland GmbH & Co. v. Janssen Biotech, Inc.*, 759 F.3d 1285, 1300 (Fed. Cir. 2014) (holding that to satisfy the written description requirement, patentees must demonstrate that they have "conceived and described sufficient representative species encompassing the breadth of the genus. Otherwise, one has only a research plan, leaving it to others to explore the unknown contours of the claimed genus.").

Second, at least under MI's infringement theories, MI's patents do not provide any substantive or meaningful description of many of the elements of its claims—*e.g.*, there is no substantial description of the internal hardware, firmware, or software required to implement many of the claim elements. This is not sufficient to show that MI was in possession of the claimed inventions as of the filing of the applications for the asserted patents. *See Ariad*, 598 F.3d at 1351. For this reason, all of the asserted claims that require the following claim elements are invalid for lack of written description. *See id.*

- **"Cache Coherence Controllers" / "Interconnection Controller."** Many of the asserted patent claims require cache coherence controllers or interconnection controllers. For example, claim 15 of the '409 patent recites "a first cache coherence controller" and claim 1 of the '121 patent recites "an interconnection controller." *See also, e.g.*, '409 patent claims 1.2-1.7, 6.2-6.8, 7.1, 7.2, 8.1, 9.1, 10.1, 11.1, 12.1, 18.1, 19.1, 20.1, 22.1, 23.1, 25.1, 25.4, 25.5, 34.4, 34.5, 51.1-51.4, and 52.1-52.4; '636 patent claims 11.1-11.5, 12.1, 15.2-15.7, 18.1, 21.2-21.8, 22.1-22.5, 23.1, 24.1, 25.1, 26.1, 27.1, 28.1, 29.1, 30.1, 31.1, 33.1, 34.1, 35.1, and 36.1-36.5; '121 patent claims

3.1, 3.2, 5.1, and 6.1; '206 patent claims 1.3, 1.4, 1.5, 1.6, 2.1, 15.1, 19.2, 21.1-21.6, 22.1, 24.1, 51.1, 27.1, 29.1, 30.2, 30.3, 31.1, 35.1, 38.2, 39.3, 39.4, and 39.6; and '254 patent claims 1.3, 1.4, 2.1, 7.1, and 8.1. However, MI's patents do not provide any substantive or meaningful description of these elements—*e.g.*, there is no substantial description of the internal hardware, firmware, or software required to implement the cache coherence controllers or interconnection controllers. Instead, MI's patents provide only a generic box diagram of the cache coherence controller (*e.g.*, '409 patent, Fig. 3) and other generic statements such as the "cache coherence controller 230 is a specially configured programmable chip" ('409 patent, 7:47-50). MI's asserted patent specifications thus do not show that the patentees were in possession of a cache coherence controller or interconnection controller that could meet all the requirements of the claimed inventions—and thus the claims that require these elements are invalid for lack of written description. *See Ariad*, 598 F.3d at 1351.

- **"Probe Filtering Unit."** All of the asserted claims of the '121 patent—*i.e.*, claims 1-6, 8, 11-17, 19-25—require a probe filtering unit. MI's patents do not provide any substantive or meaningful description of this claim element—*e.g.*, there is no substantial description of the internal hardware, firmware, or software required to implement the probe filtering unit. Instead, the '121 patent specification describes the functions performed or the desired result to be achieved by the probe filtering unit—*e.g.*, that the "probe filtering unit is operable to receive probes corresponding to memory lines from the processing nodes and to transmit the probes only to selected ones of the processing nodes with reference to probe filtering information." (*See* '121 patent, 2:52-55.) The specification also provides only generic diagrams and asserts that the "probe filtering unit" is not limited to any particular implementation or structure. (*See* '121 patent, 26:52-57 ("It should be understood that the use of the term 'probe filtering unit' or 'PFU' in the following discussion is not intended to be limiting or exclusive. Rather, any device or object operable to perform the described functionalities, e.g., a cache coherency controller as described herein, is within the scope of the invention.").) These disclosures are not sufficient to show that the patentees were in possession of a probe filtering unit that could meet all the requirements of the claimed invention—and thus the claims that require a probe filtering unit are invalid for lack of written description. *See Ariad*, 598 F.3d at 1351.

