

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Appendix 228-01

The following illustrates certain grounds for invalidity:

- U.S. Patent No. 7,805,575 to Agarwal et al. (“*Agarwal*”), which was filed on May 25, 2007, and issued on September 28, 2010, qualifies as prior art to the '228 Patent under at least pre-AIA §§ 102(a), (b), and (e).

As illustrated below, to the extent that *Agarwal* is found not to anticipate, expressly or inherently, one or more of the asserted claims of the '228 Patent, each of claims 1-2, 4-7, and 11-14 is invalid as obvious in view of *Agarwal* alone or in combination with other prior art references, including, but not limited to, the prior art identified in Defendant's Invalidity Contentions, and the prior art described in additional claim charts attached as Appendices 228-02 through 228-08. Exemplary invalidity grounds based on the *Agarwal* reference include the following, although these examples are by no means limiting:

- Anticipation based on *Agarwal* (claims 1-2, 4-7, and 11-14); and
- Obviousness based on *Agarwal* alone (claims 1-2, 4-7, and 11-14).

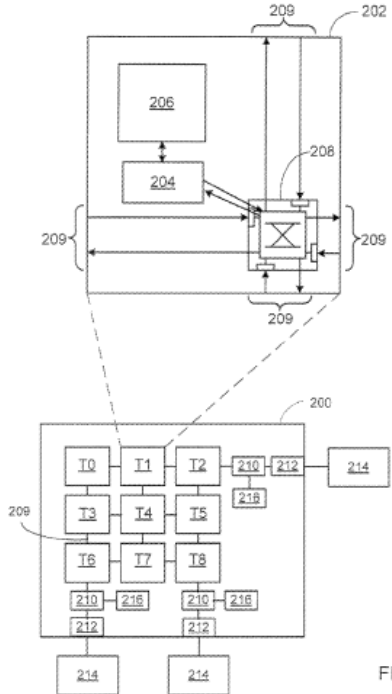
The citations included in the table below are exemplary only and are not intended to be limiting or exhaustive. Defendant reserves the right to cite additional disclosures and evidence in support of their positions as this case develops further. Additionally, citation to a particular figure in a reference should be understood to encompass the caption and description of the figure and any text relating to or discussing the figure. Similarly, citation to any text referring to a figure should be understood to include the figure as well. Additionally, Defendant may be basing the arguments below on one or more claim construction positions that Daedalus has not expressly disclosed but seems to apply in its Patent Local Rule 3-1 disclosures or Defendant's present understanding of Daedalus's view and application of claim scope. By applying the prior art in this way, Defendant does not concede its agreement with Daedalus's positions, nor do these Invalidity Contentions constitute any admission regarding claim scope. Nor is Defendant asserting any claim construction positions through these charts. Moreover, claim preambles generally are not limiting, and nothing in these contentions should be deemed an admission that the preamble in the asserted claims is limiting. As this case further develops, Defendant may amend or supplement these exemplary disclosures and any related contentions as appropriate, and in a manner consistent with all applicable rules governing this case. In particular, Defendant reserves the right to rely on (1) uncited portions of the identified prior art; (2) other prior art not identified herein; (3) references that show the state of the art (irrespective of whether such references themselves qualify as prior art to the asserted patents); (4) factual testimony from the inventors or authors of the prior art references; and/or (5) expert testimony, to provide context to or aid in understanding the prior art and the state of the art at the time of the alleged inventions. Defendant also reserves the right to

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

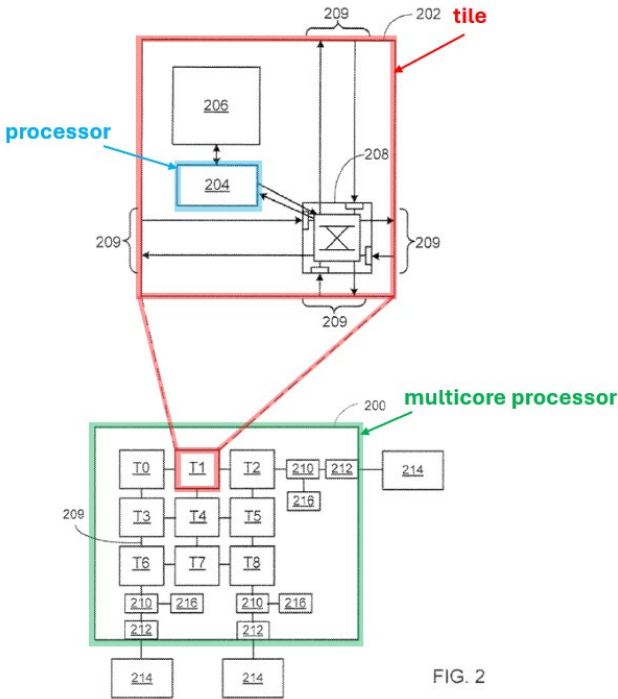
amend or supplement these exemplary disclosures based on changes in Daedalus's positions as the case progresses, including changes to Daedalus's infringement theories, claim interpretations, or apparent claim constructions.

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
[1p] An apparatus comprising:	<p><i>Agarwal</i> discloses or suggests “an apparatus.”</p> <p>For example, <i>Agarwal</i> discloses a multicore processor, which is an apparatus. <i>See, e.g., Agarwal</i> at Abstract; 1:32-33; Fig. 2.</p> <p><i>Agarwal</i> discloses the following at 1:32-33:</p> <p style="padding-left: 40px;">This invention relates to caching in multicore and multiprocessor computers.</p> <p><i>Agarwal</i> also discloses the following at 3:7-24:</p> <p style="padding-left: 40px;">This document relates to caching techniques, some or all of which apply equally well to a discrete system (not single chip) as well as to single chip systems such as a multicore processor. In a multicore processor, multiple nodes are on one chip. In the description below, the terms “multicore processor” are used interchangeably with terms such as “multicore chip,” “tiled processor,” or “chip multiprocessor” (CMP), unless otherwise indicated in their context. In one aspect, in general, a multicore processor comprises a plurality of cache memories; a plurality of processor cores, each associated with one of the cache memories; and a plurality of memory interfaces providing memory access paths from the cache memories to a main memory, at least some of the memory interfaces providing access paths to the main memory for multiple of the cache memories. Each of the memory interfaces is associated with a corresponding portion of the main memory, and includes a directory controller for the portion of the main memory.</p>

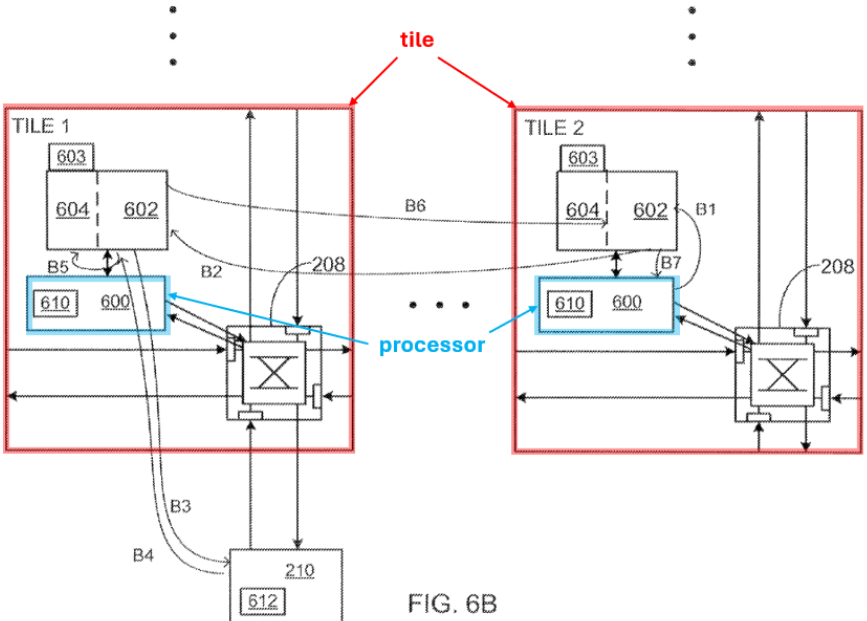
Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> further discloses the following at 7:34-40:</p> <p>As illustrated in an example shown in FIG. 2, each of the nodes or “tiles” of the tiled processor 200 contains a processor and a cache. Each of the tiles does not necessarily correspond to a respective portion of main memory. In this example, a directory state and directory controller are in the path to main memory for each cache miss, so that each cache miss request is able to query the directory.</p> <p><i>Agarwal</i> also discloses the following, Fig. 2:</p>  <p style="text-align: right;">FIG. 2</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[1a] a multicore processor including a plurality of cores,</p>	<p><i>Agarwal</i> discloses or suggests “a multicore processor including a plurality of cores.”</p> <p>For example, as shown in Fig. 2 below, <i>Agarwal</i> discloses a multicore processor 200 that includes nine instances (T0-T8) of tile 202. Each tile 202 includes a processor 204 that corresponds to one of the claimed “plurality of cores.” See <i>Agarwal</i> at 1:32-33, 7:34-37, 11:53-58, Fig. 2. <i>Agarwal</i> further discloses in Fig. 6B that the multicore processor 200 can have tiles including a processor 600, again corresponding to one of the claimed “plurality of cores.” See <i>id.</i> at 20:1-4, Fig. 6B. Thus, <i>Agarwal</i> discloses a multicore processor with multiple cores.</p> <p><i>Agarwal</i> discloses the following, Fig. 2 (annotated):</p>  <p style="text-align: right;">FIG. 2</p>

