

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ARM LTD., MEDIATEK INC,

Petitioners,

v.

DAEDALUS PRIME LLC,

Patent Owner.

Case No.: IPR2025-00207

U.S. Patent No. 8,984,228

PATENT OWNER'S RESPONSE

TABLE OF CONTENTS

I. INTRODUCTION AND SUMMARY OF ARGUMENTS.....	1
II. BACKGROUND: THE '228 PATENT, AND CLAIM CONSTRUCTION	4
A. The '228 Patent	4
B. Claim Construction and the Level of Ordinary Skill in the Art.....	5
III. THE PETITION FAILS TO DEMONSTRATE THAT ANY CHALLENGED CLAIM IS UNPATENTABLE	6
A. The Challenged Claims are Not Unpatentable in View of Agarwal	8
1. The “Single Caching Agent” Limitation of Claim 1 Is Not Disclosed or Rendered Obvious by Agarwal.....	9
2. Claim 2 Is Not Disclosed or Rendered Obvious by Agarwal	21
3. Claim 6 Is Not Obvious in View of Agarwal.....	27
4. Claim 11 and Its Dependent Claims Are Not Obvious in View of Agarwal.....	29
5. Claim 14 is Not Obvious in View of Agarwal.....	32
B. The Challenged Claims are Not Unpatentable in View of Sinharoy.....	35
1. Overview of Sinharoy	35
2. Claim 1 is Not Obvious in View of Sinharoy	35
3. Claim 4 is Not Obvious in View of Sinharoy	37
4. Claim 6 is Not Obvious in View of Sinharoy	40
5. Claim 11 is Not Obvious in View of Sinharoy	41
C. Claims 2, 3, and 16 Are Patentable Over Sinharoy and Blumrich	42

1. Claim 2 is Not Obvious in View of Sinharoy and Blumrich	42
2. Claim 3 is Not Obvious in View of Sinharoy and Blumrich	46
3. Claims 16-18 are Not Obvious in View of Sinharoy and Blumrich	49
D. Claim 5 is Patentable Over Sinharoy and Sistla and Claim 19 is Patentable Over Sinharoy, Blumrich, and Sistla	49
IV. CONCLUSION	52

TABLE OF AUTHORITIES

CASES

<i>CFMT, Inc. v. Yieldup Int’l Corp.</i> , 349 F.3d 1333 (Fed. Cir. 2003).....	20, 32, 42
<i>Harari v. Lee</i> , 656 F.3d 1331 (Fed. Cir. 2011).....	14, 22
<i>Hartness Int’l. Inc. v. Simplimatic Engineering Co.</i> , 819 F.2d 1100 (Fed. Cir. 1987).....	<i>passim</i>
<i>In re Gordon</i> , 733 F.2d 900 (Fed. Cir. 1984).....	7
<i>In re Peterson</i> , 315 F.3d 1325 (Fed. Cir. 2003).....	8
<i>In re Ratti</i> , 270 F.2d 810 (CCPA 1959)	7, 16, 40
<i>In re Warner</i> , 379 F.2d 1011 (CCPA 1967)	7
<i>Intelligent Bio-Sys., Inc. v. Illumina Cambridge Ltd.</i> , 821 F.3d 1359 (Fed. Cir. 2016).....	7
<i>KSR Int’l v. Teleflex Inc.</i> , 550 U.S. 398 (2007).....	7, 8, 33
<i>Phillips v. AWH Corp.</i> , 415 F.3d 1303 (Fed. Cir. 2005).....	5
<i>SAS Inst. Inc. v. Iancu</i> , 138 S. Ct. 1348 (2018).....	6
<i>Thorner v. Sony Comput. Ent. Am. LLC</i> , 669 F.3d 1362 (Fed. Cir. 2012).....	5

