



- (51) International Patent Classification:
G06F 12/0811 (2016.01)
- (21) International Application Number:
 PCT/US2017/030586
- (22) International Filing Date:
 02 May 2017 (02.05.2017)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
 15/169,118 31 May 2016 (31.05.2016) US
- (71) Applicant: **ADVANCED MICRO DEVICES, INC.**
 [US/US]; One AMD Place, Sunnyvale, California 94088 (US).
- (72) Inventors: **BOYER, Michael W.**; AMD - US Bellevue, 2002 156th Avenue NE, Suite 300, Bellevue, Washington 98007 (US). **JAYASENA, Nuwan**; AMD - US Sunnyvale, One AMD Place, Sunnyvale, California 94088 (US).
- (74) Agent: **CAVE, Nicole Teitler et al.**; Zagorin Cave LLP, 11675 Jollyville Road, Suite 100, Austin, Texas 78759 (US).
- (81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.
- (84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,

(54) Title: CACHE COHERENCE FOR PROCESSING IN MEMORY

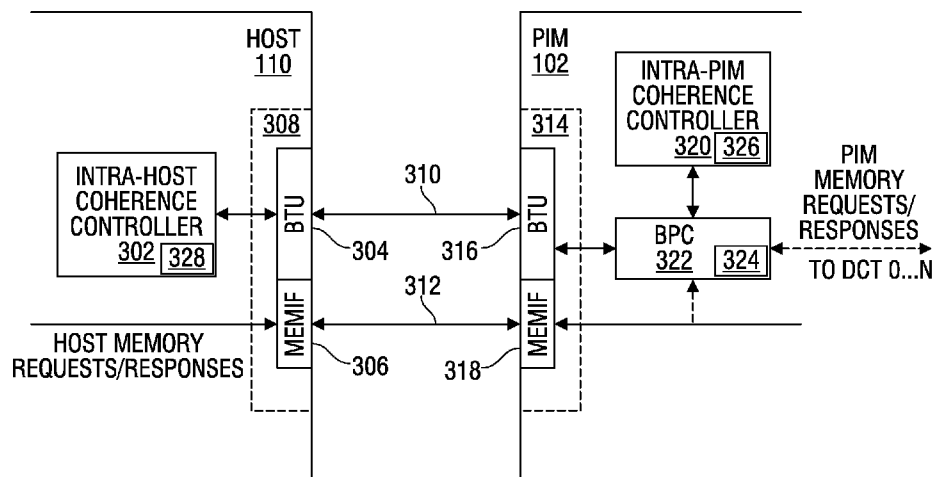


FIG. 3

(57) Abstract: A cache coherence bridge protocol provides an interface between a cache coherence protocol of a host processor and a cache coherence protocol of a processor-in-memory, thereby decoupling coherence mechanisms of the host processor and the processor-in-memory. The cache coherence bridge protocol may be used to facilitate interoperability between host processors and processor-in-memory devices designed by different vendors and both the host processors and processor-in-memory devices may implement coherence techniques among computing units within each processor. The cache coherence bridge protocol may support different granularity of cache coherence permissions than those used by cache coherence protocols of a host processor and/or a processor-in-memory.



TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,
KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

CACHE COHERENCE FOR PROCESSING IN MEMORY

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

This invention was made with Government support under Prime Contract Number
5 DE-AC52-07NA27344, Subcontract No. B608045 awarded by DOE. The
Government has certain rights in the invention.

TECHNICAL FIELD

The invention is related to processing systems and more particularly to cache
coherence in processing systems.

10 BACKGROUND ART

In general, processor-in-memory (i.e., processing-in-memory or processing near
memory) techniques improve energy efficiency and performance for data-intensive
applications by incorporating processing elements with memory (e.g., random access
memory). Incorporating processing-in-memory within a heterogeneous system
15 architecture framework and extending conventional programming models to those
systems processors requires cache coherence. Advanced processor-in-memory
designs incorporate complex processing engines with their own cache hierarchies.
However, for proper operation, a system should maintain coherence between those
caches and the caches of host processors (i.e., main processors). Even simple,
20 cacheless processor-in-memory designs must ensure coherence between host caches
and data being processed in a processor-in-memory.

Conventional techniques for providing data coherence in systems including a
processor-in-memory may include no hardware support and may require software to
orchestrate cache coherence between a host and a processor-in-memory. Such
25 solutions require complex programming. Other techniques extend existing cache
coherence solutions to encompass any processor-in-memory design. Extending
traditional cache coherence solutions, which require integration among all processors
in the system, to a processor-in-memory may be impractical because it requires the
cache coherence implementation on the host to be aware of how many memory

devices (i.e., processor-in-memory devices) the system includes and limits the system design to a particular version or type of host processor, which limits design interoperability and scalability. In addition, since cache coherence protocol designs are complex, such designs are difficult to verify and validate, resulting in

5 undesirability of any changes to existing, proven host cache coherence protocols. Other proposed processor-in-memory cache coherence schemes that do not require any modifications to existing coherence protocols do not permit common usage scenarios such as read sharing (i.e., caching of read-only data at both the host and the processor-in-memory).

10 Still other cache coherence provisions for processor-in-memory architectures do not account for caches in processor-in-memory devices and rely on extending host-side cache coherence techniques to ensure that host caches remain coherent as processor-in-memory devices modify the data. Such techniques require processor-in-memory devices to use cache line sizes and protocols consistent with those of the host. In

15 addition, those provisions may require storage of one or more dedicated cache coherence state bits along with each cache line in memory (e.g., dynamic random access memory (DRAM)), which requires specialized memory modules and may preclude the use of readily-available memory components. Accordingly, improved techniques for implementing cache coherence for a system incorporating processor-

20 in-memory devices are desired.

DISCLOSURE OF THE INVENTION

In at least one embodiment of the invention, a system includes a processor-in-memory. The processor-in-memory includes shadow directory storage configured to store a bridge coherence state of a data block residing in memory of the processor-in-

25 memory. The bridge coherence state indicates a status of any copies of the data block stored external to the processor-in-memory. The processor-in-memory includes bridge protocol translation logic configured to translate coherence messages between a first cache coherence protocol of the processor-in-memory and a second cache coherence protocol of a host processor. The second cache coherence protocol is

30 different from the first cache coherence protocol. The processor-in-memory includes bridge protocol control logic configured to maintain data coherence of the memory of

the processor-in-memory using the coherence messages and contents of the shadow directory storage. The processor-in-memory may include a stack of integrated circuit dies. The memory of the processor-in-memory may be included in a plurality of memory integrated circuit dies of the stack of integrated circuit dies. The bridge protocol control logic and the bridge protocol translation logic may be included in a logic integrated circuit die of the stack of integrated circuit dies. The second cache coherence protocol may use block sizes different from block sizes of the first cache coherence protocol. The processor-in-memory may include at least one cache of a corresponding processor. The at least one cache may be within a coherence domain of the processor-in-memory. The at least one cache may be a write-through cache and the bridge coherence state of the data block may be one of a *no-host-copy* state, a *host-read* state, and a *host-written* state. The at least one cache may be a write-back cache and the bridge coherence state of the data block may be one of a *no-host-copy* state, a *host-read* state, a *host-written* state, and a *processor-in-memory-modified* state. The processor-in-memory may include a memory-side cache outside a coherence domain of the processor-in-memory. The bridge protocol control logic may be configured to check the status of any copies of the data block stored in the memory-side cache in response to a memory access to the data block. The processor-in-memory may include a processor-in-memory directory configured to store a processor-in-memory coherence state of the data block stored within the at least one cache of the corresponding processor. The processor-in-memory may include an intra-processor-in-memory coherence controller configured to maintain data coherence of the data block stored in the at least one cache using contents of the processor-in-memory directory. The system may include the host processor coupled to the processor-in-memory. The host processor may include host bridge translation logic configured to translate coherence messages between the first cache coherence protocol of the processor-in-memory and the second cache coherence protocol of the host processor. The host processor may further include a memory-side cache outside a coherence domain of the host and outside a coherence domain of the processor-in-memory. The shadow directory storage may capture the status of any copies of the data block stored in the memory-side cache. The host processor may further include a host directory configured to store a host coherence state of a host copy data block, stored within the host processor, and intra-host coherence control logic configured to

maintain data coherence of the host copy data block residing in the host processor using the host directory.