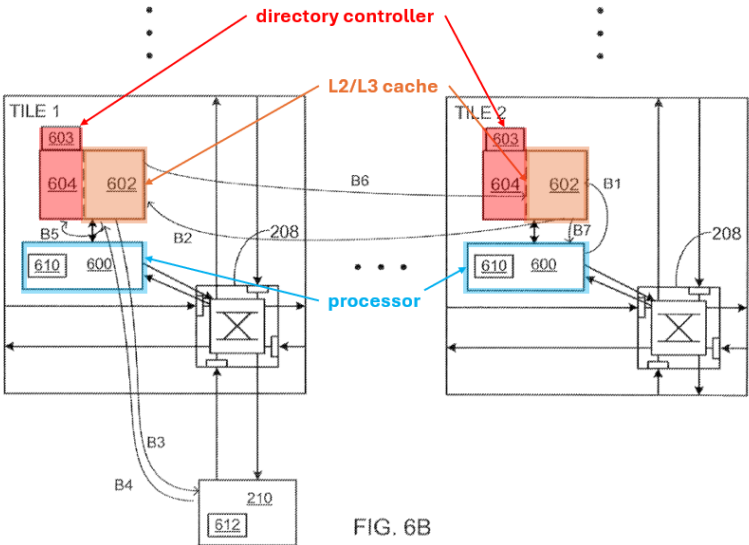
Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	Exemplary Disclosures Relevant to <i>Agarwal</i>
	<p><i>Agarwal</i> discloses the following, Fig. 6B (annotated):</p>  <p>FIG. 6B</p> <p><i>Agarwal</i> discloses the following at 1:32-33:</p> <p style="padding-left: 40px;">This invention relates to caching in multicore and multiprocessor computers.</p> <p><i>Agarwal</i> also discloses the following at 3:7-24:</p> <p style="padding-left: 40px;">This document relates to caching techniques, some or all of which apply equally well to a discrete system (not single chip) as well as to single chip systems such as a multicore processor. In a multicore processor, multiple nodes are on one chip. In the description below, the terms “multicore processor” are used interchangeably with terms such as “multicore chip,” “tiled processor,” or “chip multiprocessor” (CMP), unless otherwise indicated in their context. In one aspect, in general, a</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p data-bbox="772 253 1843 505">multicore processor comprises a plurality of cache memories; a plurality of processor cores, each associated with one of the cache memories; and a plurality of memory interfaces providing memory access paths from the cache memories to a main memory, at least some of the memory interfaces providing access paths to the main memory for multiple of the cache memories. Each of the memory interfaces is associated with a corresponding portion of the main memory, and includes a directory controller for the portion of the main memory.</p> <p data-bbox="695 545 1350 578"><i>Agarwal</i> further discloses the following at 7:34-37:</p> <p data-bbox="772 613 1843 719">As illustrated in an example shown in FIG. 2, each of the nodes or “tiles” of the tiled processor 200 contains a processor and a cache. Each of the tiles does not necessarily correspond to a respective portion of main memory.</p> <p data-bbox="695 760 1367 792"><i>Agarwal</i> further discloses the following at 11:53-59:</p> <p data-bbox="772 829 1843 967">An exemplary multicore processor 200 having a tiled processor architecture is shown in FIG. 2. The figure shows 9 tiles (corresponding to the “processor cores” or simply “cores”) numbered T0-T8, each tile 202 containing a processor 204 (or “processing engine”), a cache 206 and a switch 208.</p> <p data-bbox="695 1008 1335 1040"><i>Agarwal</i> further discloses the following at 20:1-4:</p> <p data-bbox="772 1078 1843 1183">The case illustrated in FIG. 6B is similar to the case illustrated in FIG. 6A, but shows additional levels in the cache hierarchy. The processor 600 in each of the tiles includes an L1 cache 610 that is accessed first.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
[1b] a shared cache memory,	<p><i>Agarwal</i> discloses or suggests “a shared cache memory.”</p> <p><i>Agarwal</i> explains that the cache memory can be configured to be local to the associated core or shared across multiple processor cores. <i>See Agarwal</i> at 10:1-10. For example, Fig. 6B shows that each tile may include an L1 cache 610 within the processor 600 and an L2/L3 cache 602 that includes both private L2 entries and shared L3 entries that can be shared by all cores in the tiles. <i>See id.</i> at 20:1-16, Fig. 6B. As <i>Agarwal</i> further explains, L2/L3 cache 602 can be configured to “serve as both a local L2 cache for the local tile and as a shared L3 cache for other tiles.” <i>Id.</i> at 20:12-16; <i>see also id.</i> at 23:45-51. <i>Agarwal</i> further explains that the shared cache in such a tiled-multicore processor may be based on the spatial arrangement of the tiles. <i>See id.</i> at 27:27-37.</p> <p><i>Agarwal</i> discloses the following, Fig. 6B (annotated):</p> 

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 10:1-10:</p> <p>In another aspect, in general, the caches can be configured to maintain a portion of the cache memory that is local to the associated processor core and to maintain a portion of the cache memory that is shared among multiple processor cores. The portions that are local or shared can be assigned on demand on a cache line bases, for example. State associated with the cache line can indicate whether it is local or shared. This flexibility enables the trade-off of space that is used for local vs. shared cache to be different for different processor cores.</p> <p><i>Agarwal</i> discloses the following at 11:29-41:</p> <p>Another aspect of implementing cache coherence protocols in multicore architectures is managing levels of a hierarchical cache structure. A scheme for providing private local and shared global cache levels in a hierarchy leverages mechanisms such as SNC based load stores and TLB based redirecting of home locations of cache lines. The scheme uses write-through core caches and optional coalescing write buffers to achieve high performance while at the same time keeping the design complexity low. The scheme exploits the relaxed memory consistency model of the architecture to simplify the handling of writes, and treats the existing cache memory simultaneously as both a private local L2 cache and a distributed shared L3 cache. Shared addresses and memory locations are optionally allocated using a special memory allocation function Such as shared malloc.</p> <p><i>Agarwal</i> discloses the following at 18:39-47:</p> <p>In some implementations, as described in more detail below in the section on private/shared cache hierarchy, a cache memory can include some private (L2) cache entries that are copies of data whose home location may be on a different tile, and some shared (L3) cache entries whose home location is on the tile. The directory state stored on the tile may be directory state for the memory locations homed on that tile, and not associated with the private local cache entries.</p>