OTHER AUTHORITIES

<i>Changes to the Claim Construction Standard for Interpreting Claims in Trial Proceedings Before the Patent Trial and Appeal Board</i> , 83 Fed. Reg. 51,340 (Oct. 11, 2018)	4
---	---

EXHIBIT LIST

Exhibit No.	Description
2001	<i>SB IP Holdings, LLC v. Vivint, Inc.</i> , No. 4:21-cv-00912 (E.D. Tex. Apr. 6, 2023) Slip Opinion.
2002	<i>Portsmouth Network Corp. v. Cisco Systems, Inc.</i> , No. 2:23-CV-00441 (E.D. Tex. Oct. 21, 2024) Slip Opinion.
2003	<i>Daedalus Prime LLC v. MediaTek Inc.</i> , No. 2:24-cv-235-JRG (E.D. Tex.) MediaTek's Allegations of Invalidity of the '228 Patent Based on Agarwal.
2004	<i>Daedalus Prime LLC v. MediaTek Inc.</i> , No. 2:24-cv-235-JRG (E.D. Tex.) MediaTek's Allegations of Invalidity of the '228 Patent Based on Sinharoy.
2005	Arimilli, US PG PUB 2004/01339283 A1.
2006	Declaration of Nader Bagherzadeh, Ph.D.
2007	Transcript of Deposition of Dr. Hassoun

I. INTRODUCTION AND SUMMARY OF ARGUMENTS

ARM LTD.¹ challenges the patentability of Claims 1-7, 11-14, and 16-19 of U.S. Patent No. 8,984,228 (the “’228 patent”) but fails to demonstrate that these challenged claims are unpatentable.

As to the challenges based on Agarwal, Petitioner’s theory that Agarwal discloses or renders obvious the “single caching agent ... including a plurality of distributed portions each associated with a corresponding one of the plurality of cores” of Claim 1 fundamentally misconstrues both the claim language and Agarwal’s architecture. Agarwal describes a tiled multicore system with independent directory controllers, each responsible only for its assigned memory region. There is no logically unified “single caching agent” managing coherencies for all cores and the IIO module, as required by Claim 1. The alleged “association” between directory controllers and cores in Agarwal is merely architectural, not functional—each controller manages an assigned memory region, regardless of which core is seeking

¹ On Oct., 27, 2025, Petitioner MediaTek Inc. filed a Joint (with Patent Owner) Motion to Terminate this proceeding with respect to MediaTek. Paper 18. That motion has yet to be ruled on by the Board, however, for purposes of this Response ARM is treated as the sole remaining petitioner.

data. Agarwal's ProxyIO extension does not change this: each directory controller remains autonomous, and there is no single agent responsible for coherency across the system. Petitioner also fails to provide any reasoned motivation to modify Agarwal's distributed architecture to arrive at the claimed invention.

Regarding the challenges based on Sinharoy, Petitioner's reliance on the IBM POWER7 architecture described in Sinharoy does not meet the '228 patent's requirements. First, Sinharoy's coherence is enforced by broadcast snoops among multiple independent controllers (L2/L3/I/O/memory) acting as peers on a fabric; it does not disclose a "single caching agent" that performs coherency for both the plurality of cores and an integrated IIO module, as Claim 1 requires.

Second, while POWER7's L3 cache is partitioned into eight "local" regions, each with its own centralized controller, those "local" regions are address-homed and shared among cores; they are not one-to-one associated with corresponding cores. That design contradicts Claim 4's per-core association of distributed LLC portions.

Third, Sinharoy's "cache injection" allows an I/O controller, as a full peer, to write DMA data directly into an L3 region when that region owns

the line; there is no device-originated “allocation transaction” received in an I/O and forwarded to a single caching agent shared with the cores, as demanded by Claims 6 and 11.

Finally, the combinations Petitioner proposes do not cure these gaps. Blumrich’s per-slice L2 request queue is not a unified ingress “table of requests” (TOR) for mixed core/I/O traffic nor does it disclose a dedicated entry for posted transactions (Claims 2–3, 16). Sistla’s scalability agent and ring interconnect are coordination mechanisms, not distributed portions of a single caching agent coupled by a ring (Claims 5, 19).