In at least one embodiment of the invention, a method includes storing, in a shadow directory, a bridge coherence state of a data block residing in a memory of a processor-in-memory. The bridge coherence state indicates a status of any copies of the data block stored external to the processor-in-memory. The method includes translating coherence messages between a first cache coherence protocol of the processor-in-memory and a second cache coherence protocol of a host processor. The second cache coherence protocol is different from the first cache coherence protocol.

5 The method includes maintaining data coherence of the memory of the processor-in-memory using the coherence messages and contents of the shadow directory storage. The second cache coherence protocol may use block sizes different from block sizes of the first cache coherence protocol. Maintaining data coherence of the memory of the processor-in-memory may include updating the bridge coherence state of the data

10 block in response to an access to the data block. Maintaining data coherence of the memory of the processor-in-memory may include maintaining coherence of at least one cache of a processor of the processor-in-memory with the memory of the processor-in-memory. The at least one cache may be within a coherence domain of the processor-in-memory. The method may include checking a memory-side cache of the processor-in-memory in response to an access of the memory of the processor-in-

15 memory. The method may include implementing a write-through cache policy from a cache in a coherence domain of the processor-in-memory to the memory-side cache of the processor-in-memory. The bridge coherence state of the data block in the shadow directory may be one of a *no-host-copy* state, a *host-read* state, and a *host-written*

20 state. The method may include implementing a write-back cache policy from a cache in a coherence domain of the processor-in-memory to a memory-side cache of the processor-in-memory. The bridge coherence state of the data block in the shadow directory may be one of a *no-host-copy* state, a *host-read* state, a *host-written* state, and a *processor-in-memory-modified* state. Maintaining data coherence of the

25 memory of the processor-in-memory may include generating coherence messages in response to a memory access to the data block. The method may include storing a processor-in-memory coherence state of the data block stored within at least one

30

cache of a corresponding processor. The method may include maintaining data coherence of the data block residing in the at least one cache using contents of the processor-in-memory directory. The method may include storing a host coherence state of a host copy data block, stored within the host processor. The method may
5 include maintaining data coherence of the host copy data block residing in the host processor using the host directory.

In at least one embodiment of the invention, an apparatus includes a host processor. The host processor includes host bridge translation logic configured to translate coherence messages between a first cache coherence protocol of a processor-in-
10 memory and a second cache coherence protocol of the host processor. The second cache coherence protocol is different from the first cache coherence protocol. The host processor may include a host directory configured to store a host coherence state corresponding to a host copy, stored within the host, of a data block residing in the processor-in-memory. The host processor may include intra-host coherence control
15 logic configured to maintain data coherence of data blocks residing in the host processor using the host directory. The second cache coherence protocol may use block sizes different from block sizes of the first cache coherence protocol.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be better understood, and its numerous objects, features,
20 and advantages made apparent to those skilled in the art by referencing the accompanying drawings.

Figure 1 illustrates a functional block diagram of an exemplary processing system including processor-in-memory devices.

Figure 2 illustrates a functional block diagram of an exemplary processor-in-memory.

25 **Figure 3** illustrates a functional block diagram of an exemplary cache coherent link between a host processor and a processor-in-memory consistent with at least one embodiment of the invention.

Figure 4 illustrates a functional block diagram of exemplary cache coherent links between multiple hosts and various processor-in-memory devices consistent with at least one embodiment of the invention.

Figure 5 illustrates a functional block diagram of an exemplary processor-in-memory including an exemplary processor-in-memory cache organization.

Figure 6 illustrates a state diagram of an exemplary cache coherence bridge protocol between a host processor and a processor-in-memory having a first caching protocol consistent with at least one embodiment of the invention.

Figure 7 illustrates a state diagram of an exemplary cache coherence bridge protocol between a host and a processor-in-memory having a second caching protocol consistent with at least one embodiment of the invention.

The use of the same reference symbols in different drawings indicates similar or identical items.

MODE(S) FOR CARRYING OUT THE INVENTION

A cache coherence bridge protocol provides an interface between a cache coherence protocol of a host processor and a cache coherence protocol of a processor-in-memory, thereby decoupling coherence mechanisms of the host processor and the processor-in-memory. The cache coherence bridge protocol requires limited change to existing host processor cache coherence protocols. The cache coherence bridge protocol may be used to facilitate interoperability between host processors and processor-in-memory devices designed by different vendors and both the host processors and processor-in-memory devices may implement coherence techniques among computing units within each processor. The cache coherence bridge protocol may support different granularity of cache coherence permissions than those used by cache coherence protocols of a host processor and/or a processor-in-memory. The cache coherence bridge protocol uses a shadow directory that maintains status information indicating an aggregate view of copies of data cached in a system external to a processor-in-memory containing that data. In contrast, conventional coherence protocols track the state of a local copy of data in a processor.

A processor-in-memory may be implemented as processing logic within one or more memory dies or as a separate logic die stacked with one or more memory dies.

Referring to Fig. 1, system 100 includes host 110 and four processor-in-memory devices on interposer 112. Processor-in-memory 102 includes processing logic 104,
5 which may be included in a separate die stacked with multiple memory dies 106. Processing logic 104 may include at least one accelerated processing unit (i.e., an advanced processing unit including a central processing unit and a graphics processing unit), central processing unit, graphics processing unit, or other processor and may include coprocessors or fixed-function processing hardware.

10 Figure 2 illustrates exemplary processing logic 104 that includes an advanced processing unit that includes a graphics processing unit, e.g., computing unit cluster 206, which includes multiple graphics processing unit computing units 214. The advanced processing unit may include central processing unit cluster 208, which includes multiple central processing unit cores 216. The advanced processing unit
15 may also include fixed-function hardware 210, which includes multiple hardware accelerators 218. Components within each processor-in-memory 102 (e.g., each graphics processing unit computing unit 214 and each central processing unit core 216 of processing logic 104) may implement hardware cache coherence among them.

Referring back to Fig. 1, host 110 may include multiple processing cores and may
20 implement hardware cache coherence among those processing cores. Host 110 may support cache coherence among multiple host devices via a multi-socket implementation interface (e.g., coherent HyperTransport). Extending any of the conventional cache coherence policies to encompass processor-in-memory requires the host and the processor-in-memory to implement the same cache coherence
25 protocol. That requirement limits the commercial opportunity for processor-in-memory designs by preventing interoperability between host and processor-in-memory designs from different vendors since different vendors have different existing coherence protocols. In addition, supporting high-performance systems with large numbers of memories, each of which may contain a processor-in-memory, requires
30 host and processor-in-memory cache coherence protocols to be scalable to variable numbers (i.e., numbers greater than in systems without processor-in-memory devices

and greater than in conventional systems) of coherent nodes, greatly increasing coherence state tracking overhead of conventional techniques.

A cache coherence bridge protocol provides an interface between existing cache coherence protocols (e.g., a host cache coherence protocol and a processor-in-memory
5 cache coherence protocol). The cache coherence bridge protocol uses host-side support that may include the addition of only a single potential sharing device for any particular cache line in the system of an arbitrary number of processor-in-memory devices, resulting in only a negligible increase in overhead and limited additional states needed independent of the number of processor-in-memory devices in the
10 system. The cache coherence bridge protocol includes a processor-in-memory side representation of all components on the host side by a single additional potential sharer, resulting in only a negligible increase in overhead of the processor-in-memory, as well as limiting the number of additional states needed, independent of the nature and number of hosts in the system.