- **"Protocol Engines."** All of the asserted claims of the '206 and '254 patents—*i.e.*, '206 claims 1-2, 7, 14-15, 19, 21-22, 24-32, 34-35, 37-41, and 43-44 and '254 claims 1-3 and 5-8—require protocol engines. MI's patents do not provide any substantive or meaningful description of these protocol engines—*e.g.*, there is no substantial description of the internal hardware, firmware, or software required to implement the claimed protocol engines. Instead, the '206 and '254 patent specifications describe the functions performed or the desired result to be achieved by the claimed protocol engines—*e.g.*, that the "protocol engine 305 [is] configured to handle packets such as probes and requests received from processors in various clusters of a multi-processor system." ('206 patent, 4:46-50.) Similarly, the '206 and '254 patent specifications assert that "protocol engines are blocks of hardware on the interconnection controller ASIC chip" ('206 patent, 12:11-13), but the specifications do not describe that

hardware in any meaningful detail. These disclosures are not sufficient to show that the patentees were in possession of protocol engines that could meet all the requirements of the claimed invention—and thus the claims that require protocol engines are invalid for lack of written description. *See Ariad*, 598 F.3d at 1351.

3. *Lack of Enablement*

Under 35 U.S.C. § 112, first paragraph, a patent claim is invalid as not enabled if it fails to "teach those skilled in the art how to make and use the full scope of the claimed invention without undue experimentation." *MagSil Corp. v. Hitachi Global Storage Techs., Inc.*, 687 F.3d 1377, 1380 (Fed. Cir. 2012). The enablement requirement "prevents both inadequate disclosure of an invention and overbroad claiming that might otherwise attempt to cover more than was actually invented." *Id.* at 1381. The asserted patent claims are invalid for failure to meet the enablement requirement for multiple independent reasons:

First, for all of the asserted claims of all of the asserted patents, MI has alleged that the claim terms "cluster of processors" and "plurality of processing nodes" could be met by a single processor chip with multiple cores. But the specifications for the '409, '636, '121, '206, and '254 patents do not enable a person of skill in the art, as of the filing date of the asserted patents' applications, to implement the claimed inventions without undue experimentation where the required "cluster of processors" or "plurality of processing nodes" is a single processor chip with multiple cores. Indeed, the asserted patents never even suggest that the required "cluster of processors" or "plurality of processing nodes" could be implemented on a single processor chip with multiple cores, and the specifications provide no teaching or guidance on how to implement such a system without undue experimentation. The difference between a cluster of separate processor chips and a single chip with multiple cores is important and material because multi-core chips require fundamentally different designs, protocols, and manufacturing process technology. Thus, if the claims were to be construed to cover implementations of the claimed invention where

the claimed "cluster of processors" or "plurality of processing nodes" could be a single processor chip, as MI apparently contends, then all the asserted patent claims would be invalid for lack of enablement. *See Automotive Techs. Int'l, Inc. v. BMW of N. Am., Inc.*, 501 F.3d 1274, 1282 (Fed. Cir. 2007) (claims not enabled because they covered both mechanical and electronic side impact sensors, but specification failed to enable electronic side impact sensors); *Liebel-Flarsheim Co. v. Medrad, Inc.*, 481 F.3d 1371, 1380 (Fed. Cir. 2007) ("The irony of this situation is that Liebel successfully pressed to have its claims include a jacketless system, but, having won that battle, it then had to show that such a claim was fully enabled, a challenge it could not meet. The motto, 'beware of what one asks for,' might be applicable here."); *AK Steel Corp. v. Sollac & Ugine*, 344 F.3d 1234, 1244-1245 (Fed. Cir. 2003) (affirming finding of non-enablement where the claim was construed to cover steel strips containing either a Type 1 or a Type 2 aluminum coating, but the patent did not enable strips containing Type 1 aluminum coating).

Second, at least under MI's infringement theories, MI's patents do not provide any substantive or meaningful teaching regarding how to enable the claimed inventions—*e.g.*, there is no substantial teaching regarding the internal hardware, firmware, or software required to implement the claimed inventions. Accordingly, the patent specifications do not enable a person of ordinary skill in the art, as of the filing date of the asserted patents' applications, to practice the claimed inventions without undue experimentation—and thus the asserted claims are invalid for lack of enablement. *MagSil*, 687 F.3d at 1380. In particular, all of the asserted claims that require the following claim elements are invalid for lack of enablement. *See id.*