In short, Sinharoy, alone or in combination with Blumrich or Blumrich and Sistla, fails to teach or suggest the ’228 patent’s single, logically unified caching agent with per-core-associated distributed portions, I/O-originated allocation transaction semantics and forwarding, and a unified TOR with posted-transaction capacity. The challenged claims therefore remain patentable over Sinharoy.

These and further reasons for the patentability of the challenged claims are discussed in detail below.

II. BACKGROUND: THE '228 PATENT, AND CLAIM CONSTRUCTION

A. The '228 Patent

The '228 patent addresses critical scalability and performance challenges that arise when input/output (“IO”) functionality is integrated onto the same die as a multicore processor. Traditional architectures relied on separate caching agents for IO components, which significantly increased snoop traffic and system complexity in multiprocessor environments. Ex. 1001, 1:7–30. The '228 patent introduces a unified solution: a single caching agent—distributed across portions associated with individual cores—that performs cache coherency operations for both processor cores and an integrated IO module. *Id.*, 2:17–19, 35-50; 2:57 – 3:3. This architecture enables direct allocation of IO data into shared cache memory, reduces memory bandwidth consumption, and improves latency, while incorporating advanced arbitration and deadlock avoidance mechanisms to ensure robust operation under heavy traffic. *Id.*, 3:4-32; 5:22–6:19. These innovations collectively deliver superior scalability, efficiency, and reliability compared to conventional designs. Ex. 2006, ¶¶ 29-31.

B. Claim Construction and the Level of Ordinary Skill in the Art

In *inter partes* reviews such as this one, the Board applies the same claim construction standard used to construe a claim in a civil action under 35 U.S.C. § 282(b). *Changes to the Claim Construction Standard for Interpreting Claims in Trial Proceedings Before the Patent Trial and Appeal Board*, 83 Fed. Reg. 51,340 (Oct. 11, 2018). Accordingly, the claims should be construed consistent with their plain and ordinary meaning as understood by a person of ordinary skill in the art when read in the context of the specification and prosecution history. *Thorner v. Sony Comput. Ent. Am. LLC*, 669 F.3d 1362, 1365 (Fed. Cir. 2012) (citing *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312–13 (Fed. Cir. 2005) (en banc)). “The ordinary and customary meaning of a claim term is the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention.” *Phillips* at 1313.

In its Institution Decision, the Board defined a person of ordinary skill in the art as someone with: “(1) at least a bachelor’s degree in electrical engineering, computer engineering, physics, or a similar field, (2) approximately two years of industry experience in a field related to microelectronic architecture design, including experience with cache

management in multicore systems-on-a-chip (SoCs)” Paper 15, 15. That definition is used for purposes of this Response.

Claim 1 recites a “single caching agent.” Ex. 1001, 7:43. Dr. Bagherzadeh explains that while “a single caching agent” means “one caching agent for the processor,” that is not the whole story. The single caching agent should be understood as a logically unified entity, that is, a cohesive caching agent where all distributed portions are interconnected, consistent, and function together in a rational or systematic manner, not a collection of independent caching agents that merely communicate with one another. Ex. 2006, ¶ 33 (citing Ex. 1001, 2:18-20, 58-60; 6:25-29; 7:5-7).

III. THE PETITION FAILS TO DEMONSTRATE THAT ANY CHALLENGED CLAIM IS UNPATENTABLE

A petition must be judged as it is written. The Board may not overlook flaws in the petition or reformulate its challenges to enhance their suitability. *See SAS Inst. Inc. v. Iancu*, 138 S. Ct. 1348, 1355 (2018) (“[I]t’s the petitioner, not the Director, who gets to define the contours of the proceeding.”); *see also id.* (“[T]he petitioner is master of its complaint . . .”); *see also id.* at 1353 (forbidding the Board from “curat[ing]” the petition).