15 The cache coherence bridge protocol includes a bridge translation unit and a bridge protocol controller implemented on the processor-in-memory. The cache coherence bridge protocol includes a shadow directory stored in the processor-in-memory. The shadow directory stores state information for tracking cache coherence with an outward perspective, i.e., instead of tracking the status of a cached copy on the
20 processor-in-memory, the shadow directory stores bridge coherence states that track an aggregate view of all copies of a data element cached external to the processor-in-memory in which the data element resides. The shadow directory stores a subset of the information that may be contained in a host directory, but from the perspective of the processor-in-memory. In at least one embodiment, the cache coherence bridge
25 protocol uses a subset of cache coherence messages already generated by conventional cache coherence protocols, thus limiting modifications to adapt previously developed and validated cache coherence protocols that are likely to be implemented on a host and within the processor-in-memory.

Referring to Fig. 3, in at least one embodiment of the cache coherence bridge
30 protocol, intra-host coherence controller 302 generates cache coherence messages to processor-in-memory 102 as if processor-in-memory 102 is just another node in the

host coherence protocol. Bridge translation unit 304 may translate those cache coherence messages into bridge protocol coherence messages. In processor-in-memory 102, bridge translation unit 316 may translate received cache coherence messages from the host coherence protocol or an intermediate bridge coherence protocol into a form usable by bridge protocol control logic 322. In the opposite direction, bridge protocol control logic 322 generates cache coherence messages consistent with the host coherence protocol or an intermediate bridge coherence protocol for the host based on contents of shadow directory 324, messages from intra-processor-in-memory coherence controller 320 may be based on contents of directory 326, and/or memory requests received or generated by processor-in-memory 102.

Bridge translation unit 316 translates those cache coherence messages received from bridge protocol control logic 322 into bridge protocol messages or into host coherence controller protocol messages. Bridge translation unit 304 of host 110 may translate the cache coherence messages from the bridge protocol into a host protocol. Although illustrated as separate logic, bridge translation unit 304 may be implemented as an integral part of memory interface controller 308 and bridge translation unit 316 may be implemented as an integral part of memory interface controller 314. In at least one embodiment of the cache coherence bridge protocol, intermediate cache coherence messages are not used and bridge translation unit 304 and 316 translate cache coherence messages directly into processor-in-memory cache coherence protocol messages or host cache coherence protocol messages based on predetermined protocol formats that may be configured upon reset of the system or in response to other user configuration.

Bridge protocol control logic 322 observes intra-processor-in-memory coherence messages and in some cases intervenes in, memory accesses (e.g., cache misses) by host 110, of the one or more memory channels of processor-in-memory 102. Since bridge protocol control logic 322 is implemented on processor-in-memory 102, bridge protocol control logic 322 has visibility into all accesses that occur within the memory channels of processor-in-memory 102 (those generated internally within processor-in-memory 102 as well as those generated externally by host 110 or other devices) without generating additional off-device traffic. Functionality of bridge protocol control logic 322 is described further below.

Referring to Fig. 4, an exemplary system including the cache coherence bridge protocol, includes multiple processor-in-memory devices and host 410 and host 412, which are coupled by host-to-host coherent link 411. Processor-in-memory 402 and 406 are coupled to host 410 by cache coherent bridge protocol link 403 and cache
5 coherent bridge protocol link 405, respectively. Processor-in-memory 404 and processor-in-memory 408 are coupled to host 412 by cache coherent bridge protocol link 407 and cache coherent bridge protocol link 409, respectively. As referred to herein, with regard to a particular memory address, a home processor-in-memory is the processor-in-memory that stores data for the particular address. As referred to
10 herein, a home node is the host attached to the home processor-in-memory for the particular address. The host coherence protocol directory may be distributed such that the home node for coherence directory purposes is the node attached to the processor-in-memory holding the in-memory copy of the data. As referred to herein, a data block is the granularity of coherence tracking between a host and processor-in-
15 memory. The data block is an integer multiple of the granularity of intra-host and host-to-host coherence tracking. In at least one embodiment of a system including the cache coherence bridge protocol, host 410, host 412, processor-in-memory 402, processor-in-memory 404, processor-in-memory 406, and processor-in-memory 408 implement a write-allocate cache coherence policy (i.e., the host or processor-in-
20 memory reads the cache line from memory into a corresponding cache before writing any part of that cache line).

As described above with reference to Fig. 3, the cache coherence bridge protocol uses host-side support, which may include addition of only a single sharing device to host
25 directory 328. That single sharing device in host directory 328 may be used to track the cache coherence state of a home processor-in-memory from the perspective of the host coherence protocol. Referring back to Fig. 4, for example, for a cache line for which processor-in-memory 402 is the home processor-in-memory and that is tracked in a host coherence directory on home node host 410, the host coherence directory on host 410 tracks the cache coherence state of the cache line in processor-in-memory
30 402. The shadow directory at each processor-in-memory maintains an aggregate view of copies of any data block that may exist elsewhere in the system. In at least in one embodiment, for a cache block whose home processor-in-memory is processor-in-

memory 402, the shadow directory on processor-in-memory 402 maintains an aggregate view of any copies of the cache line that may exist in the caches of host 410 or host 412.

In general, a processor-in-memory cache may be either a write-through cache or a write-back cache. If the processor-in-memory cache is a write-through cache, then the processor-in-memory cache need not be checked in response to a memory access by a host. If the processor-in-memory cache is a write-back cache, then servicing a host memory access includes checking the processor-in-memory cache for most recent data. For example, referring to Fig. 5, the level-one caches are write-through caches in a coherence domain on a processor side of memory interconnect 204 and the level-two caches are memory-side caches in the path to memory, external to the coherence domain on the processor side of memory interconnect 204. Every memory access by the host checks the memory-side caches (e.g., level-two caches) external to the processor-in-memory cache coherence domain. In other embodiments, processing logic 104 does not include caches and therefore does not implement intra-processor-in-memory coherence.

Referring back to Fig. 3, each processor-in-memory 102 includes bridge protocol control logic 322 that tracks a bridge coherence state of each data block stored within memory of that processor-in-memory 102. That bridge coherence state is stored in shadow directory 324. Bridge protocol control logic 322 controls state transitions in shadow directory 324 and stores a corresponding indicator. Referring to Figs. 3 and 6, in at least one embodiment, for each data block in the processor-in-memory, shadow directory 324 stores an indicator of the following bridge coherence states: *no-host-copy (I)* state 602, *host-read (S)* state 604, and *host-written (M)* state 606. The state diagram corresponds to processor-in-memory 102 having a write-through level-one cache and a memory-side (e.g., level-two) cache outside the processor-in-memory cache coherence domain. Bridge protocol control logic 322 checks the memory-side cache in response to each access to memory (e.g., either from processor-in-memory 102 or from host 110). The *no-host-copy (I)* state 602 indicates that none of the hosts in the bridge cache coherence domain have stored any part of the data block in a cache memory. This bridge coherence state is analogous to an *invalid* state in conventional *MSI*, *MESI*, *MOESI* cache coherency protocols, where *M* represents

modified, *O* represents *owned*, *E* represents *exclusive*, *S* represents *shared*, and *I* represents *invalid*. The *host-read (S)* state 604 indicates that one or more of the hosts in the bridge coherence domain have read some part of the data block and may have an unmodified copy in their cache(s), but have not written it. This bridge coherence state is analogous to a *shared* state in *MSI*, *MESI*, *MOESI* cache coherency protocols. The *host-written (M)* state 606 indicates that one or more of the hosts in the bridge coherence domain have exclusive ownership of the data block and may have written some part of it (e.g., a host-side core has a copy of the line in *modified* state for *MSI* cache coherency protocol, in *modified* state or *exclusive* state in *MESI* cache coherency protocol, or in *modified* state, *owned* state, or *exclusive* state for *MOESI* cache coherency protocol).

A read of local data (i.e., data associated with a memory address for which processor-in-memory 102 is the home processor-in-memory) by processor-in-memory 102 and a write of local data by processor-in-memory 102 are indicated by *P.r* and *P.w*, respectively and bridge protocol control logic 322 responds with an associated state transition of a corresponding entry in shadow directory 324. A read of processor-in-memory local data by host 110 and a write of processor-in-memory local data by host 110 are indicated by *H.r* and *H.w*, respectively, and bridge protocol control logic 322 responds with an associated state transition of a corresponding entry in shadow directory 324. Intermediate states and actions on transitions are not illustrated in the state diagram for clarity.