- **"Cache Coherence Controllers" / "Interconnection Controller."** Many of the asserted patent claims require cache coherence controllers or interconnection controllers. For example, claim 15 of the '409 patent recites "a first cache coherence controller" and claim 1 of the '121 patent recites "an interconnection controller." *See also, e.g.*, '409 patent claims 1.2-1.7, 6.2-6.8, 7.1, 7.2, 8.1, 9.1, 10.1, 11.1, 12.1, 18.1, 19.1, 20.1, 22.1, 23.1, 25.1, 25.4, 25.5, 34.4, 34.5, 51.1-51.4, and 52.1-52.4; '636

patent claims 11.1-11.5, 12.1, 15.2-15.7, 18.1, 21.2-21.8, 22.1-22.5, 23.1, 24.1, 25.1, 26.1, 27.1, 28.1, 29.1, 30.1, 31.1, 33.1, 34.1, 35.1, and 36.1-36.5; '121 patent claims 3.1, 3.2, 5.1, and 6.1; '206 patent claims 1.3, 1.4, 1.5, 1.6, 2.1, 15.1, 19.2, 21.1-21.6, 22.1, 24.1, 51.1, 27.1, 29.1, 30.2, 30.3, 31.1, 35.1, 38.2, 39.3, 39.4, and 39.6; and '254 patent claims 1.3, 1.4, 2.1, 7.1, and 8.1. However, MI's patents do not provide any substantive or meaningful teaching regarding how to practice these elements—*e.g.*, there is no substantial teaching regarding the internal hardware, firmware, or software required to implement the claimed cache coherence controller or interconnection controller. Instead, MI's patents provide only a generic box diagram of the cache coherence controller (*e.g.*, '409 patent, Fig. 3) and other generic statements such as the "cache coherence controller 230 is a specially configured programmable chip" ('409 patent, 7:47-50). MI's asserted patent specifications thus do not enable a person of ordinary skill in the art, as of the filing date of the asserted patents' applications, to practice the claimed cache coherence controller or interconnection controller without undue experimentation—and thus the claims that require these elements are invalid for lack of enablement. *MagSil*, 687 F.3d at 1380.

- **"Probe Filtering Unit."** All of the asserted claims of the '121 patent—i.e., claims 1-6, 8, 11-17, 19-25—require a probe filtering unit. However, MI's patents do not provide any substantive or meaningful teaching regarding how to practice the claimed probe filtering unit—*e.g.*, there is no substantial teaching regarding the internal hardware, firmware, or software required to implement the probe filtering unit. Instead, the '121 patent specification describes the functions performed or the desired result to be achieved by the probe filtering unit—*e.g.*, that the "probe filtering unit is operable to receive probes corresponding to memory lines from the processing nodes and to transmit the probes only to selected ones of the processing nodes with reference to probe filtering information." (*See* '121 patent, 2:52-55.) The specification also provides only generic diagrams and asserts that the "probe filtering unit" is not limited to any particular implementation or structure. (*See* '121 patent, 26:52-57 ("It should be understood that the use of the term 'probe filtering unit' or 'PFU' in the following discussion is not intended to be limiting or exclusive. Rather, any device or object operable to perform the described functionalities, e.g., a cache coherency controller as described herein, is within the scope of the invention.").) MI's asserted patent specifications thus do not enable a person of ordinary skill in the art, as of the filing date of the asserted patents' applications, to practice the claimed probe filtering unit without undue experimentation—and thus the claims that require this element are invalid for lack of enablement. *MagSil*, 687 F.3d at 1380.

- **"Protocol Engines."** All of the asserted claims of the '206 and '254 patents—*i.e.*, '206 claims 1-2, 7, 14-15, 19, 21-22, 24-32, 34-35, 37-41, and 43-44 and '254 claims 1-3 and 5-8—require protocol engines. However, MI's patents do not provide any substantive or meaningful teaching regarding how to practice the claimed protocol engines—*e.g.*, there is no substantial teaching regarding the internal hardware, firmware, or software required to implement the claimed protocol engines. Instead, the '206 and '254 patent specifications describe the functions performed or the desired result to be achieved by the claimed protocol engines—*e.g.*, that the "protocol engine

> 305 [is] configured to handle packets such as probes and requests received from processors in various clusters of a multi-processor system." ('206 patent, 4:46-50.) Similarly, the '206 and '254 patent specifications assert that "protocol engines are blocks of hardware on the interconnection controller ASIC chip" ('206 patent, 12:11-13), but the specifications do not describe that hardware in any meaningful detail. MI's asserted patent specifications thus do not enable a person of ordinary skill in the art, as of the filing date of the asserted patents' applications, to practice the claimed protocol engines without undue experimentation—and thus the claims that require this element are invalid for lack of enablement. *MagSil*, 687 F.3d at 1380.

4. *Indefiniteness*

Under 35 U.S.C. § 112, a patent claim is invalid for indefiniteness if its language, when "read in light of the specification and the prosecution history, fail[s] to inform, with reasonable certainty, those skilled in the art about the scope of the invention." *Nautilus, Inc. v. Biosig Instruments, Inc.*, 134 S. Ct. 2120, 2124 (2014) (quotation marks omitted).