Because all of the challenges in the petition rely on conclusions of obviousness, *see* Pet., 5, the petition must include “some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness.” *KSR Int’l v. Teleflex Inc.*, 550 U.S. 398, 418 (2007). Petitioner must supply the factual basis for the challenge and “may not resort to speculation, unfounded assumptions or hindsight reconstruction to supply deficiencies in its factual basis.” *In re Warner*, 379 F.2d 1011, 1017 (CCPA 1967). Additionally, because the obviousness challenges presented in the petition involve a modification or combination of teachings, Petitioner must also establish that one skilled in the art would have had a reasonable expectation of success to make the modification or combination. *Intelligent Bio-Sys., Inc. v. Illumina Cambridge Ltd.*, 821 F.3d 1359, 1367-68 (Fed. Cir. 2016). The modification or combination must not “require a substantial reconstruction and redesign of the elements” in a reference or “a change in the basic principles under which [a reference] was designed to operate.” *In re Ratti*, 270 F.2d 810, 813 (CCPA 1959). Nor can the modification or combination render a reference unsatisfactory or “inoperable for its intended purpose.” *In re Gordon*, 733 F.2d 900, 902 (Fed. Cir. 1984).

Also, the prior art must be considered for all that it teaches, and a *prima facie* case of obviousness may be rebutted “by showing that prior art teaches away from claimed invention in any material respect.” *In re Peterson*, 315 F.3d 1325, 1331 (Fed. Cir. 2003).

Last but not least, obviousness cannot be established by hindsight combination to produce the claimed invention, as the Supreme Court has emphasized:

A factfinder should be aware, of course, of the distortion caused by hindsight bias and must be cautious of arguments reliant upon *ex post* reasoning. *See Graham [v. John Deere Co.]*, 383 U.S. [1,] 36, 86 S. Ct. 684, 15 L. Ed. 2d 545 [(1966)] (warning against a “temptation to read into the prior art the teachings of the invention in issue” and instructing courts to “guard against slipping into use of hindsight”) (quoting *Monroe Auto Equip. Co. v. Heckethorn Mfg. & Supply Co.*, 332 F.2d 406, 412 (6th Cir. 1964)).

KSR, 550 U.S. at 421.

A. The Challenged Claims are Not Unpatentable in View of Agarwal

ARM contends that Claims 1–2, 4–7, and 11–14 of the ’228 patent are obvious in view of Agarwal, U.S. Patent No. 7,805,575 (Ex. 1005). Pet., 5.

ARM's theory is that Agarwal's "proxy cache coherence" (PCC) protocol and "ProxyIO" extension disclose or render obvious every element of these challenged claims, including the "single caching agent ... including a plurality of distributed portions each associated with a corresponding one of the plurality of cores." *Id.*, 10-24. This argument, however, fundamentally misconstrues both the claim language and the technical teachings of Agarwal.

1. The "Single Caching Agent" Limitation of Claim 1 Is Not Disclosed or Rendered Obvious by Agarwal

Agarwal describes a tiled multicore processor architecture in which a so-called "home tile" includes a directory controller responsible for managing cache coherence for a specific region of memory. The processor implements a distributed directory protocol, where each directory controller operates autonomously for its assigned memory region. Ex. 2006, ¶ 35 (citing Ex. 1005, 3:7-14, 21-24, 39-41, 55-61; 4:9-12, 40-45, 55-57; 17:42-57; 23:51-53). When a processor core or an IO module wishes to access a memory location, the request is routed to the home tile for that address and the directory controller on the home tile determines the appropriate coherence action. Each directory controller acts independently for its

assigned memory region. *Id.*, ¶ 36. With the “ProxyIO” coherence mechanism, the same protocol is extended to IO transactions. When an IO module initiates a transaction—such as a DMA write or read—that transaction is sent directly to the home tile’s directory controller, which processes the IO request in the same manner as a request from a core. *Id.*, ¶ 37.