In response to a processor-in-memory read access to a local data block (*P.r*), if the data block has *no-host-copy (I)* state 602 or *host-read (S)* state 604 in shadow directory 324, bridge protocol control logic 322 does not generate any coherence messages outside the processor-in-memory and remains in the current state, *no-host-copy (I)* state 602 or *host-read (S)* state 604. If the data block has the *host-written (M)* state 606 in the shadow directory, bridge protocol control logic 322 generates a read coherence request to the home node attached to the processor-in-memory and transitions (704) the coherence state to *host-read (S)* state 604. Since that host is also the home node, this will integrate with the host's coherence protocol as if it were any other read request. Processor-in-memory must wait until the coherence request

completes, receives a coherence acknowledgement, and updated data are available, before proceeding with the read.

In response to a processor-in-memory write access to local data (*P.w*), if the data block has *no-host-copy (I)* state 602 in shadow directory 324, bridge protocol control logic 322 does not generate any coherence messages outside the processor-in-memory and remains in current state *no-host-copy (I)* state 602. If the data block has *host-read (S)* state 604 in shadow directory 324, bridge protocol control logic 322 generates an invalidation request to the host and waits for a completion signal before proceeding with the write and transitions (702) to *no-host-copy (I)* state 602. If the data block has *host-written (M)* state 606 in shadow directory 324, bridge protocol control logic 322 generates an invalidation request, waits for modified data, and then resumes state transition (707) to *no-host-copy (I)* state 602.

In response to a read access from the host side that may have originated on the host or another component accessing the memory through the host, (*H.r*), if the data block has *no-host-copy (I)* state 602 or *host-written (M)* state 606 in shadow directory 324, bridge protocol control logic 322 transitions (702 or 704, respectively) the state to *host-read (S)* state 604. Note that additional read requests from the host side while in *host-written (M)* state 606 imply that the modified copy in a host cache (which led to the *host-written (M)* state 606) has already been demoted by the host-side cache coherence protocol. If the data block has *host-read (S)* state 604 in shadow directory 324, bridge protocol control logic 322 makes no state change.

In response to a read for ownership access from the host side (*H.w*), if the data block has a *no-host-copy (I)* state 602 or *host-read (S)* state 604 in the shadow directory, bridge protocol control logic 322 transitions (706 or 711, respectively) the state to *host-written (M)* state 606. In addition, bridge protocol control logic 322 invalidates any copies of that data block in any cache in the processor-in-memory cache coherence domain (e.g., level one cache) and responds to the read for ownership access with the requested data. If the data block has a *host-written (M)* state 606 in shadow directory 324, bridge protocol control logic 322 makes no change to the state and responds to the read request with the requested data.

A home node may perform additional host coherence operations. For example, for every state transition from *I* or *S* to *E*, *M* or *O* in *MSI*, *MESI* and *MOESI* cache coherency protocols (or equivalent transitions in other protocols), a home node must generate a *transition-to-M* request to the home processor-in-memory (e.g., processor-in-memory 102). This may be triggered by modifying the host protocol to assume all unallocated lines are in *host-read (S)* state 604 on the processor-in-memory by default. The *transition-to-M* request may be in conjunction with a read request (for allocation) preceding a write if the memory interface protocol supports doing so. In at least one embodiment, the home node generates a *transition-to-I* request to the processor-in-memory in response to a last cached copy being evicted from the entire host-side cache coherence domain. That request may be generated in conjunction with flushing out of dirty data on a final eviction from the host side. This request generation may impact hosts with memory-side last-level caches, as discussed further below. In other embodiments, the host does not generate such request.

Referring back to Fig. 5, in at least one embodiment, processor logic 104 includes processor-side (e.g., the level one caches of GPU CUs 214, CPU cores 216, and hardware accelerators 218) write-back caches. As a result, the bridge coherence state of each data block also indicates whether any part of the data block has been modified by processor-in-memory 102 in its processor-side caches using an additional state, *processor-in-memory-modified (P)* state. Figure 7 illustrates a modified state diagram for bridge protocol control logic 322 updates to bridge coherence state information in shadow directory 324. If the data block has *processor-in-memory-modified (P)* state 608 in shadow directory 324, then, in response to any read or write access from outside the processor-in-memory, bridge protocol control logic 322 provides the updated data from a processor-in-memory cache, invalidates any local copy, if needed (if external request is for exclusive or write permissions), and modifies the bridge coherence state of the data block appropriately via transition 710 or transition 717, respectively. In at least one embodiment, when a modified copy of a particular data block in a processor-side cache of the processor-in-memory is evicted from the cache (*P.e*), the state of the particular data block transitions to *no-host-copy (I)* state 602 (715) in shadow directory 324.

In at least one embodiment, the home node generates a *transition-to-I* request to the home processor-in-memory in response to a last cached copy of a particular data block being evicted from the entire host-side cache coherence domain. If the particular data block has *host-read (S)* state 604 or *host-written (M)* state 606 in the shadow directory 324, it transitions to *no-host-copy (I)* state 602 in shadow directory 324. That request may be generated in conjunction with flushing out of dirty data on a final eviction. This request generation may impact hosts with memory-side last-level caches. Such request is optional in the cache coherence bridge protocol (and not illustrated on Fig. 7), and in other embodiments, the home node does not generate such request.

The cache coherence bridge protocols described above are independent of the granularity of coherence tracking. The cache coherence bridge protocol may have a cache-line granularity or may have a coarser granularity (e.g., region coherence, where the region includes more than one cache line). Accordingly, cache coherence bridge protocol implementations having coarser granularities require less state storage and smaller shadow directories. The cache coherence bridge protocols described herein are also implementable with a variable granularity. For example, a portion of the shadow directory may be used to track data blocks having a coarser granularity while another portion of the shadow directory may be used to track data blocks having a finer-granularity.

In at least one embodiment of a cache coherence bridge protocol, coherence granularity of a host coherence protocol is different from the coherence granularity of the cache coherence bridge protocol and the coherence granularity of a processor-in-memory. For example, region coherence, which tracks coherence permissions at a coarser granularity (e.g., 4KB) than individual cache lines, may greatly reduce coherence traffic (e.g., coherence traffic between a host and a processor-in-memory). Accordingly, a bridge coherence protocol may operate at a coarse granularity (e.g., 4KB) while an existing host-side coherence protocol continues to operate at a conventional cache line granularity (e.g., 64B). In addition, support of cache coherence bridge protocol of different coherence granularities of a host coherence protocol and a processor-in-memory cache coherence protocol facilitates interoperability among components from different vendors (e.g., products having

different cache line sizes). To improve efficiency, coarser coherence granularities within a system are multiples of finer coherence granularities within that system, which is typical since conventional systems use coherence granularities that are a power-of-two number of bytes.

- 5 In at least one embodiment, a bridge translation unit translates a cache coherence request from a coarse-grain coherence permission request (e.g., a 4KB region request from a processor-in-memory) to a finer-grain coherence permission request (e.g., 64B cache lines to the host). That translation spans the entirety of the address range of the coarse request (e.g., sixty four 64B requests to the host in response to a single 4KB
10 request from the processor-in-memory) and issues the requests to a corresponding coherence controller on the receiving side. The receiving bridge translation unit then collects responses to all requests within that set before signaling completion to the requesting bridge translation unit.

- In at least one embodiment, either a sending bridge translation unit or receiving bridge
15 translation unit translates from a fine-grain permission request (e.g., a 64B request from a host) to a responder that tracks coarse permissions (e.g., 4KB regions at a processor-in-memory). The sending bridge translation unit or receiving bridge translation unit converts the request to the coarse-grain region that encompasses the address range of the original request. Note that a requestor (e.g., host) may be
20 unaware that it received coherence permissions for a larger region than it requested and may continue to issue additional requests within the same region. In order to reduce inefficiency, the sending bridge translation unit may maintain a cache of recently received region permissions. If subsequent fine-grain requests from the same requestor match entries in this region permission cache, the sending bridge translation
25 unit may signal completion without initiating a request to the other entities.