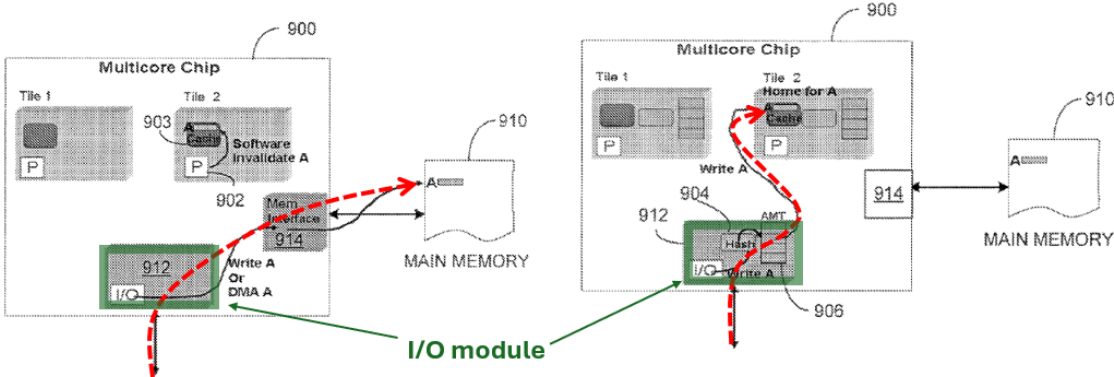
Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 20:1-16:</p> <p>The case illustrated in FIG. 6B is similar to the case illustrated in FIG. 6A, but shows additional levels in the cache hierarchy. The processor 600 in each of the tiles includes an L1 cache 610 that is accessed first. After a miss in the L1 cache 610, action B1 on tile 2 attempts to access the cache 602 on tile 2 as an L2 access. The access of action B2 to the home tile's cache 602 in tile 1 is the L3 access for tile 2 after a miss from the L2 access of action B1. The multicore processor can also include an optional L4 cache 612 in the shim 210 as shown, where the action B3 in this case checks the L4 cache 612 before going to main memory. Thus, the cache 602 can serve as both a local L2 cache for the local tile and as a shared L3 cache for other tiles, as described in more detail below, such that for an access to location a from Tile 1, the cache 60 on tile 1 will serve as the L2 cache for that access.</p> <p><i>Agarwal</i> discloses the following at 23:42-63:</p> <p>The first level (L1) cache within the processor 204 of a tile is a private cache. It maps the entire global PA (physical address) space. The cache 206 coupled to the processor 204 in the tile can be used as a combined shared/private cache. It can be configured to cache data in two ways: (a) as part of a shared distributed cache (equivalent to a shared global/distributed L3 cache), and (b) as a private cache for data requested by this tile but homed in a different tile (equivalent to a private local L2 cache). For the shared distributed cache, each memory line has an assigned home location in the cache 206 of one of the tiles. For the private local cache, private copies of the memory lines homed in a different tile can be stored in the cache 206. The private local cache is “private” in the sense that it is for the private use of the locale tile, however it may store cache lines that contain copies of “shared data” that is shared among the tiles. When a processor in a tile accesses a memory line, if the memory line is not in the private L1 cache, and not in the private (L2) cache portion of the cache 206, the processor determines the home tile location (e.g., using a hash</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>function) to access the data from the shared distributed cache (which may in some cases be on the same tile).</p> <p><i>Agarwal</i> discloses the following at 27:27-37:</p> <p>Constructing hierarchical shared caches in a tiled processor can be based on spatial arrangement of the tiles. For example, in a 64 tile chip, each quadrant could serve as a shared L3 cache for the tiles in each quadrant, and all 64 tiles could serve as a shared L4 cache. This saves directory overhead (16 bits for 64 tiles). In other words, when there is a need to find a home location for a cache line, a home location is found from only 16 tiles out of 64 by using one hash function. Then, if the cache line is not present in those 16 tiles, the hash function is applied again and the resulting cache is examined to see if the home location is there.</p>
<p>[1c] an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor, and</p>	<p><i>Agarwal</i> discloses or suggests “an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor.”</p> <p><i>Agarwal</i> teaches the multicore processor can include an integrated I/O module to interface with an external IO device. <i>Agarwal</i>’s Figures 9A-9C illustrate cache coherence operations of the multicore processor shown in Figures 2 and 6B. As Figures 9A-9C show, multicore processor 900 (equivalent of multicore processor 200 in Fig. 2) includes an I/O module 912 that is integrated onto the same chip as the plurality of processor cores. <i>See Agarwal</i> at 28:11-22, 29:5-7, Figs. 9A-9C. Figures 9A-9C also show L2/L3 cache 903 in each tile (equivalent to cache 602 in Fig. 6B). <i>See id.</i> at 28:44-47. The I/O module “includes an I/O interface 913 such as a network port (e.g., a XAUI interface or a PCI-e interface.” <i>See id.</i> at 28:11-22. <i>Agarwal</i> explains that I/O module 912 supports storing data from an off-chip IO device to the main memory (i.e., direct memory access (DMA) transactions described in Fig. 9B and 28:11-22, 28:33-43) or directly into the L2/L3 cache (i.e., direct cache access (DCA) transactions described in Fig. 9C and 28:23-25, 29:5-7).</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	Exemplary Disclosures Relevant to <i>Agarwal</i>
	<p>Compare annotated Figs. 9B and 9C below:</p>  <p>FIG. 9B</p> <p>FIG. 9C</p> <p><i>Agarwal</i> discloses the following at 10:31-32:</p> <p>FIGS. 9A-9C are diagrams of a multicore chip with an I/O module.</p> <p><i>Agarwal</i> discloses the following at 27:40-46:</p> <p>Some modules on a multicore processor other than the processor cores may have access to memory. An example of Such a module is an I/O module. Maintaining memory coherence in the presence of such modules can be part of an overall coherence protocol. As an extension to the proxy cache coherence approach, a technique for ensuring I/O coherence is called “Proxy IO coherence.”</p> <p><i>Agarwal</i> discloses the following at 28:10-22:</p> <p>5.1 I/O Coherence Examples</p> <p>An I/O module 912 includes an I/O interface 913 such as a network port (e.g., a XAUI interface or a PCI-e interface) and circuitry 915 to enable the module 912 to process direct memory access (DMA) transactions initiated from a tile, for example,</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>or to otherwise handle a transaction to write data into off-chip main memory via a memory interface 914 (for example, into main memory location A). In this situation, if Some tile has the same data location (A) cached in its cache, a coherence protocol ensures that these caches are managed appropriately (e.g., invalidated or updated). This situation is shown in FIG. 9B. This situation is case 1: the “memory coherence case.”</p> <p><i>Agarwal</i> discloses the following at 28:23-32:</p> <p>An I/O module is also able to process a DMA transaction or otherwise store data directly into the cache of a given tile, for example, into some other memory location B. In this case, a coherence protocol handles the situation that the main memory location B will have different data from the cache location B written by the I/O module. Further, if one or more other tiles have the same data location (B) cached in their cache, there is another potential coherence situation to be dealt with. The situation that occurs when the I/O module 912 writes directly into a tile is case 2: the “cache coherence case.”</p> <p><i>Agarwal</i> discloses the following at 28:33-43:</p> <p>Various approaches can be used to maintain coherence for case 1 and/or case 2. One approach to maintaining coherence for case 1 relies on software. To do so, as illustrated in FIG. 9B, the I/O module 912 writes the memory location (for example, A) to main memory 910 via the memory interface 914. Now, suppose Tile 2 needs to consume the data that has just been written to main memory 910 by the I/O module 912. Further, assume that Tile 2 potentially has the location A in its cache as shown. To ensure that Tile 2 does not read Stale data, the location A is invalidated in association with the write transaction from the I/O module 912.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 28:44-47:</p> <p>For example, in some coherence protocols Tile 2 pre-invalidates the copy of the memory location in its cache 903 that is being written by the I/O module 912 (in this example location A) prior to reading the location.</p> <p><i>Agarwal</i> discloses the following at 29:5-16:</p> <p>FIG.9C illustrates an example in which a Proxy IO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile's cache. As part of the provide I/O coherence protocol, the I/O module 912 includes a second mode of operation, as follows. In this mode, the I/O module 912 is configured to deliver write transactions directly to home tiles in a coherent fashion, over the on-chip interconnect (e.g., a dynamic network). This mode of operation is in addition to the first mode described above in which the I/O module 912 writes main memory and software maintains coherence with explicit invalidation of the location in a tile's cache.</p>
<p>[1d] a caching agent to perform cache coherency operations for the plurality of cores and the IIO module,</p>	<p><i>Agarwal</i> discloses or suggests “a caching agent to perform cache coherency operations for the plurality of cores and the IIO module.”</p> <p><i>Agarwal</i> explains that when multiple processors cache data for a given memory location, a cache coherency problem may arise because copies of the same memory location (cache line) may exist in multiple caches. <i>Agarwal</i>, 1:34-39. To address this, <i>Agarwal</i> teaches several cache coherence management techniques, including Shim Cache Coherence (SCC) and Proxy Cache Coherence (PCC) protocols. <i>Id.</i> at 16:26-20:34, 27:39-30:4. Both are directory based protocols used to maintain cache coherence by employing: (1) a directory controller (i.e. coherence controller, <i>id.</i> at 2:47-54, 8:38-42, 1:46-50, 11:12-26,12:28-32) and (2) directory storage that collectively operate as the claimed “caching agent” to perform cache coherency operations for the plurality of cores and the IIO module. <i>See also id.</i> at 1:46-50, 2:21-24. The two approaches differ as to where those components are located. <i>See id.</i> at 11:15-21.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>Under Agarwal's PCC protocol, the directory controller and directory storage are distributed across the tiles in the multicore processor. <i>See Agarwal</i> at 11:22-26. A particular cache operation is managed by one of the tiles in the multiprocessor, referenced as the "home tile" for that cache memory operation. <i>See id.</i> at 16:26-45, 23:51-53. The home tile, including its directory controller and directory storage, performs cache coherency operations as a "proxy" for cache memory locations assigned to that home tile, giving rise to the term Proxy Cache Coherence (PCC). <i>See id.</i> at 16:26-45. The assignment of a home tile to manage a cache operation can be done "based on a function of the address such as a hash function." <i>Agarwal</i> at 17:50-51.</p> <p>Agarwal thus teaches the same type of distributed caching agent as in the '228 patent (i.e. "a caching agent can be distributed such that each of different portions of the caching agent can be associated with a corresponding core and LLC bank of slice." <i>See</i> '228 Patent at 2:66-3:3.). In <i>Agarwal</i>, just like in the '228 Patent, the caching agent portions at each CPU (tile) manage the cache coherency for a portion / slice of the shared cache thus constituting a single caching agent with distributed portions associated with each processor core (tile) that manages cache coherency for accesses by CPUs and I/O components. <i>See Agarwal</i> at 27:39-28:8, 29:3-30:12.</p> <p>For example, Agarwal discloses that the SCC and PCC schemes work in a similar manner. One difference is that with SCC, the directory controller (i.e., coherency controller) and directory storage are centralized (e.g., DCC 216 with directory adjacent to an on-chip memory controller shim, <i>see Agarwal</i> at 11:15-21 and Fig. 2) whereas with PCC, the directory controller and directory storage are distributed in the tiles (e.g., directory controller 603 and directory storage 604 within a core in the multicore system. <i>See Agarwal</i> at 11:11-26, 16:27-34.</p> <p>Agarwal's distributed caching agent portions also manage access to the cache by IO components – in the same manner as taught/claimed in the '228 Patent. In <i>Agarwal</i>, in an extension of the PCC protocol called "ProxyIO coherence," the home tile can manage cache coherence in the presence of an I/O module ("I/O coherence"). <i>See Agarwal</i> at 27:39-28:9; <i>see also id.</i> at 16:26-45, 29:3-30:11. This is the same proxy I/O coherence concept described</p>

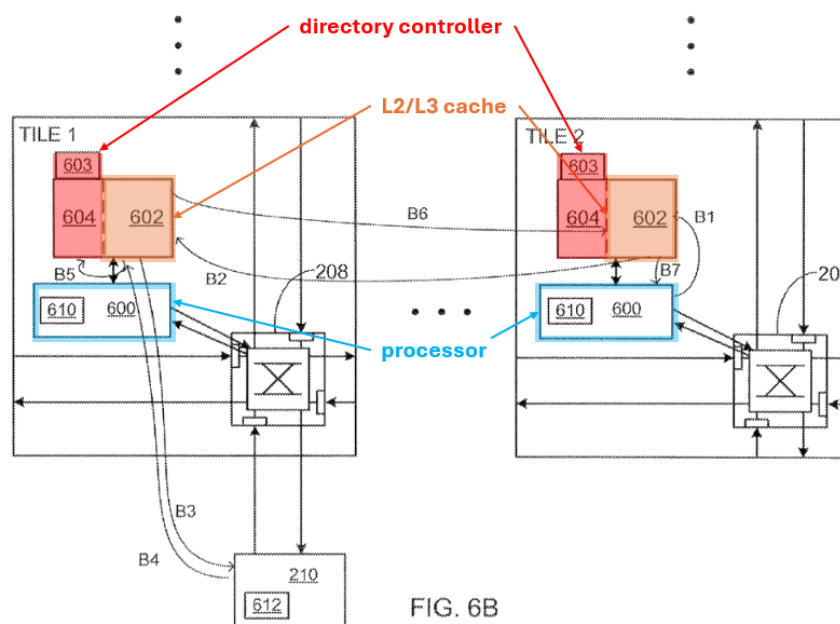
Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1

Exemplary Disclosures Relevant to *Agarwal*

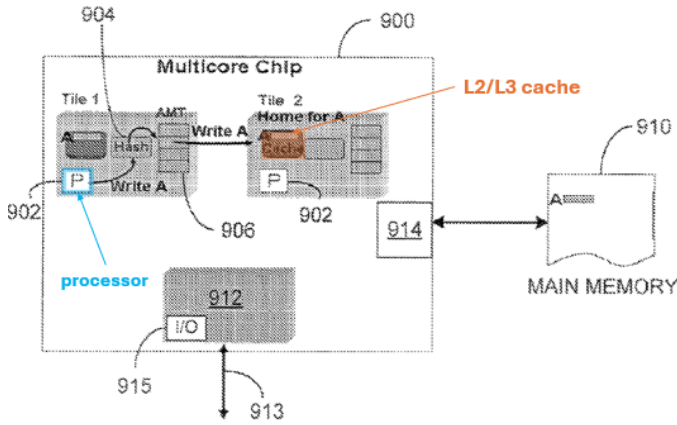
in the '228 Patent, which explains that “by proxying I/O-related transactions through a CPU caching agent as opposed to having separate dedicated IO agents,” “a better scaling profiling and performance advantages can be realized.” '228 Patent at 6:5-7, 6:35-37.

Agarwal depicts the tile architecture configured for the PCC protocol in Figures 6A-6B. Each tile includes a processor 600, L2/L3 cache 602, directory controller 603, and directory storage 604:

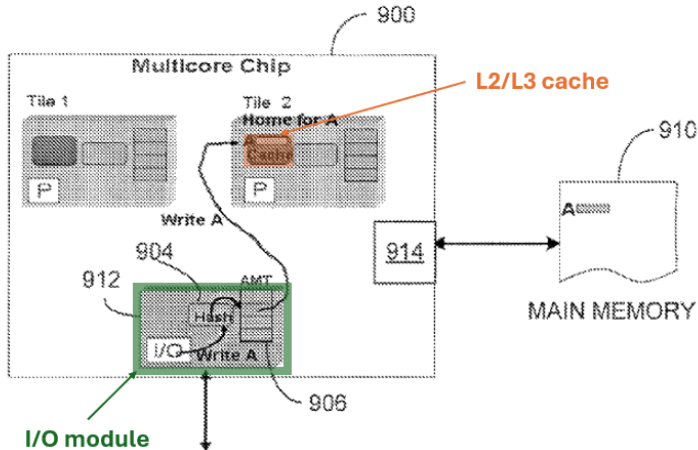


Directory controller 603 and directory storage 604 manages cache coherence for the cache memory locations assigned to that home tile regardless of whether the cache access is from a tile (CPU) or an I/O module. *See Agarwal* at 27:39-28:9, 29:25-29.