Dependent claim 6 of the '254 patent is invalid as indefinite. Claim 6 reads as follows:

> 6. The cluster of claim 5, wherein the circuitry is contained in at least one of said plurality of processing nodes and said circuitry is further configured to forward at least one of said one or more packets ***to a selected the first one or more protocol engines of the plurality of protocol engines*** with reference to at least one bit of the target address.

('254 patent, claim 6 (emphasis added).) In claim 6, the use of the phrase "to a selected the first one or more protocol engines of the plurality of protocol engines" is nonsensical and thus does not provide "reasonable certainty" to a person of ordinary skill in the art regarding the scope of the claim. *Nautilus*, 134 S. Ct. at 2124.

In addition, claim 34 of the '409 patent and claim 36 of the '636 patent recite a cache coherence controller or aspects of such a controller with "means-plus-function" elements. Accordingly, the scope of these claim elements is limited to the specific structures disclosed in the '409 and '636 patent specifications for performing the functions recited in the claim elements, as well as equivalent structures. *See* 35 U.S.C. § 112, sixth paragraph.

However, as noted above, the '409 and '636 patent specifications do not disclose substantive or meaningful structure for these elements—*e.g.*, there is no substantial description of the internal hardware, firmware, or software required to implement the cache coherence controllers or interconnection controllers. Instead, MI's patents provide only a generic box diagram of the cache coherence controller (*e.g.*, '409 patent, Fig. 3) and other generic statements such that the "cache coherence controller 230 is a specially configured programmable chip" ('409 patent, 7:47-50). Since the scope of these claim elements is limited to the structure disclosed in the patents for performing their claimed function, and since the specification does not disclose the details of structure for these elements, the claims are invalid for being indefinite under 35 U.S.C. § 112, second paragraph. *See, e.g., Biomedino, LLC v. Waters Tech. Corp.*, 490 F.3d 946, 952 (Fed. Cir. 2007) (holding that claim reciting a "control means for operating [a] valving" was indefinite because the specification's disclosure that a valve "may be controlled by known differential pressure, valving and control equipment" did not constitute sufficient structure).

IV. Invalidity under 35 U.S.C. § 101

Under 35 U.S.C. § 101, transitory signals and computer media are not patentable subject matter. *See In re Nuijten*, 500 F.3d 1346, 1357 (Fed. Cir. 2007) ("[S]ignals, standing alone, are not 'manufacture[s]' under the meaning of that term in § 101."); *see also Ex Parte Satish Laxmanrao*, 2015 WL 2378820, at *4 n.4 (P.T.A.B. May 14, 2015) ("Because Appellant's Specification does not define computer-usable storage medium to exclude transitory media, the claimed medium encompasses transitory media, which is not patent eligible."); *Ex parte Mewherter*, No. 2012-7692, 2013 Pat. App. LEXIS 5934, at *6 (P.T.A.B. May 8, 2013) (precedential) (holding that claims directed to a "machine-readable storage medium" were directed to non-patentable subject matter under § 101).

Under this precedent, claims 19-23 of the '121 patent are directed to non-patentable subject matter under 35 U.S.C. § 101 because they claim a "computer readable medium" and are not limited to a non-transitory computer readable medium. *See Ex parte Mewherter*, 2013 Pat. App. LEXIS 5934, at *6 (explaining that claims directed to a "computer readable storage medium" encompass both non-transitory and transitory media and that transitory media are not patentable under § 101).

DATED this 27th day of May 2015.

By:

/s/ Michael J. Summersgill

Renée E. Rothauge (OSB #903712)
MARKOWITZ HERBOLD PC
1211 SW Fifth Avenue, Suite 3000
Portland, OR 97204-3730
(503) 295-3085

Michael J. Summersgill (admitted *pro hac vice*)
Jordan L. Hirsch (admitted *pro hac vice*)
Sean K. Thompson (admitted *pro hac vice*)
WILMERHALE LLP
60 State Street
Boston, MA 02109
(617) 526-6000
michael.summersgill@wilmerhale.com
jordan.hirsch@wilmerhale.com
sean.thompson@wilmerhale.com

Grant K. Rowan (admitted *pro hac vice*)
WILMERHALE LLP
1875 Pennsylvania Avenue NW
Washington, DC 20006
(202) 663-6011
grant.rowan@wilmerhale.com

Arthur W. Coviello (admitted *pro hac vice*)
WILMERHALE LLP
950 Page Mill Road
Palo Alto, CA 94304
(650) 858-6000
Arthur.coviello@wilmerhale.com

Attorneys for Intel Corporation