- In systems that generate messages on coherence demotions at a fine granularity, as described above, the sending bridge translation unit may discard demotion messages since they are only used to enhance performance. In other embodiments, at a fine
30 granularity, the sending bridge translation unit collects demotion messages until a contiguous coarse grain region is demoted before communicating the demotion to a recipient that tracks coherence at a coarse granularity. The sending bridge translation

unit may communicate fine-grain demotions and expect the receiving bridge translation unit to aggregate them into coarse-grain regions. The sending bridge translation unit that generates the demotion requests to other parts of the coarse-grain region may force demotion of a full coarse-grain region and communicate the region demotion. In at least one embodiment, the sending bridge translation unit may perform one or more of the options described above according to heuristics based on prior memory requests.

Referring back to Fig. 3, shadow directory 324 may be implemented using a separate storage structure (e.g., SRAM either in processor-in-memory 102 in a separate die or distributed across stacked with processor-in-memory 102) to store the bridge coherence state of each data block within the processor-in-memory. That storage structure may be physically distributed across multiple memory channels and each memory channel of the processor-in-memory may maintain a separate table, which may alleviate bandwidth limitations to this structure.

In at least one embodiment, shadow directory 324 is integrated with the memory and stores a bridge coherence state for each data block in a subset of the bits of the same row of memory that stores the data block, e.g., in a design that allows a DRAM to be customized to include sufficient additional bits and does not require non-standard addressing support. In at least one embodiment, shadow directory 324 is located in a subset of the memory space of each memory channel dedicated to store bridge coherence state separately from the data blocks (e.g., an upper-most address range of the memory space). The address mapping may be designed such that the cache coherence state for a particular memory address always resides on a channel other than the one that contains the data for the particular memory address to reduce DRAM page conflicts. Provided the address space is distributed among processor-in-memory devices and among channels within a processor-in-memory at a relatively fine grain (i.e., at a granularity much less than the capacity of a single memory channel), that mapping technique may introduce relatively little special addressing support. A shadow directory implemented using DRAM and the subset of memory space alternatives described, may be augmented with an SRAM cache to store recently accessed shadow directory entries to reduce DRAM accesses needed to access and modify shadow directory states.

From a processor-in-memory perspective, a memory-side, last-level cache of the host that is not included in a host-side coherence domain is viewed as part of the host. The cache coherence bridge protocol captures the states of data blocks cached on the host by default in the shadow directory tracking scheme described above. The cache
5 coherence bridge protocol may modify the memory-side last-level cache as if it were any other host cache in response to cache coherence requests from the processor-in-memory, including requests for modified data from host caches, if the memory-side cache is the only one that contains that data, i.e., no copies of the data exist in other caches tracked by the host directory. From the host perspective, the memory-side
10 cache requires no special treatment from the cache coherence bridge protocol. However, if evictions from a directory-tracked host cache are signaled to the processor-in-memory, as described above, these messages should appropriately account for data in host memory-side caches. Those messages do not communicate demotions to the processor-in-memory if the state in the memory-side cache is
15 inconsistent with those messages. Optionally, the cache coherence bridge protocol generates such messages in response to a data block eviction from the memory-side last-level cache.

Thus a cache coherence bridge protocol has been described. Structures described herein may be implemented using software executing on a processor (which includes
20 firmware) or by a combination of software and hardware. Software, as described herein, may be encoded in at least one tangible (i.e., non-transitory) computer readable medium. As referred to herein, a tangible computer-readable medium includes at least a disk, tape, or other magnetic, optical, or electronic storage medium. The description of the invention set forth herein is illustrative, and is not intended to
25 limit the scope of the invention as set forth in the following claims. For example, while the invention has been described in an embodiment in which processor-in-memory devices include caches, one of skill in the art will appreciate that the teachings herein can be utilized with simple processor-in-memory devices that do not include caches, but require host caches to be coherent with processor-in-memory
30 accesses. Variations and modifications of the embodiments disclosed herein, may be made based on the description set forth herein, without departing from the scope of the invention as set forth in the following claims.

WHAT IS CLAIMED IS:

1. A system comprising:

a processor-in-memory comprising:

shadow directory storage configured to store a bridge coherence state
of a data block residing in memory of the processor-in-
5 memory, the bridge coherence state indicating a status of any
copies of the data block stored external to the processor-in-
memory;

bridge protocol translation logic configured to translate coherence
messages between a first cache coherence protocol of the
10 processor-in-memory and a second cache coherence protocol of
a host processor, the second cache coherence protocol being
different from the first cache coherence protocol; and

bridge protocol control logic configured to maintain data coherence of
the memory of the processor-in-memory using the coherence
15 messages and contents of the shadow directory storage.

2. The system, as recited in claim 1, wherein the second cache coherence
protocol uses block sizes different from block sizes of the first cache coherence
protocol.

3. The system, as recited in claim 1, wherein the processor-in-memory further
20 comprises:

a stack of integrated circuit dies, the memory of the processor-in-memory
being included in a plurality of memory integrated circuit dies of the
stack of integrated circuit dies, the bridge protocol control logic and
the bridge protocol translation logic being included in a logic
25 integrated circuit die of the stack of integrated circuit dies; and
at least one cache of a corresponding processor, the at least one cache being
within a coherence domain of the processor-in-memory.

4. The system, as recited in claim 1, 2, or 3, wherein the at least one cache is a write-through cache and the bridge coherence state of the data block is one of a no-host-copy state, a host-read state, and a host-written state.

5. The system, as recited in claim 1, 2, or 3, wherein the at least one cache is a write-back cache and the bridge coherence state of the data block is one of a no-host-copy state, a host-read state, a host-written state, and a processor-in-memory-modified state.

6. The system, as recited in claim 1, 2, or 3, wherein the processor-in-memory further comprises:

10 a memory-side cache outside a coherence domain of the processor-in-memory, wherein the bridge protocol control logic is configured to check the status of any copies of the data block stored in the memory-side cache in response to a memory access to the data block;

15 a processor-in-memory directory configured to store a processor-in-memory coherence state of the data block stored within the at least one cache of the corresponding processor; and

an intra-processor-in-memory coherence controller configured to maintain data coherence of the data block stored in the at least one cache using contents of the processor-in-memory directory.

20 7. The system, as recited in claim 1, 2, or 3, further comprising:
the host processor coupled to the processor-in-memory, the host processor comprising:
host bridge translation logic configured to translate coherence
25 messages between the first cache coherence protocol of the processor-in-memory and the second cache coherence protocol of the host processor;

30 a memory-side cache outside a coherence domain of the host and outside a coherence domain of the processor-in-memory, wherein the shadow directory storage captures the status of any copies of the data block stored in the memory-side cache;

a host directory configured to store a host coherence state of a host copy data block, stored within the host processor; and
intra-host coherence control logic configured to maintain data coherence of the host copy data block residing in the host processor using the host
5 directory.

8. A method comprising:

storing, in a shadow directory, a bridge coherence state of a data block
residing in a memory of a processor-in-memory, the bridge coherence
state indicating a status of any copies of the data block stored external
10 to the processor-in-memory; and

translating coherence messages between a first cache coherence protocol of
the processor-in-memory and a second cache coherence protocol of a
host processor, the second cache coherence protocol being different
from the first cache coherence protocol;

15 maintaining data coherence of the memory of the processor-in-memory using
the coherence messages and contents of the shadow directory.

9. The method, as recited in claim 8, wherein the second cache coherence
protocol uses block sizes different from block sizes of the first cache coherence
protocol.

20 10. The method, as recited in claim 8 or 9, wherein maintaining data
coherence of the memory of the processor-in-memory comprises updating the bridge
coherence state of the data block in response to an access to the data block.