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	Exemplary Disclosures Relevant to <i>Agarwal</i>
	<p>Thus, when a given cache line is accessed (for both I/O module access and CPU access to cache), the directory controller and directory storage assigned to the “home tile” for the shared cache partition that includes that cache line manage the coherence. All directory controllers 603 and directory storages 604, collectively for all tiles, thus manage cache coherence for the entire shared cache (and memory space). Thus, <i>Agarwal</i>’s distributed components teach, or render obvious, a “single” caching agent with distributed portions managing cache coherence for the plurality of processor cores and the I/O module in the same manner as the '228 Patent teaches for distributed caching agent partitions managing memory partitions / slices. '228 Patent at 2:66-3:3, 6:20-35, Figs. 2, 6.</p> <p>Fig. 9A describes <i>Agarwal</i>’s cache coherence operations for the plurality of cores. When the processor in Tile 1 writes a data to memory location having address A, Tile 1 sends the write request to the home tile of the address A (Tile 2) based on a hash function and an address mapping table (AMT). <i>See Agarwal</i> at 17:42-67, 19:3-29, 27:46-28:8, 29:17-30:11, 23:6-53, 27:32-38. The home tile (Tile 2) performs cache coherence operations as a proxy and writes the data into the L2/L3 cache in Tile 2. <i>See id.</i> at 27:46-28:8; <i>see also id.</i> at 20:1-16, Fig. 6B.</p>  <p style="text-align: center;">FIG. 9A</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	Exemplary Disclosures Relevant to <i>Agarwal</i>
	<p>Fig. 9C describes <i>Agarwal</i>'s ProxyIO cache coherence operations for the integrated I/O module. <i>Agarwal</i> explains that I/O module 912 includes circuitry implementing a hash function and address mapping table. Thus, I/O module 912 delivers the write transaction to the home tile of the address A (Tile 2) based on a hash function and an address mapping table (AMT). <i>See Agarwal</i> at 29:17-20. Once receiving the IO write transaction, the home tile (Tile 2) performs cache coherence operations as a proxy to I/O module 912 and writes the data into the L2/L3 cache in Tile 2. <i>See id.</i> at 29:5-28, Fig. 9C.</p>  <p style="text-align: center;">FIG. 9C</p> <p><i>Agarwal</i> discloses the following at 1:34-41:</p> <p>Cache coherence is a useful mechanism in multiple processor systems to provide a shared memory abstraction to the programmer. When multiple processors cache a given shared memory location, a cache coherence problem may arise because a copy of the same memory location exists in multiple caches. A cache coherence protocol guarantees that a given memory location has a consistent view across all processors.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 16:27-45:</p> <p>The proxy cache coherence approach or PCC works in a similar manner to the SCC method described above. One difference is that the home location for cache lines are not the shims, but rather other tiles, as illustrated in FIGS. 6A and 6B. In PCC, the tile that serves as a home location for a given cache line is called the proxy tile. Because the proxy tile serves as the proxy for the directory memory, the scheme is called proxy cache coherence or PCC. Referring to FIG. 6A, each of the tiles (e.g., tile 1 and tile 2) that are used as home tiles for a given portion of main memory includes a processor 600 coupled to a cache 602 (e.g., providing an L2 cache to back an L1 cache that is integrated into the processor 600). A directory storage 604 for directory state can be included within the same memory that provides the cache 602, or can be included in a separate memory module on the tile. In some implementations, directory entries can be stored along side cache tags for a given memory line. A directory controller 603 is coupled to the directory storage 604.</p> <p><i>Agarwal</i> discloses the following at 17:50-51:</p> <p>The home tile assigned to a given address can also be based on a function of the address such as a hash function.</p> <p><i>Agarwal</i> discloses the following at 19:30-20:25:</p> <p>When the message is received at tile 1, if the message corresponds to a shared location for which cache coherence must be dealt with, then the message is redirected to the local processor for software handling (e.g., if the software coherence mode bit is set on the tile), or otherwise it is then handled by the directory controller in hardware. Note that coherence actions need not be taken for data that is not marked as one for which coherence is desired.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>The directory controller on tile 1 of the multicore chip then reads the state memory for the directory state corresponding to the cache line. Directory state might be associated with each cache line in the cache of tile 1. (In this case, if the cache line is ever evicted, then the directory controller might cause all the copies of the line on other tiles on the chip to be invalidated).</p> <p>In some cases, the cache of tile 1 contains a copy of that cache line. If the directory states that the line is not dirty in someone's cache, then the directory will add a pointer (or a bit, or some other form of state information) in the directory entry indicating that a new tile (tile 2) now has a copy as well. The directory controller will then (usually) get the cache line data from its own cache (L1 or L2 cache on its own tile) and then send a message with the data to the appropriate tile (tile 2 in this case). This causes a second network message to go through. Note that in the SCC scheme, for read cache misses that are clean everywhere, there are typically two network transactions.</p> <p>...</p> <p>The data comes back to tile 1 from the memory controller (or L4 cache) (action B4), and the directory controller then updates the directory storage 604 with tile 2's address (action B5). The directory then sends the data value to tile 2 cache 602 (action B6), and the cache 602 sends the value to the processor 600 (action B7) and to an optional L1 cache. In this latter case, notice that there are two extra network transactions.</p> <p><i>Agarwal</i> discloses the following at 20:1-16:</p> <p>The case illustrated in FIG. 6B is similar to the case illustrated in FIG. 6A, but shows additional levels in the cache hierarchy. The processor 600 in each of the tiles includes an L1 cache 610 that is accessed first. After a miss in the L1 cache 610, action B1 on tile 2 attempts to access the cache 602 on tile 2 as an L2 access. The access of action B2 to the home tile's cache 602 in tile 1 is the L3 access for tile 2 after a miss from the L2 access of action B1. The multicore processor can also include an optional L4 cache 612 in the shim 210 as shown, where the action B3 in this case checks the L4 cache 612 before going to main memory. Thus, the cache</p>

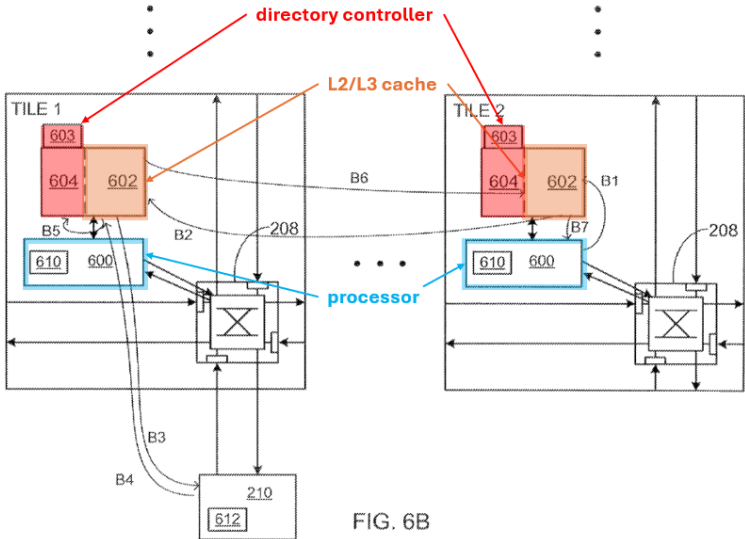
Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>602 can serve as both a local L2 cache for the local tile and as a shared L3 cache for other tiles, as described in more detail below, such that for an access to location a from Tile 1, the cache 602 on tile 1 will serve as the L2 cache for that access.</p> <p><i>Agarwal</i> discloses the following at 23:45-53:</p> <p>The cache 206 coupled to the processor 204 in the tile can be used as a combined shared/private cache. It can be configured to cache data in two ways: (a) as part of a shared distributed cache (equivalent to a shared global/distributed L3 cache), and (b) as a private cache for data requested by this tile but homed in a different tile (equivalent to a private local L2 cache). For the shared distributed cache, each memory line has an assigned home location in the cache 206 of one of the tiles.</p> <p><i>Agarwal</i> discloses the following at 27:39-67:</p> <p>Some modules on a multicore processor other than the processor cores may have access to memory. An example of such a module is an I/O module. Maintaining memory coherence in the presence of such modules can be part of an overall coherence protocol. As an extension to the proxy cache coherence approach, a technique for ensuring I/O coherence is called “Proxy IO coherence.” As a review of proxy cache coherence, FIG. 9A shows a multicore processor 900 in which a processor 902 in Tile 1 is writing a shared cache location having address A (e.g., a cache line, or a data word within a cache line, or a region extending over multiple cache lines) using circuitry 904 implementing a hash function and address mapping table (AMT) 906 to deliver a write transaction to the home location in the cache 906 of Tile 2 using an on-chip interconnect (e.g., a dynamic network). The home tile Tile 2 accepts the write transaction from Tile 1. The home tile performs the normal coherence processing, and sends a write acknowledgement message back to Tile 1 when the write transaction is complete. Normal coherence processing at Tile 2 (home tile) can include the following actions as needed: (1) updating the location A in Tile 2’s cache 906; (2) if A was not in Tile 2’s cache, then fetching the cache line (or at least the background data from the cache line) from main memory 910, storing</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>the background data from that line in its cache 906, and optionally recording Tile 1 as a sharer in the directory entry on Tile 2 associated with location A if the cache 906 of Tile 1 is updated with the written location A (including any background data of the cache line).</p> <p><i>Agarwal</i> discloses the following at 29:5-28</p> <p>FIG. 9C illustrates an example in which a ProxyIO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile's cache. As part of the provide I/O coherence protocol, the I/O module 912 includes a second mode of operation, as follows. In this mode, the I/O module 912 is configured to deliver write transactions directly to home tiles in a coherent fashion, over the on-chip interconnect (e.g., a dynamic network). This mode of operation is in addition to the first mode described above in which the I/O module 912 writes main memory and software maintains coherence with explicit invalidation of the location in a tile's cache. In the second mode, the I/O module 912 includes circuitry 904 implementing a hash function and address mapping table 906 to deliver write transactions to the home tile (Tile 2 in this example). Instead of a hash function and address mapping table, the I/O module 912 can use other techniques that use, for example, one or more bits in the address to find the home location. The home tile accepts the write transactions from the I/O module 912 just as it would write transactions from a tile. The home tile performs the normal coherence processing, with some differences as noted below, and sends a write acknowledgement message back to the I/O module 912 when the write transaction is complete.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	Exemplary Disclosures Relevant to <i>Agarwal</i>
<p>[1e] the caching agent a single caching agent for the multicore processor and including a plurality of distributed portions each associated with a corresponding one of the plurality of cores.</p>	<p><i>Agarwal</i> discloses or suggests “a single caching agent for the multicore processor and including a plurality of distributed portions each associated with a corresponding one of the plurality of cores.”</p> <p>As discussed above in [1d], <i>Agarwal</i> discloses a Proxy Cache Coherence (PCC) architecture system. <i>See generally Agarwal</i> at 16:26-20:34, 27:39-30:4. Under this protocol, a given cache location is assigned to a home tile, such that the shared distributed cache is partitioned based on the number of tiles, and each partitioned cache region is assigned to one home tile. This is just like in the '228 Patent discussed above in which the distributed portions of each caching agent managed specific portions (banks / slices) of the shared LLC cache partitioned between those shared distributed caching agent portions. <i>See supra</i> at [1d]. Each tile has a directory controller 603 and directory storage 604 coupled to a corresponding processor core. <i>See Agarwal</i> at 16:35-45, 19:30-20:25, Fig. 6A-6B. For example, <i>Agarwal</i> discloses the following, Fig. 6B (annotated):</p> <div style="text-align: center;">  <p>FIG. 6B</p> </div>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>Claim 1 requires that the caching agent portions are “each associated with a corresponding one of the plurality of cores.” The '228 Patent teaches that “a caching agent can be distributed such that each of different portions of the caching agent can be associated with a corresponding core and LLC bank or slice.” '228 Patent at 2:67-3:3. Thus, the claimed correspondence can be with the core, not necessarily the bank or slice.</p> <p><i>Agarwal</i> teaches this association explicitly as detailed above for Figs. 6A-6B. For example, <i>Agarwal</i> teaches “[a]t least some of the directory controllers are associated with a processor core....” <i>Agarwal</i> at 3:42-44. Similarly, <i>Agarwal</i> teaches “[e]ach of at least some of the processor cores includes a directory controller...” and “at least some of the storage for directory state.” <i>Id.</i> at 4:7-8, 6:35-36. Each directory controller 603 / directory storage 604 pair (“distributed portion [of the caching agent]”) is “associated with” a processor 600 within each tile (i.e., “corresponding one of the plurality of cores”).</p> <p><i>Agarwal</i> discloses the following at 16:27-45:</p> <p style="padding-left: 40px;">The proxy cache coherence approach or PCC works in a similar manner to the SCC method described above. One difference is that the home location for cache lines are not the shims, but rather other tiles, as illustrated in FIGS. 6A and 6B. In PCC, the tile that serves as a home location for a given cache line is called the proxy tile. Because the proxy tile serves as the proxy for the directory memory, the scheme is called proxy cache coherence or PCC. Referring to FIG. 6A, each of the tiles (e.g., tile 1 and tile 2) that are used as home tiles for a given portion of main memory includes a processor 600 coupled to a cache 602 (e.g., providing an L2 cache to back an L1 cache that is integrated into the processor 600). A directory storage 604 for directory state can be included within the same memory that provides the cache 602, or can be included in a separate memory module on the tile. In some implementations, directory entries can be stored along side cache tags for a given memory line. A directory controller 603 is coupled to the directory storage 604.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 19:30-20:25:</p> <p>When the message is received at tile 1, if the message corresponds to a shared location for which cache coherence must be dealt with, then the message is redirected to the local processor for software handling (e.g., if the software coherence mode bit is set on the tile), or otherwise it is then handled by the directory controller in hardware. Note that coherence actions need not be taken for data that is not marked as one for which coherence is desired.</p> <p>The directory controller on tile 1 of the multicore chip then reads the state memory for the directory state corresponding to the cache line. Directory state might be associated with each cache line in the cache of tile 1. (In this case, if the cache line is ever evicted, then the directory controller might cause all the copies of the line on other tiles on the chip to be invalidated).</p> <p>In some cases, the cache of tile 1 contains a copy of that cache line. If the directory states that the line is not dirty in someone's cache, then the directory will add a pointer (or a bit, or some other form of state information) in the directory entry indicating that a new tile (tile 2) now has a copy as well. The directory controller will then (usually) get the cache line data from its own cache (L1 or L2 cache on its own tile) and then send a message with the data to the appropriate tile (tile 2 in this case). This causes a second network message to go through. Note that in the SCC scheme, for read cache misses that are clean everywhere, there are typically two network transactions.</p> <p>...</p> <p>The data comes back to tile 1 from the memory controller (or L4 cache) (action B4), and the directory controller then updates the directory storage 604 with tile 2's address (action B5). The directory then sends the data value to tile 2 cache 602 (action B6), and the cache 602 sends the value to the processor 600 (action B7) and to an optional L1 cache.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 1	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 29:5-28:</p> <p>FIG. 9C illustrates an example in which a ProxyIO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile's cache. As part of the provide I/O coherence protocol, the I/O module 912 includes a second mode of operation, as follows. In this mode, the I/O module 912 is configured to deliver write transactions directly to home tiles in a coherent fashion, over the on-chip interconnect (e.g., a dynamic network). This mode of operation is in addition to the first mode described above in which the I/O module 912 writes main memory and software maintains coherence with explicit invalidation of the location in a tile's cache. In the second mode, the I/O module 912 includes circuitry 904 implementing a hash function and address mapping table 906 to deliver write transactions to the home tile (Tile 2 in this example). Instead of a hash function and address mapping table, the I/O module 912 can use other techniques that use, for example, one or more bits in the address to find the home location. The home tile accepts the write transactions from the I/O module 912 just as it would write transactions from a tile. The home tile performs the normal coherence processing, with some differences as noted below, and sends a write acknowledgement message back to the I/O module 912 when the write transaction is complete.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