a) ARM Conflates Agarwal’s “Distributed Directory Controllers” with the Claimed “Single Caching Agent”

Agarwal’s architecture is characterized by a set of distributed and independent directory controllers, each managing coherence for a distinct memory region. *Id.*, ¶ 38. In particular, each tile (core) has its own directory controller and directory storage, which operates independently for its assigned memory region. This is an example of a distributed directory protocol, where each directory controller is responsible for its own partition of memory, and there is no central or logically unified agent managing coherency for the entire multicore processor. *Id.*, ¶ 41; Ex. 1005, 16:26–45, 23:51–53.

In contrast, the “single caching agent” of Claim 1 is not a mere collection of independent cache controllers, but instead, a logically unified agent with distributed portions, each tied to a core, that collectively perform

coherency for both the cores and the IIO module. Ex. 2006, ¶ 39; Ex. 1001 at 2:66 – 3:3; 6:22–29. Indeed, the “single caching agent” of Claim 1 specifically avoids the need for multiple caching agents, as are found in Agarwal. Ex. 2006, ¶ 40. Among other advantages, by employing the “single caching agent,” a separate, dedicated IO caching agent is avoided, and the amount of snoop traffic is reduced. *Id.*; Ex. 1001, 3:4-12.

Dr. Hassoun argues that Agarwal’s distributed directory controllers “collectively” manage coherence. Ex. 2007, 83:5–84:19; *and see* Ex. 1003, ¶ 64. But what Dr. Hassoun calls “collective action” is merely message passing and it does not satisfy Claim 1’s requirement for a single caching agent. Ex. 2006, ¶¶ 42-44.

First, message passing among the directory controllers of different tiles does not mean the independent directory controllers are a logically unified entity. In Agarwal, each directory controller is autonomously responsible for its own assigned memory region. It, and it alone, makes coherence decisions for that memory region. *Id.*, ¶¶ 42-43; Ex. 1005, 16:26–45; 19:7 – 20:25 (explaining that in the event of a cache miss on a tile, the “miss goes as a message through the network to the home tile,” which handles cache coherence operations for the associated address); 23:51–53.

The distinction is one between cooperation among peers (Agarwal) and unity of purpose and control (the '228 patent). Ex. 2006, ¶ 43.

Collective action by independent agents such as Agarwal's directory controllers involves a group of autonomous entities with their own decision-making processes. Any interaction is simply in the nature of informing other controllers of those decisions. A single agent with distributed portions, in contrast, is a unified entity whose components operate under centralized control. Dr. Bagherzadeh explains it by way of analogy: In a company with "collective" management, several independent managers each run their own department while periodically informing the others about their decisions. In contrast, a company with a single manager with deputies in each department ("single agent with distributed portions") operates under a centralized control. The two are not the same, even if in the first example individual department managers coordinate through communication with one another. *Id.*, ¶¶ 43-44.

b) The Association of Distributed Portions with Cores Is Not Disclosed

Claim 1 requires each distributed portion of the single caching agent be "associated with a corresponding one of the plurality of cores." ARM argues that because each tile includes a directory controller, this constitutes

the “association” recited in the claim. Pet., 23-24. But this is not so. In fact, Agarwal fails to meet this requirement.

Agarwal’s directory controllers serve—are associated with—memory regions, not cores. Ex. 1005, 19:7-29. Memory regions are assigned according to a hash of the memory address or other mapping function. Any core or IO module that generates a request for a particular memory address, has that request routed to the directory controller for that address. Ex. 2006, ¶ 46. The directory controller’s responsibility in terms of cache coherency operations is to its assigned memory region, not to any one core. *Id.*; Ex. 1005, 14:60 – 15:12; 16:31-45; 17:50-51; 23:51-53. In contrast, Claim 1 requires each distributed portion of the caching agent be associated with ***one*** of the plurality of cores. Ex. 1001, 7:44-46. Dr. Bagherzadeh explains that “a POSITA would understand that the distributed portion is not merely co-located with a core, but as the component that performs cache coherency operations for that core.” Ex. 2006, ¶ 47.