11. The method, as recited in claim 8 or 9, wherein maintaining data
coherence of the memory of the processor-in-memory comprises:

25 maintaining coherence of at least one cache of a processor of the processor-in-
memory with the memory of the processor-in-memory, the at least one
cache being within a coherence domain of the processor-in-memory.

12. The method, as recited in claim 8 or 9, further comprising:

checking a memory-side cache of the processor-in-memory in response to an
access of the memory of the processor-in-memory; and
implementing a write-through cache policy from a cache in a coherence
domain of the processor-in-memory to the memory-side cache of the
processor-in-memory, wherein the bridge coherence state of the data
5 block in the shadow directory is one of a no-host-copy state, a host-
read state, and a host-written state.

13. The method, as recited in claim 8 or 9, further comprising:

10 implementing a write-back cache policy from a cache in a coherence domain
of the processor-in-memory to a memory-side cache of the processor-
in-memory, wherein the bridge coherence state of the data block in the
shadow directory is one of a no-host-copy state, a host-read state, a
host-written state, and a processor-in-memory-modified state.

14. The method, as recited in claim 8 or 9, wherein maintaining data

15 coherence of the memory of the processor-in-memory comprises generating
coherence messages in response to a memory access to the data block.

15. The method, as recited in claim 8 or 9, further comprising:

storing a processor-in-memory coherence state of the data block stored within
at least one cache of a corresponding processor; and

20 maintaining data coherence of the data block residing in the at least one cache
using contents of a processor-in-memory directory;

storing a host coherence state of a host copy data block, stored within the host
processor; and

25 maintaining data coherence of the host copy data block residing in the host
processor using a host directory.