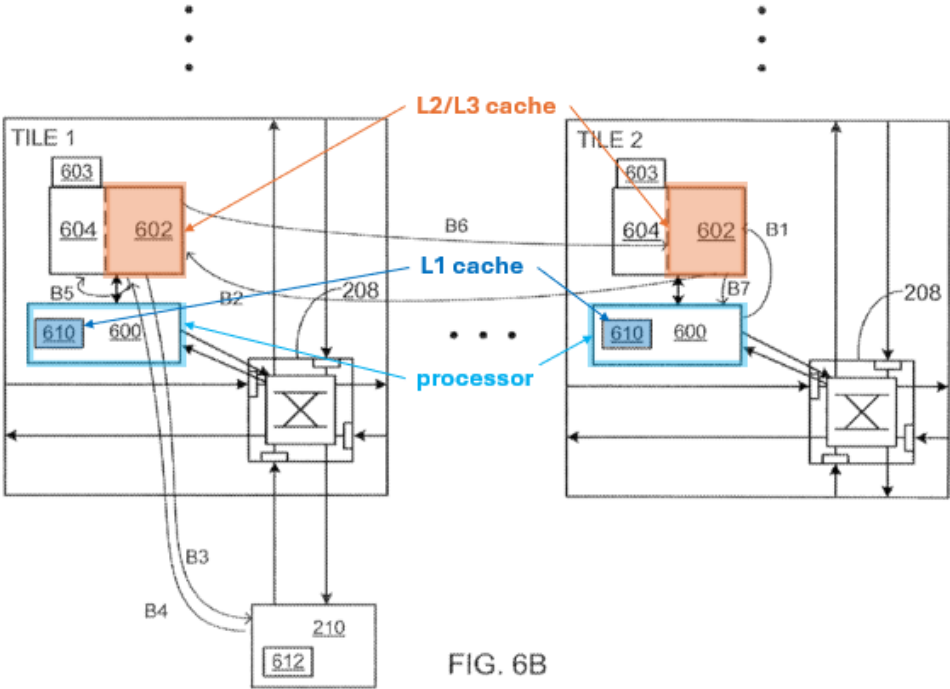
Claim 2	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[2] The apparatus of claim 1, wherein the caching agent includes a table of requests to store pending requests for the plurality of cores and the IIO module.</p>	<p><i>Agarwal</i> discloses or suggests “a table of requests to store pending requests for the plurality of cores and the IIO module.”</p> <p>The specification of the '228 Patent defines a table of requests (TOR) as “a buffer or other temporary storage for holding incoming requests.” '228 Patent at 2:46-48. <i>Agarwal</i> discloses this feature. For example, <i>Agarwal</i> discloses a transaction buffer that “contains a list of outstanding transactions with an acknowledgement counter associate with each.” <i>See Agarwal</i> at 27:21-25; <i>see also id.</i> at 16:1-5, 20:53-54. <i>Agarwal</i> further explains that the directory controller (part of the caching agent) maintains the count of outstanding acknowledgements. <i>See id.</i> at 5:32-34. And the directory storage (also a part of the caching agent) can store the list of outstanding transactions and associated counters. <i>See id.</i> at 27:11-25; <i>see also id.</i> at 16:39-42.</p> <p><i>Agarwal</i> discloses the following at 5:32-34:</p> <p style="padding-left: 40px;">The directory controller is further configured to maintain a count of outstanding acknowledgements to be received from processor cores in response to the invalidate request.</p> <p><i>Agarwal</i> discloses the following at 16:1-5:</p> <p style="padding-left: 40px;">A processor can also maintain a small number of transaction buffers associated with the directory controller at the shim. The number of such buffers can be 1, 2, 8, 16 or 64, for example. Each buffer entry can correspond to ongoing transactions associated with a cache line.</p> <p><i>Agarwal</i> discloses the following at 20:53-54:</p> <p style="padding-left: 40px;">The PCC protocol can also use all the mechanisms described in the SCC protocol.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 2	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 27:11-25:</p> <p>The coherence scheme can be modified further by using the cache state that is used to store directory state for multiple purposes. For example, during an eviction or other situation in which invalidates are happening, the tile keeps track of the number of acknowledgements that have been returned from other tiles. This can be stored as a counter in the same set of bits that are used to store directory information. Each time an acknowledgement comes back, the counter is incremented (or decremented).</p> <p>The tile can also keep a small buffer that contains a list of outstanding transactions along with an acknowledgement counter associated with each. This information can also be stored in one of the memory storage locations normally used for cache and memory access, such as a memory retry FIFO.</p>

Claim 4	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[4] The apparatus of claim 1, wherein the shared cache memory comprises a last level cache (LLC) including a plurality of distributed portions each associated with a corresponding one of the plurality of cores.</p>	<p><i>Agarwal</i> discloses or suggests “a last level cache (LLC) including a plurality of distributed portions each associated with a corresponding one of the plurality of cores.”</p> <p>As discussed above in [1b], <i>Agarwal</i> discloses an L2/L3 cache 602 in each of the tiles in the multicore processor. <i>See Agarwal</i> at 20:1-16, Fig. 6B. The L2/L3 cache 602 is coupled (i.e., associated) to processor 600 to provide L2 cache to back an L1 cache in the processor 600. <i>See id.</i> at 16:35-45, Fig. 6A-6B. <i>Agarwal</i> further discloses that L2/L3 cache 602 can be configured to “serve as both a local L2 cache for the local tile and as a shared L3 cache for other tiles.” <i>Id.</i> at 20:12-16; <i>see also id.</i> at 23:45-51.</p>