This distinction can be illustrated as follows: In Agarwal, if core A on Tile 1 accesses a memory address whose home tile is Tile 5, the directory controller on Tile 5, not Tile 1, manages coherence for that address. The same is true if core B on Tile 2 accesses that same memory address. In no

meaningful way, in the context of Claim 1, is the directory controller on Tile 5 associated with *a core* (i.e., with only a core on Tile 5), as it must manage coherence for requests made by any and all cores of the multiprocessor whenever those cores access one of the directory controller’s assigned memory addresses. *Id.*, ¶ 48.

The Federal Circuit has made clear that “[w]hen the claim language and specification indicate that ‘a’ means one and only one, it is appropriate to construe it as such even in the context of an open-ended ‘comprising’ claim.” *Harari v. Lee*, 656 F.3d 1331, 1341 (Fed. Cir. 2011). Here, both the claim itself and the specification indicate a one-to-one relationship between a distributed portion of the single caching agent and a core. Ex. 2006, ¶ 47. Agarwal, on the other hand, describes a one-to-one relationship between a directory controller and a region of memory, not a core. In Agarwal, a given directory controller handles coherency operations for any/all core(s) with respect to its assigned memory addresses. Accordingly, Agarwal does not teach or suggest at least this “association” requirement of claim 1.

c) *The “Single Caching Agent” Must Perform Coherency for Both Cores and the I/O Module*

Agarwal’s “ProxyIO” extension allows an I/O module to send requests to the appropriate directory controller for an assigned memory

region. Ex. 1005, 27:44-46; 29:5-31; Ex. 2008, 66:18 – 67:10. But again, that appropriate directory controller is the one for the target memory address. Ex. 1005, 29:10-12. There is no single caching agent that manages coherency for both the cores and the I/O module; rather, the I/O module will use whichever directory controller manages the target memory region. Ex. 1005, 29:5-31; Ex. 2006, ¶¶ 49-51; Ex. 2008, 67:11-15. And each directory controller separately manages coherence operations for its respective memory region. There is no single agent responsible for coherency for all cores and the I/O module together. Ex. 2006, ¶ 51.

d) Petitioner Fails to Identify a Motivation to Combine or Modify Agarwal to Arrive at the Claimed Invention

The petition does not provide a reasoned motivation for a POSITA to modify Agarwal's distributed architecture to be a logically unified single caching agent with distributed portions, as recited in Claim 1. *Id.*, ¶¶ 52-57. Indeed, several considerations demonstrate why such a modification is neither practical nor possible:

(a) Physical distribution and scalability: Agarwal's system is designed for scalability and parallelism. Each tile is a self-contained unit with its own local resources, including cache and directory controller. Introducing a single, centralized caching agent would create a bottleneck as the number of

tiles increases, undermining the main advantage of the tiled approach.

Routing all cache coherence traffic routed through a single agent would introduce significant wiring complexity and latency, especially as chip sizes grow. *Id.*, ¶ 54.

(b) Distributed directory protocols are fundamental to Agarwal. The underlying principle is that each directory controller is responsible for coherence for a subset of memory (its “home region”). This localizes coherence operations, reduces global communication, and enables the system to scale. *Id.*, ¶ 55. A single caching agent, by contrast, necessarily requires all coherence operations—reads, writes, invalidations, etc.—to be under the control of one logical entity; something fundamentally at odds with the distributed, parallel nature of tiled systems. *Id.*; Ex. 1005, 16:26–45, 27:39–28:9. Such a departure would amount to a change in the principle of operation of Agarwal, further demonstrating the nonobviousness of Claim 1. *In re Ratti*, 270 F.2d at 813.