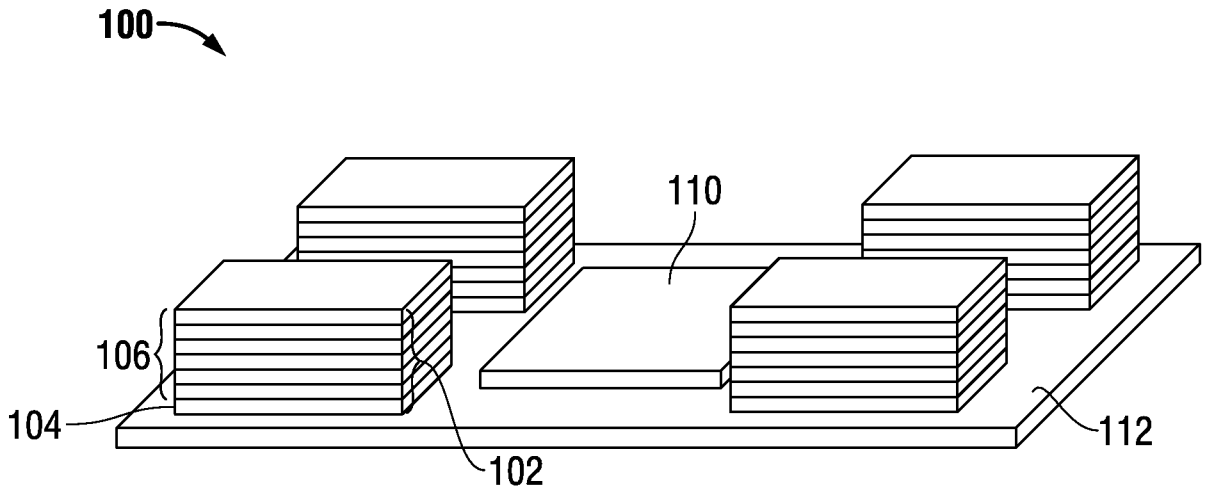


FIG. 1

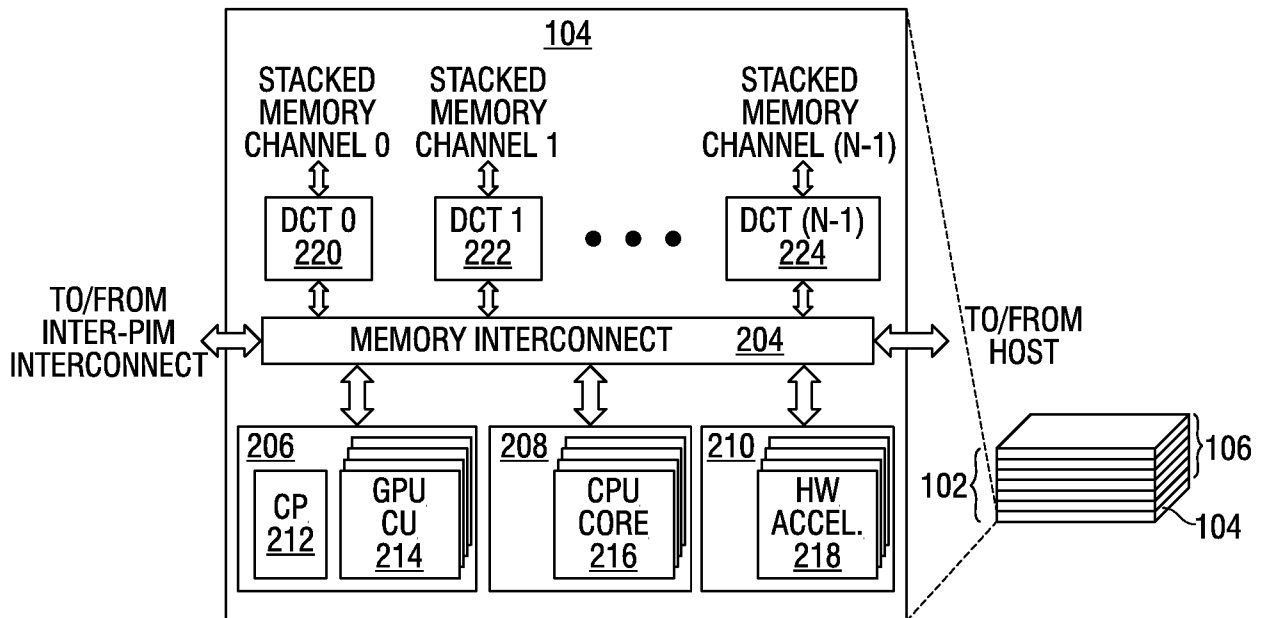


FIG. 2

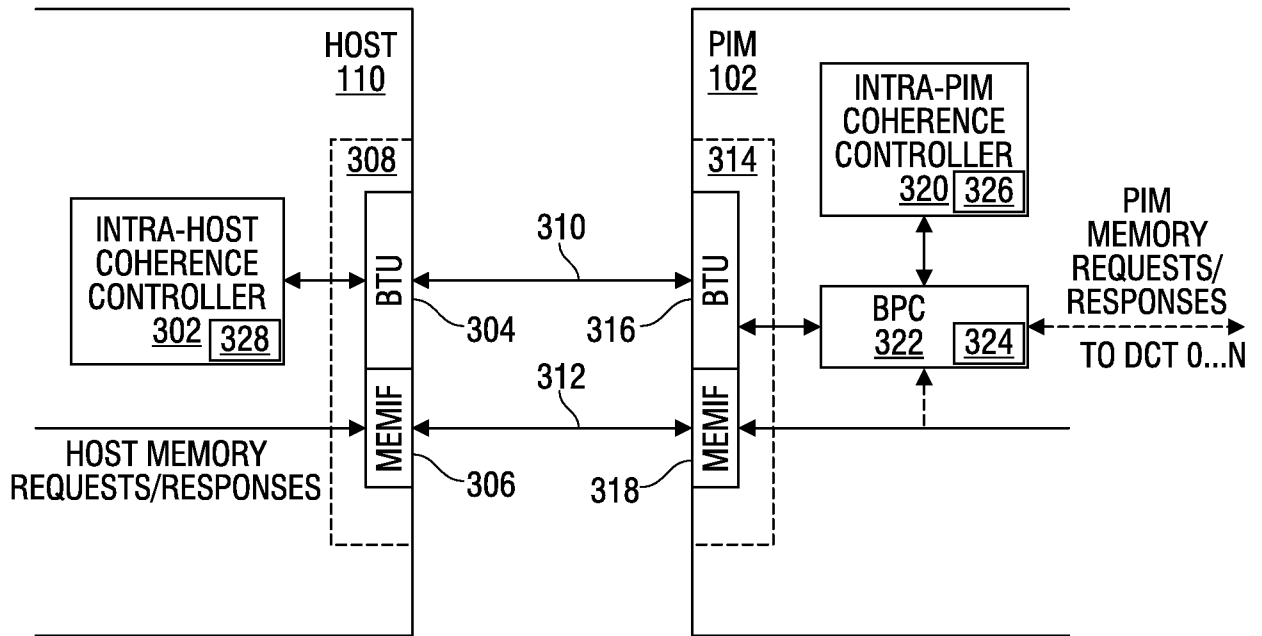


FIG. 3

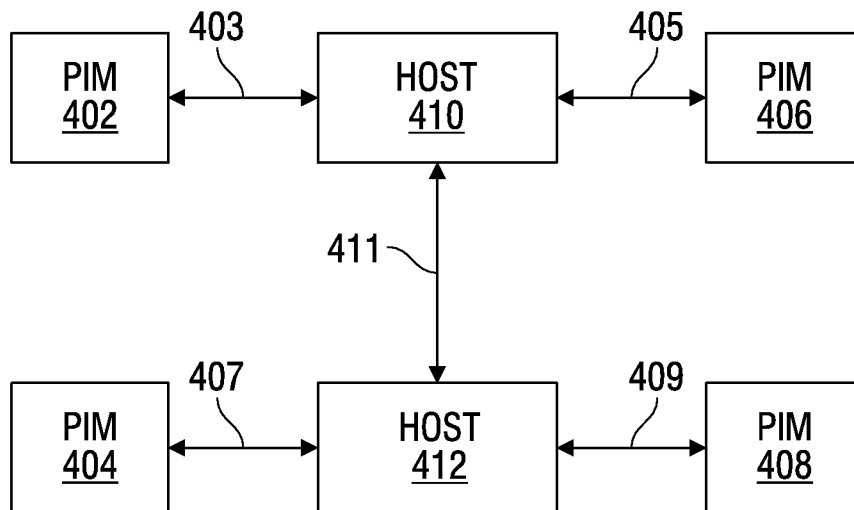


FIG. 4

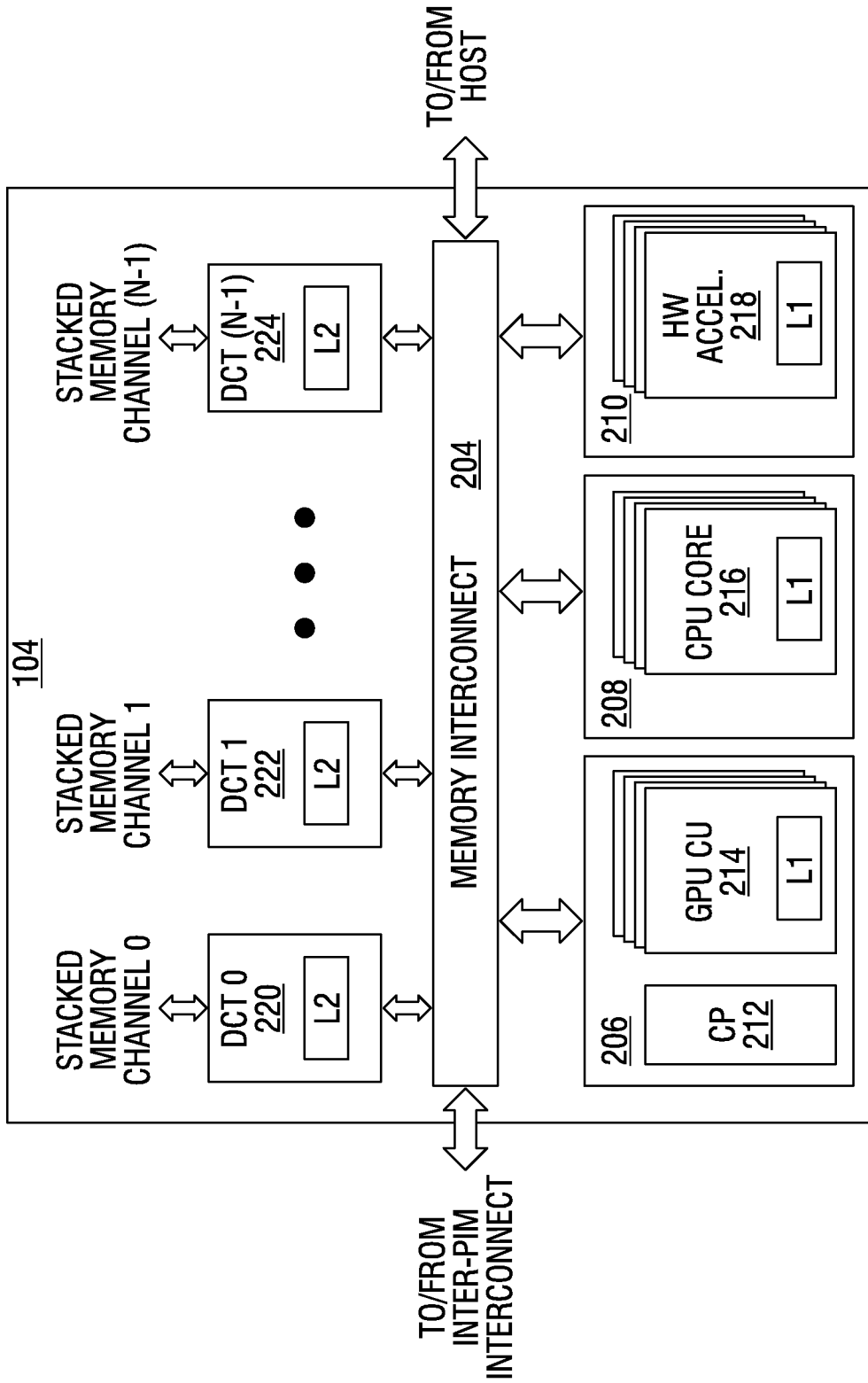


FIG. 5

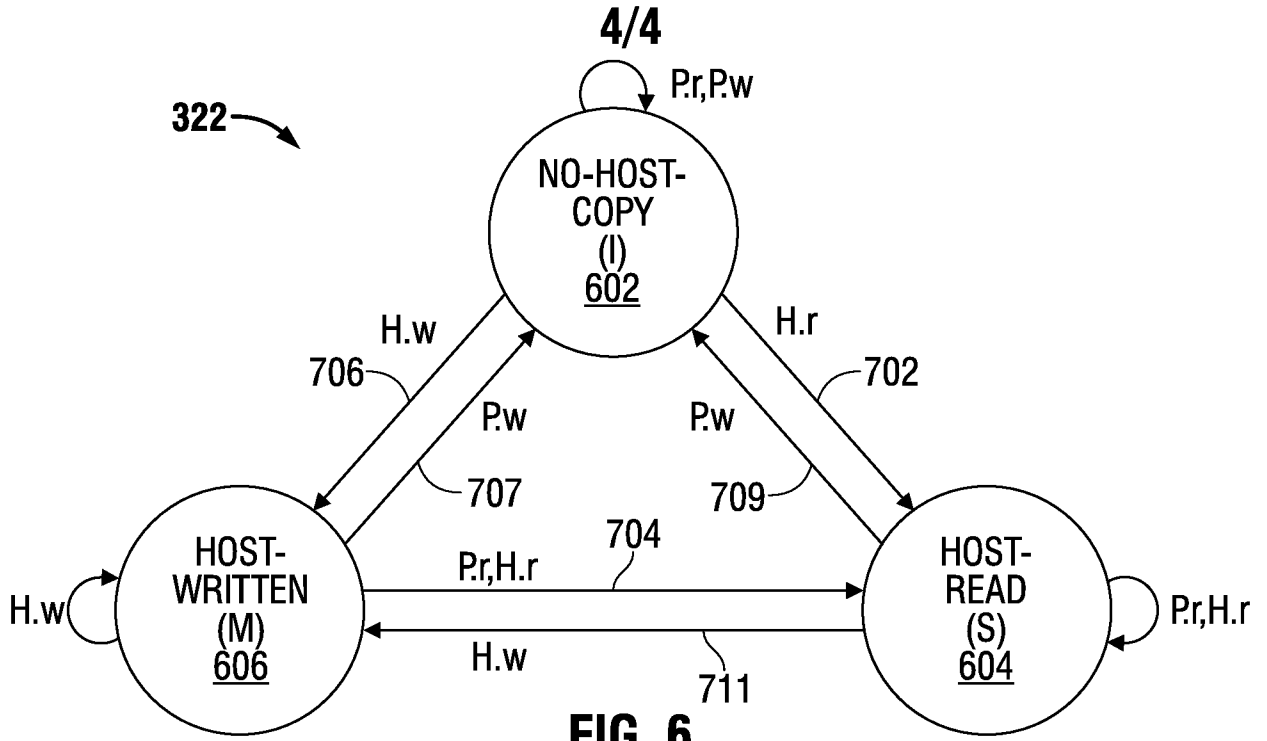


FIG. 6

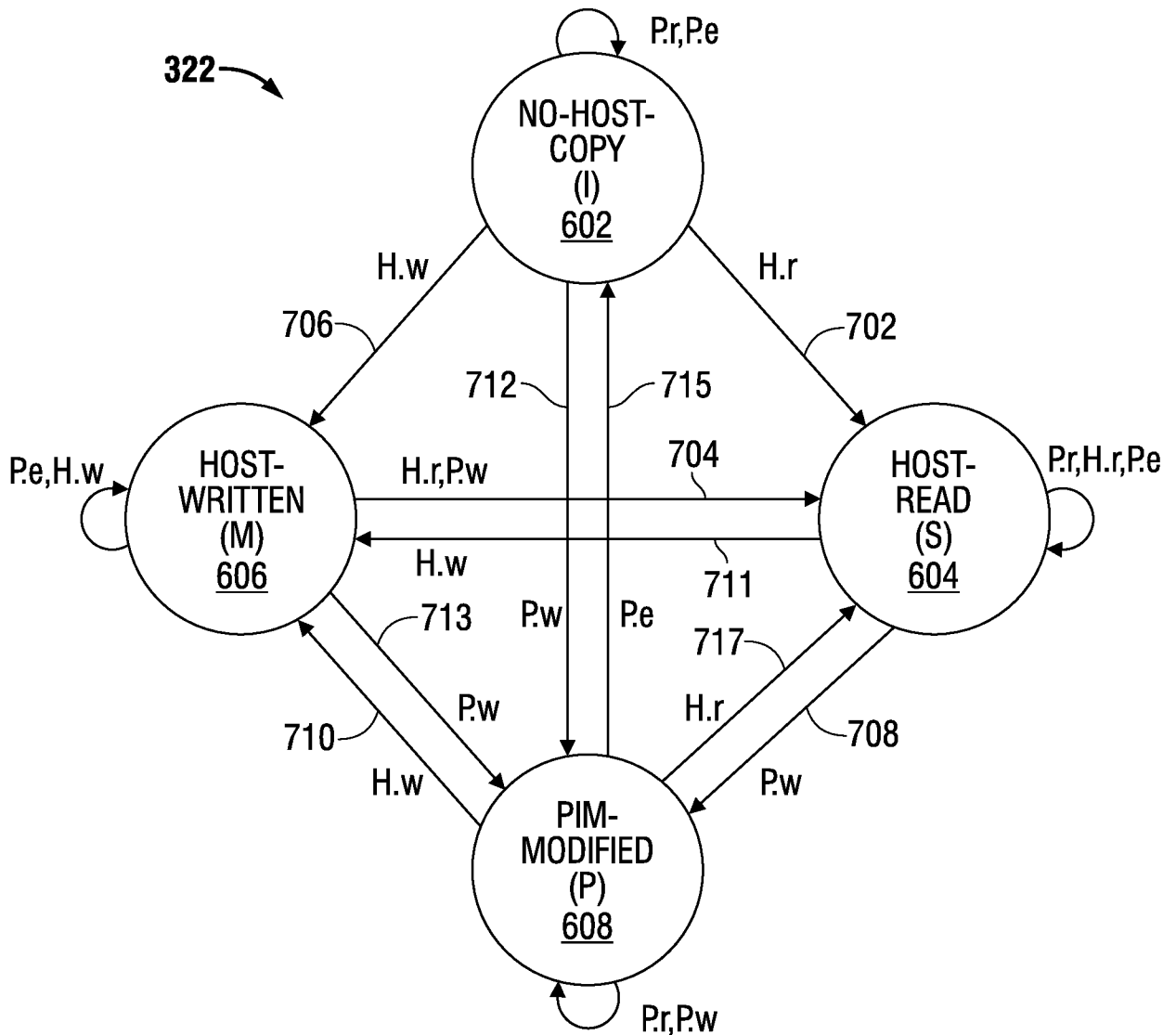


FIG. 7

A. CLASSIFICATION OF SUBJECT MATTER**G06F 12/0811(2016.01)i**

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHEDMinimum documentation searched (classification system followed by classification symbols)
G06F 12/0811; G06F 12/02; G06F 13/16; G06F 12/00; G11C 5/02; G06F 12/08Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility modelsElectronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & keywords: processor-in-memory(PIM), directory, bridge protocol, different cache coherence protocol, host**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2015-0324290 A1 (JOHN LEIDEL et al.) 12 November 2015 See paragraphs [0011]-[0015], [0021]-[0030], [0041]-[0044]; claim 1; and figures 1, 6.	1-15
A	US 2015-0012713 A1 (WILLIAM HENRY FLANDERS et al.) 08 January 2015 See paragraphs [0002], [0056]-[0063]; claim 1; and figures 1-3.	1-15
A	US 2014-0181417 A1 (ADVANCED MICRO DEVICES, INC.) 26 June 2014 See paragraphs [0015]-[0052]; claim 1; and figures 1-5.	1-15