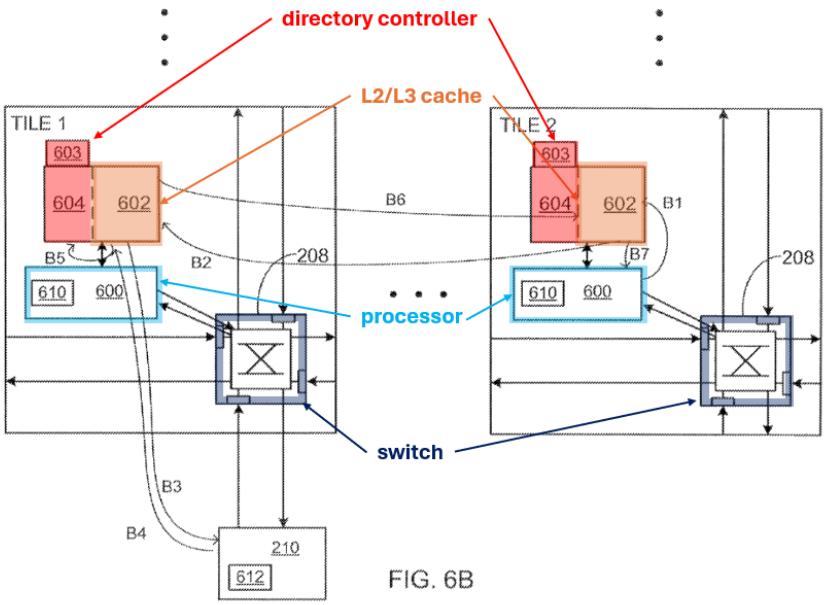
Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 4	Exemplary Disclosures Relevant to <i>Agarwal</i>
	<p><i>Agarwal</i> discloses the following at Figure 6B:</p>  <p><i>Agarwal</i> discloses the following at 16:35-45:</p> <p>Referring to FIG. 6A, each of the tiles (e.g., tile 1 and tile 2) that are used as home tiles for a given portion of main memory includes a processor 600 coupled to a cache 602 (e.g., providing an L2 cache to back an L1 cache that is integrated into the processor 600). A directory storage 604 for directory state can be included within the same memory that provides the cache 602, or can be included in a separate memory module on the tile. In some implementations, directory entries can be stored along side cache tags for a given memory line. A directory controller 603 is coupled to the directory storage 604.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 4	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 20:1-16:</p> <p>The case illustrated in FIG. 6B is similar to the case illustrated in FIG. 6A, but shows additional levels in the cache hierarchy. The processor 600 in each of the tiles includes an L1 cache 610 that is accessed first. After a miss in the L1 cache 610, action B1 on tile 2 attempts to access the cache 602 on tile 2 as an L2 access. The access of action B2 to the home tile's cache 602 in tile 1 is the L3 access for tile 2 after a miss from the L2 access of action B1. The multicore processor can also include an optional L4 cache 612 in the shim 210 as shown, where the action B3 in this case checks the L4 cache 612 before going to main memory. Thus, the cache 602 can serve as both a local L2 cache for the local tile and as a shared L3 cache for other tiles, as described in more detail below, such that for an access to location a from Tile 1, the cache 60 on tile 1 will serve as the L2 cache for that access.</p> <p><i>Agarwal</i> discloses the following at 23:45-53:</p> <p>The cache 206 coupled to the processor 204 in the tile can be used as a combined shared/private cache. It can be configured to cache data in two ways: (a) as part of a shared distributed cache (equivalent to a shared global/distributed L3 cache), and (b) as a private cache for data requested by this tile but homed in a different tile (equivalent to a private local L2 cache). For the shared distributed cache, each memory line has an assigned home location in the cache 206 of one of the tiles.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 5	Exemplary Disclosures Relevant to <i>Agarwal</i>
<p>[5] The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.</p>	<p><i>Agarwal</i> discloses or suggests “the plurality of distributed portions of the caching agent are coupled via a ring interconnect.”</p> <p>For example, <i>Agarwal</i> discloses that a mesh network connects the tiles, including the processor cores, L2/L3 caches, and directory controllers, through the switches 208. <i>See Agarwal</i> at 11:53-64; Fig. 2. The processor requests are routed via this mesh network to the directory controller in the home tile. <i>See id.</i> at 11:66-12:2; <i>see also id.</i> at 19:7-29. <i>Agarwal</i> further discloses that the tiles may be connected via a ring interconnect, among other types of interconnects. <i>See id.</i> at 3:45-47, 7:16-18, 11:66-12:2. <i>Agarwal</i> specifically explains that cache coherency can be maintained using a ring interconnect in the multicore processor. <i>See id.</i> at 20:63-67.</p> <p><i>Agarwal</i> discloses the following at Figure 6B:</p>  <p style="text-align: center;">FIG. 6B</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 5	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 3:45-47:</p> <p style="padding-left: 40px;">The processor cores are interconnected over a network that includes at least one of a bus, a ring, a mesh, a torus, a 3D cube, or a hypercube.</p> <p><i>Agarwal</i> discloses the following at 7:16-18:</p> <p style="padding-left: 40px;">The processor cores are interconnected over a network that includes at least one of: a bus, a ring, a mesh, a torus, a 3D cube, or a hypercube.</p> <p><i>Agarwal</i> discloses the following at 11:66-12:2:</p> <p style="padding-left: 40px;">A memory dynamic network can be dedicated to routing messages associated with memory access. The memory coherence techniques can be applied to architectures with any type of network including a bus, a ring, a mesh, a torus, a 3D cube, a hypercube, or any other direct or indirect network.</p> <p><i>Agarwal</i> discloses the following at 20:63-67</p> <p style="padding-left: 40px;">As before, this protocol maintains cache coherence in a multicore (also referred to as a chip multiprocessor or CMP) processor, wherein the multiple processors each having private caches are connected together in a mesh, ring, bus or other network (direct or indirect) network. The protocol is architected to be both logically simple and easy to verify as well as inexpensive to support in hardware.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 6	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[6] The apparatus of claim 1, wherein the caching agent is to receive an allocation transaction from the IIO module, the allocation transaction received in the IIO module from an IO device coupled to the multicore processor.</p>	<p><i>Agarwal</i> discloses or suggests “the caching agent is to receive an allocation transaction from the IIO module, the allocation transaction received in the IIO module from an IO device coupled to the multicore processor.”</p> <p>According to the '228 Patent, an allocation transaction can be generated by an I/O device to “lodge data directly into the [last level cache (LLC)].” '228 Patent at 3:22-25; <i>see also id.</i> at 3:47-51 (“this allocation transaction can be a request to write data directly into a cache memory of the processor, and it can be received from an IO device”).</p> <p><i>Agarwal</i> discloses this feature. For example, <i>Agarwal</i> describes two alternatives of an I/O module writing into memory (the “memory coherence case”) or writing into the cache (the “cache coherence case”). <i>See Agarwal</i> at 28:11-32. Figure 9C shows the “cache coherence case.” <i>See id.</i> at 29:5-28.</p> <p><i>Agarwal</i> further discloses that an I/O device sends a request (i.e., “allocation transaction”) to the shared cache of a home tile through the I/O module (i.e., “the caching agent is to receive an allocation transaction from the IIO module” and “the allocation transaction received in the IIO module from an IO device”). <i>Agarwal</i> at 29:5-30:11. The directory controller, in turn, allocates a shared cache line to the targeted cache. <i>Id.</i></p> <p><i>Agarwal</i> discloses the following at 27:44-54:</p> <p style="padding-left: 40px;">As an extension to the proxy cache coherence approach, a technique for ensuring I/O coherence is called “Proxy IO coherence.” As a review of proxy cache coherence, FIG. 9A shows a multicore processor 900 in which a processor 902 in Tile 1 is writing a shared cache location having address A (e.g., a cache line, or a data word within a cache line, or a region extending over multiple cache lines) using circuitry 904 implementing a hash function and address mapping table (AMT) 906 to deliver a write transaction to the home location in the cache 906 of Tile 2 using an on-chip interconnect (e.g., a dynamic network).</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 6	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 28:23-32:</p> <p>An I/O module is also able to process a DMA transaction or otherwise store data directly into the cache of a given tile, for example, into some other memory location B. In this case, a coherence protocol handles the situation that the main memory location B will have different data from the cache location B written by the I/O module. Further, if one or more other tiles have the same data location (B) cached in their cache, there is another potential coherence situation to be dealt with. The situation that occurs when the I/O module 912 writes directly into a tile is case 2: the “cache coherence case.</p> <p><i>Agarwal</i> discloses the following at 29:5-16:</p> <p>FIG.9C illustrates an example in which a Proxy IO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile’s cache. As part of the provide I/O coherence protocol, the I/O module 912 includes a second mode of operation, as follows. In this mode, the I/O module 912 is configured to deliver write transactions directly to home tiles in a coherent fashion, over the on-chip interconnect (e.g., a dynamic network). This mode of operation is in addition to the first mode described above in which the I/O module 912 writes main memory and software maintains coherence with explicit invalidation of the location in a tile’s cache.</p> <p><i>Agarwal</i> discloses the following at 29:46-62:</p> <p>For processing read transactions, the I/O module 912 sends read requests to the home tile, again, over the interconnection network (e.g., over a first dynamic network). The home tile will check its cache, and if the requested cache line exists, it will deliver the requested cache line back to the I/O module 912 over the interconnection network (e.g., over a second dynamic network different from the first dynamic network). If the requested line is not present in the home tile’s cache, the home tile will get the cache line from main memory and deliver it to the I/O module 912, with</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 6	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>or without allocating the line into the cache, depending on the protocol being used. In some cases, the tile does not allocate the line in the cache to avoid polluting that tile's cache since, if the line was not present in the cache, it is likely that the line is not needed by the tile. If the cache line exists in the home tile's cache, then the home tile delivers it to the I/O module 912, and cache line continues to be stored in the home tile's cache.</p>

Claim 7	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[7] The apparatus of claim 6, wherein the caching agent is to directly store data of the allocation transaction into the shared cache memory and to not initiate a memory transaction to a system memory coupled to the multicore processor.</p>	<p><i>Agarwal</i> discloses or suggests “the caching agent is to directly store data of the allocation transaction into the shared cache memory and to not initiate a memory transaction to a system memory coupled to the multicore processor.”</p> <p>As discussed above in [1c], <i>Agarwal</i> describes that the multicore processor supports I/O write transactions directly (i) into the main memory (<i>Agarwal</i> at 28:11-22, 28:33-43, Fig. 9B) or (ii) into the shared L2/L3 cache (<i>id.</i> at 27:39-28:8, 28:23-32, 29:5-30:11, Fig. 9C). In the first flow (“memory coherence case” depicted in Fig. 9B), the I/O write data is written to the memory first, and the home tile (Tile 2) performs coherence operation to make sure the copies in the tile caches are invalidated or updated. <i>See id.</i> at 28:11-22. In the second flow (“cache coherence case” depicted in Fig. 9C), the IO write data is written directly into the shared L2/L3 cache in the home tile (Tile 2), and the home tile performs the same coherence operation to make sure the copies in the tile caches are invalidated or updated. <i>See id.</i> at 23:26-41; 28:23-32. Compare the annotated Fig. 9B and 9C below:</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 7