A	US 2005-0160230 A1 (STEPHEN R. VAN DOREN et al.) 21 July 2005 See paragraphs [0018], [0020]-[0043]; claim 1; and figure 1.	1-15
A	US 2015-0325272 A1 (RICHARD C. MURPHY) 12 November 2015 See paragraphs [0012]-[0013], [0042]-[0053]; claims 1-2, 4; and figures 1, 6-7.	1-15

 Further documents are listed in the continuation of Box C. See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

25 July 2017 (25.07.2017)

Date of mailing of the international search report

01 August 2017 (01.08.2017)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

CHIN, Sang Bum

Telephone No. +82-42-481-8398



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2017/030586

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2015-0324290 A1	12/11/2015	CN 106462501 A EP 3140743 A1 KR 10-2017-0002586 A TW 201612755 A WO 2015-171914 A1	22/02/2017 15/03/2017 06/01/2017 01/04/2016 12/11/2015
US 2015-0012713 A1	08/01/2015	CN 104145251 A GB 2514024 A JP 2015-512102 A JP 5941168 B2 KR 10-2014-0131965 A US 9372798 B2 WO 2013-130090 A1	12/11/2014 12/11/2014 23/04/2015 29/06/2016 14/11/2014 21/06/2016 06/09/2013
US 2014-0181417 A1	26/06/2014	US 9170948 B2	27/10/2015
US 2005-0160230 A1	21/07/2005	US 7177987 B2	13/02/2007
US 2015-0325272 A1	12/11/2015	CN 106415522 A EP 3140749 A1 KR 10-2017-0002570 A TW 201612753 A WO 2015-171905 A1	15/02/2017 15/03/2017 06/01/2017 01/04/2016 12/11/2015