Exemplary Disclosures Relevant to *Agarwal*

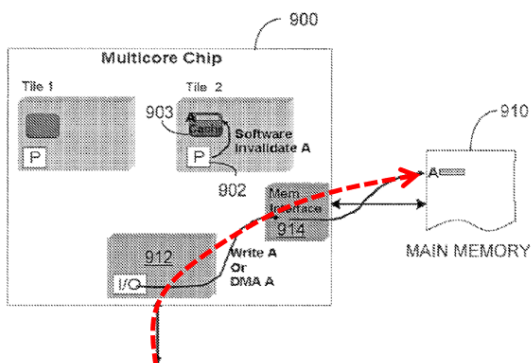


FIG. 9B

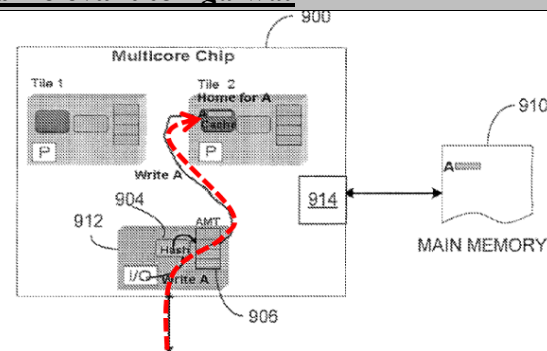


FIG. 9C

Thus, *Agarwal* discloses a DCA (direct cache access) flow in Fig. 9C which stores I/O data directly into the shared cache while not initiating a main memory update transaction.

Agarwal discloses the following at 23:26-41:

The second level cache (or optionally the first level cache) is augmented with an optional coalescing write buffer (CWB). The coalescing write buffer has a 64 B (64 byte) data field, a physical address CAM, and a 64 B dirty mask. (The specific data sizes are used for clarity only. The sizes can be modified.) For example, the CWB can store the addresses written into, and the values written into them. When coalescing subsequent writes to the same addresses, the tile overwrite [sic] the previous values in the CWB.

Keeping the stored values in the CWB without having them go to the memory system (e.g., the cache and/or the main memory) is useful because a write issued into the memory system causes the coherence protocol to potentially invalidate all the other copies. By storing the values locally in a CWB, the memory invalidate messages can be postponed to a later time.

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 7	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 28:10-43:</p> <p>An I/O module 912 includes an I/O interface 913 such as a network port (e.g., a XAUI interface or a PCI-e interface) and circuitry 915 to enable the module 912 to process direct memory access (DMA) transactions initiated from a tile, for example, or to otherwise handle a transaction to write data into off-chip main memory via a memory interface 914 (for example, into main memory location A). In this situation, if Some tile has the same data location (A) cached in its cache, a coherence protocol ensures that these caches are managed appropriately (e.g., invalidated or updated). This situation is shown in FIG. 9B. This situation is case 1: the “memory coherence case.</p> <p>An I/O module is also able to process a DMA transaction or otherwise store data directly into the cache of a given tile, for example, into some other memory location B. In this case, a coherence protocol handles the situation that the main memory location B will have different data from the cache location B written by the I/O module. Further, if one or more other tiles have the same data location (B) cached in their cache, there is another potential coherence situation to be dealt with. The situation that occurs when the I/O module 912 writes directly into a tile is case 2: the “cache coherence case.”</p> <p>Various approaches can be used to maintain coherence for case 1 and/or case 2. One approach to maintaining coherence for case 1 relies on software. To do so, as illustrated in FIG. 9B, the I/O module 912 writes the memory location (for example, A) to main memory 910 via the memory interface 914. Now, suppose Tile 2 needs to consume the data that has just been written to main memory 910 by the I/O module 912. Further, assume that Tile 2 potentially has the location A in its cache as shown. To ensure that Tile 2 does not read stale data, the location A is invalidated in association with the write transaction from the I/O module 912.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 7	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 29:5-16:</p> <p>FIG. 9C illustrates an example in which a ProxyIO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile's cache. As part of the provide I/O coherence protocol, the I/O module 912 includes a second mode of operation, as follows. In this mode, the I/O module 912 is configured to deliver write transactions directly to home tiles in a coherent fashion, over the on-chip interconnect (e.g., a dynamic network). This mode of operation is in addition to the first mode described above in which the I/O module 912 writes main memory and software maintains coherence with explicit invalidation of the location in a tile's cache.</p>

Claim 11	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
[11pre] A method comprising:	<p><i>Agarwal</i> discloses or suggests “a method.”</p> <p><i>Agarwal</i> discloses, among other things, a method of maintaining cache coherency for a plurality of distributed cache regions that share the same memory space. <i>See, e.g., Agarwal</i> at 4:55-64, 7:22-31.</p> <p><i>Agarwal</i> discloses the following at 4:55-64:</p> <p>The processor cores associated with the cache memory is configured to provide data requested in the memory request from the cache memory if the data is present in the cache memory or from the main memory if the data is not present in the cache memory. The directory controller coupled to the cache memory is configured to implement a directory coherence protocol for the corresponding portion of the main memory.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 11	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 7:22-31:</p> <p>In another aspect, in general, a method is described for managing cache memories associated with respective processor cores in a multicore processor that includes one or more memory interfaces providing memory access paths from the cache memories to a main memory that is external to the multicore processor. The method comprises operating one or more directory controllers for respective portions of the main memory from the processor cores or the memory interfaces; and storing directory state associated with the directory controllers within the multicore processor.</p>
<p>[11a] receiving an allocation transaction in an integrated input/output (IIO) module of a multicore processor from an IO device coupled to the multicore processor, the allocation transaction comprising a request to directly store data of the allocation transaction into a shared cache memory of the multicore processor;</p>	<p><i>Agarwal</i> discloses or suggests “receiving an allocation transaction in an integrated input/output (IIO) module of a multicore processor from an IO device coupled to the multicore processor, the allocation transaction comprising a request to directly store data of the allocation transaction into a shared cache memory of the multicore processor.”</p> <p><i>See supra</i> [6] (“receiv[ing] an allocation transaction from the IIO module, the allocation transaction received in the IIO module from an IO device coupled to the multicore processor); <i>see also supra</i> [7] (“directly store data of the allocation transaction into the shared cache memory”).</p> <p><i>Agarwal</i> discloses the following at 27:39-28:8:</p> <p>Some modules on a multicore processor other than the processor cores may have access to memory. An example of such a module is an I/O module. Maintaining memory coherence in the presence of such modules can be part of an overall coherence protocol. As an extension to the proxy cache coherence approach, a technique for ensuring I/O coherence is called “Proxy IO coherence.” As a review of proxy cache coherence, FIG. 9A shows a multicore processor 900 in which a processor 902 in Tile 1 is writing a shared cache location having address A (e.g., a cache line, or a data word within a cache line, or a region extending over multiple cache lines) using circuitry 904 implementing a hash function and address mapping</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 11	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>table (AMT) 906 to deliver a write transaction to the home location in the cache 906 of Tile 2 using an on-chip interconnect (e.g., a dynamic network). The home tile Tile 2 accepts the write transaction from Tile 1. The home tile performs the normal coherence processing, and sends a write acknowledgement message back to Tile 1 when the write transaction is complete. Normal coherence processing at Tile 2 (home tile) can include the following actions as needed: (1) updating the location A in Tile 2's cache 906; (2) if A was not in Tile 2's cache, then fetching the cache line (or at least the background data from the cache line) from main memory 910, storing the background data from that line in its cache 906, and optionally recording Tile 1 as a sharer in the directory entry on Tile 2 associated with location A if the cache 906 of Tile 1 is updated with the written location A (including any background data of the cache line). A similar set of actions happen at Tile 2 if Tile 1 makes a read request of data at location A. Tile 1 is recorded as a sharer in Tile 2's directory for a read request from Tile 1 since Tile 1 stores the latest value of location A in its cache on a read request, and option ally recorded as a sharer (depending on the specific protocol being used—e.g., write-allocate or no-write-allocate) for write requests from Tile 1, since Tile 1 does not necessarily need to store the latest value of location A in its cache.</p> <p><i>Agarwal</i> discloses the following at 28:23-32:</p> <p>An I/O module is also able to process a DMA transaction or otherwise store data directly into the cache of a given tile, for example, into some other memory location B. In this case, a coherence protocol handles the situation that the main memory location B will have different data from the cache location B written by the I/O module. Further, if one or more other tiles have the same data location (B) cached in their cache, there is another potential coherence situation to be dealt with. The situation that occurs when the I/O module 912 writes directly into a tile is case 2: the “cache coherence case.”</p> <p><i>See also Agarwal</i> at 29:5-30:11; Fig. 9C.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 11	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[11b] forwarding the allocation transaction to a caching agent of the multicore processor, the caching agent shared by a plurality of cores of the multicore processor and the IIO module; and</p>	<p><i>Agarwal</i> discloses or suggests “forwarding the allocation transaction to a caching agent of the multicore processor, the caching agent shared by a plurality of cores of the multicore processor and the IIO module.”</p> <p>As discussed above in [6], <i>Agarwal</i>’s multicore processor supports IO write transactions that store data directly into the L2/L3 cache memory in the home tile, allowing an I/O device coupled to the processor to send an allocation transaction (e.g., IO write transaction) to the directory controller in the home tile (part of the caching agent) that writes data directly into the shared L2/L3 cache.</p> <p>Furthermore, as discussed for [1d] and [1e] above, <i>Agarwal</i> discloses or suggests a single caching agent with distributed portions (e.g., directory controller and directory storage in each tile) that maintain cache coherency for the plurality of cores as well as the IIO module.</p>
<p>[11c] directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.</p>	<p><i>Agarwal</i> discloses or suggests “directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.”</p> <p>As discussed above in [7], <i>Agarwal</i> discloses a DCA (direct cache access) flow in Fig. 9C which stores IO data directly into the shared cache while not initiating a main memory update transaction. <i>See also Agarwal</i> at 28:23-25, 29:5-7, Fig. 9C.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 12	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[12] The method of claim 11, further comprising accessing the data in the shared cache memory via at least one of the plurality of cores of the multicore processor a number of times, without sending a transaction to a system memory coupled to the multicore processor.</p>	<p><i>Agarwal</i> discloses or suggests “accessing the data in the shared cache memory via at least one of the plurality of cores of the multicore processor a number of times, without sending a transaction to a system memory coupled to the multicore processor.”</p> <p>As discussed above for claims 7 and 11, <i>Agarwal</i> discloses a DCA (direct cache access) flow in Fig. 9C, which stores I/O data directly into the shared cache while not initiating a main memory update transaction. <i>See Agarwal</i> at 23:26-41, 28:23-32, 29:5-30:11, Fig. 9C. Further, <i>Agarwal</i> explains that, for the cache line displacement, the shared cache line “is either written back to memory (if dirty) or dropped (if clean).” <i>Id.</i> at 21:55-56; <i>see also id.</i> at 19:47-52, 20:39-42, 24:61-64. This means the shared cache uses a write-back policy. <i>See id.</i> at 24:61-64. In other words, when a processor core or I/O device writes a new data to a memory location, the new data is installed into the cache as a “modified” or “dirty” state (which indicates that the cache has the latest data while the memory has a stale data) without updating the main memory immediately. The modified data is only written back to the main memory later when the line is evicted/replaced, to create room for another memory location. This is why it is called a “write-back” policy. A write-back policy differs from a write-through cache policy, which forces the write data always to be sent out to the higher-level memory (e.g., system memory) immediately.</p> <p>Because <i>Agarwal</i>’s L2/L3 cache uses the write-back policy, data written by the I/O device through the DCA operation is installed/allocated into the L2/L3 cache is a “modified” state. In other words, if a core searches for data and hits in the private L1 or L2 cache, the data is returned, and if it misses in the private L1 and L2 caches, the core searches in the shared L3 cache of its home tile, and if hits there, the data is returned—all without going out to the main memory. <i>See id.</i> at 25:27-45. Thus, subsequent core accesses the same data hit in the L2/L3 cache, and therefore those core accesses do not go out to the main memory.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 12	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 19:47-52:</p> <p style="padding-left: 40px;">In some cases, the cache of tile 1 contains a copy of that cache line. If the directory states that the line is not dirty in someone's cache, then the directory will add a pointer (or a bit, or some other form of state information) in the directory entry indicating that a new tile (tile 2) now has a copy as well.</p> <p><i>Agarwal</i> discloses the following at 20:39-42:</p> <p style="padding-left: 40px;">That one pointer also tracks the location of the exclusively owned dirty copy of the data. (It might also allow both shared dirty copies and shared readers in update protocols).</p> <p><i>Agarwal</i> discloses the following at 21:55-56:</p> <p style="padding-left: 40px;">On the displacement of an L2 block, the block is either written back to memory (if dirty) or dropped (if clean).</p> <p><i>Agarwal</i> discloses the following at 23:26-41:</p> <p style="padding-left: 40px;">The second level cache (or optionally the first level cache) is augmented with an optional coalescing write buffer (CWB). The coalescing write buffer has a 64 B (64 byte) data field, a physical address CAM, and a 64 B dirty mask. (The specific data sizes are used for clarity only. The sizes can be modified.) For example, the CWB can store the addresses written into, and the values written into them. When coalescing subsequent writes to the same addresses, the tile overwrite [sic] the previous values in the CWB.</p> <p style="padding-left: 40px;">Keeping the stored values in the CWB without having them go to the memory system (e.g., the cache and/or the main memory) is useful because a write issued into the memory system causes the coherence protocol to potentially invalidate all the other copies. By storing the values locally in a CWB, the memory invalidate messages can be postponed to a later time.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 12	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>Agarwal discloses the following at 24:61-64:</p> <p style="padding-left: 40px;">The cache 206 uses a write-back policy for writing copies of cache lines stored as shared L3 cache lines, and uses a write-allocate write miss policy for these.</p> <p><i>Agarwal</i> discloses the following at 25:27-45:</p> <p style="padding-left: 40px;">On a read, a tile first checks its private level 1 and level 2 caches. The index function used is the PLIF. If hit, data is returned. If miss, based on PA<11:6>, the tile sends a read request over the TDN to the home tile. The home tile is indicated by PA<11:6>. In some cases, the home tile might be the same tile, in which case the cache is reaccessed with the shared home index function SHIF. The home tile checks its cache for the requested block, and assuming hit, returns the data to the requesting tile. The home tile adds the requesting tile to the sharing list for the block.</p> <p style="padding-left: 40px;">4.2 Simple Write Flow</p> <p style="padding-left: 40px;">When a write is pushed out of a tile's coalescing write buffer, the masked write (i.e., which specific byte or word is to be updated) is sent to the home tile's cache. The home tile merges the write with the background data (i.e., the data previously contained in that cache line), invalidates any sharers, and collects invalidate acknowledgements.</p> <p><i>Agarwal</i> discloses the following at 28:23-25:</p> <p style="padding-left: 40px;">An I/O module is also able to process a DMA transaction or otherwise store data directly into the cache of a given tile, for example, into some other memory location B.</p> <p><i>Agarwal</i> discloses the following at 29:5-7:</p> <p style="padding-left: 40px;">FIG.9C illustrates an example in which a Proxy IO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile's cache.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 12	<p><u>Exemplary Disclosures Relevant to Agarwal</u></p>
	<p>Agarwal discloses the following in Figure 9C:</p>
	<p>Figure 9C is a block diagram of a Multicore Chip (900). The chip contains two tiles: Tile 1 and Tile 2. Tile 2 is labeled "Home for A" and contains a processor (P) (906) and a cache (912). A "Write A" arrow points from the cache (912) to an I/O module (910). The I/O module (910) is connected to a Hash block (908) and an AMT block (906). The I/O module (910) is also connected to MAIN MEMORY (910) via a bidirectional arrow. The I/O module (910) is labeled "I/O module" with a green arrow. The cache (912) is labeled "L2/L3 cache" with an orange arrow. The I/O module (910) is labeled "I/O module" with a green arrow. The MAIN MEMORY (910) is labeled "MAIN MEMORY".</p>
	<p>FIG. 9C</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

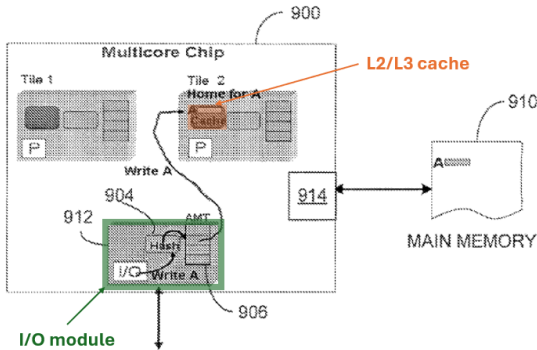
Claim 13	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[13] The method of claim 12, further comprising sending data of the cache line to the system memory from the shared cache memory if the cache line is evicted from the shared cache memory, when the cache line includes modified data.</p>	<p><i>Agarwal</i> discloses or suggests “sending data of the cache line to the system memory from the shared cache memory if the cache line is evicted from the shared cache memory, when the cache line includes modified data.”</p> <p>For example, as discussed above for claim 12, <i>Agarwal</i>’s L2/L3 cache uses the write-back policy and marks the modified lines as “modified” state (also known as “dirty” state). <i>See Agarwal</i> at 19:47-52, 20:39-42, 21:55-56, 24:61-64. <i>Agarwal</i> further teaches that, if a dirty cache line is evicted, the data of the cache line is written back to system memory and all copies of other tile caches sharing the line are invalidated. <i>See id.</i> at 26:23-30; <i>see also id.</i> at 21:55-56.</p> <p><i>Agarwal</i> discloses the following at 19:47-52:</p> <p style="padding-left: 40px;">In some cases, the cache of tile 1 contains a copy of that cache line. If the directory states that the line is not dirty in someone’s cache, then the directory will add a pointer (or a bit, or some other form of state information) in the directory entry indicating that a new tile (tile 2) now has a copy as well.</p> <p><i>Agarwal</i> discloses the following at 20:39-42:</p> <p style="padding-left: 40px;">That one pointer also tracks the location of the exclusively owned dirty copy of the data. (It might also allow both shared dirty copies and shared readers in update protocols).</p> <p><i>Agarwal</i> discloses the following at 21:55-56:</p> <p style="padding-left: 40px;">On the displacement of an L2 block, the block is either written back to memory (if dirty) or dropped (if clean).</p> <p><i>Agarwal</i> discloses the following at 24:61-64:</p> <p style="padding-left: 40px;">The cache 206 uses a write-back policy for writing copies of cache lines stored as shared L3 cache lines, and uses a write-allocate write miss policy for these.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 13	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p><i>Agarwal</i> discloses the following at 26:23-30:</p> <p>One possible performance issue with the previous scheme is the fact that evictions on home tiles induce evictions in tiles sharing the data. Because the directory information is not stored in memory, when a shared line is evicted from its home tile, all copies cached at tiles sharing the line are invalidated. If this is not done, it is possible that a tile might end up with stale data. This subsection describes one possible approach to address this issue. The next subsection describes another approach.</p>

Claim 14	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
<p>[14] The method of claim 12, further comprising directly storing the data of the allocation transaction into the shared cache memory without communicating with the system memory.</p>	<p><i>Agarwal</i> discloses or suggests “directly storing the data of the allocation transaction into the shared cache memory without communicating with the system memory.”</p> <p>As discussed above for claim 7, <i>Agarwal</i> teaches a “cache coherence case,” wherein I/O write data is written directly into the shared L2/L3 cache without initiating a main memory (i.e., “system memory”) update transaction. <i>See supra</i> at [7], [11], [12]; <i>see also Agarwal</i> at 23:26-41, 28:23-32, 29:5-30:11, Fig. 9C.</p> <p><i>Agarwal</i> discloses the following at 23:26-41:</p> <p>The second level cache (or optionally the first level cache) is augmented with an optional coalescing write buffer (CWB). The coalescing write buffer has a 64 B (64 byte) data field, a physical address CAM, and a 64 B dirty mask. (The specific data sizes are used for clarity only. The sizes can be modified.) For example, the CWB can store the addresses written into, and the values written into them. When coalescing subsequent writes to the same addresses, the tile overwrite [sic] the previous values in the CWB.</p>

Appendix 228-01: Invalidity of U.S. Patent No. US 8,984,228 (the '228 Patent) Based on *Agarwal*

Claim 14	<u>Exemplary Disclosures Relevant to <i>Agarwal</i></u>
	<p>Keeping the stored values in the CWB without having them go to the memory system (e.g., the cache and/or the main memory) is useful because a write issued into the memory system causes the coherence protocol to potentially invalidate all the other copies. By storing the values locally in a CWB, the memory invalidate messages can be postponed to a later time.</p> <p><i>Agarwal</i> discloses the following at 28:23-25:</p> <p>An I/O module is also able to process a DMA transaction or otherwise store data directly into the cache of a given tile, for example, into some other memory location B.</p> <p><i>Agarwal</i> discloses the following at 29:5-7:</p> <p>FIG.9C illustrates an example in which a Proxy IO coherence protocol is used for situations in which an I/O module 912 writes or reads data to or from a tile's cache.</p> <p><i>Agarwal</i> discloses the following in Figure 9C:</p>  <p style="text-align: center;">FIG. 9C</p>