(c) Logical vs. physical unity: The “single caching agent” of Claim 1 is a logically unified agent, even if physically distributed. In Agarwal, independent directory controllers each act autonomously for assigned memory regions. To modify this arrangement to be a true “single caching

agent” would require unifying all directory controllers under common control, which would necessitate architectural changes not taught or suggested by Agarwal. Ex. 2006, ¶ 56. For example, an arbiter and/or scheduler would be needed to coordinate the different operations and actions arriving from different ones of the directory controllers. *See, e.g.*, Ex. 1001, 2:44-52.

(d) Performance and power: Routing all coherence traffic through a single agent would create a performance bottleneck and increase power consumption due to longer communication paths and higher contention. Ex. 2006, ¶ 57.

As none of the above considerations are present in or suggested by Agarwal, Claim 1 is not obvious in view of Agarwal.

e) Sharing of Cache Lines in Agarwal Does Not Require Logical Unification of Caching Agents

Petitioner may argue that sharing of cache lines in Agarwal must mean there is logical unity or a “single caching agent” present. Not so. Sharing cache lines in different caches does require communication between different, independent directory controllers. But that is not logical unity in the sense of a single caching agent as recited in claim 1. *Id.*, ¶ 58. When a cache line is shared, it is still the home tile’s directory controller that tracks

which tiles have copies (using pointers, sharer lists, or bits). If another tile wants to write to a shared line, it must communicate with the home tile's controller, which then issues invalidations or updates as needed. *Id.*, ¶ 59 (citing Ex. 1005 3:39-41; 4:9-12, 19-22; 14:33-36; 16:58-66; 22:27-29; 23:51-53). In other words, each directory controller still acts autonomously for its assigned memory region. *Id.*, ¶ 60. There is no one entity maintaining global state, no unified logic, and no single point of control. Agarwal itself describes the protocol as “distributed,” with each home tile acting as a “proxy” for its assigned region. There is no suggestion that the controllers are just portions of a single agent. *Id.*, ¶¶ 60-62.

f) *The Protocol-Driven Cooperation Among Directory Controllers in Agarwal Does Not Mean They are Parts of a Single Caching Agent*

In Agarwal, directory controllers cooperate by following a distributed coherence protocol, e.g., sending invalidation messages, acknowledgments, etc. Dr. Hassoun describes this as a “collective” approach to coherency operations that establishes Agarwal as teaching a single caching agent. Ex. 1003, ¶ 64. But this is wrong.

The message passing between directory controllers is necessary for correctness, but it is not centralized control; i.e., not a logical unity of

directory controllers. As discussed above, each directory controller operates autonomously for its assigned memory region. Message passing between independent agents does not transform them into a single, unified decision-making entity. Ex. 2006, ¶¶ 65-70 (describing Agarwal’s design philosophy of “distributed coordination” which includes communication among peer director controllers and contrasting it with the ’228 patent’s single caching agent, meaning one agent, even if physically distributed).

The ’228 patent repeatedly describes what Claim 1 recites as “a single caching agent,” configured to handle coherency for both cores and the I/O module—that is, under unified control. *See, e.g.*, Ex. 1001, 2:35-52, 58-60 (“Thus via caching agent 200, cores of the processor and an I/O module can maintain coherency without the need for additional caching agents.”). This means that there is one logical entity that knows the complete coherence status of every memory line across all participants. Decisions about cache coherence are made with awareness of the entire system’s state. Ex. 2006, ¶ 70.

Agarwal never suggests such unified control. Each directory controller handles coherence operations for its own memory region, but there is no single agent or mechanism that aggregates coherence information

across all controllers/tiles. Instead, each controller acts independently. System-wide consistency is achieved through protocol-driven messaging, providing a unified view of memory, but there is no single unified caching agent. *Id.*

For all of the above reasons, the petition fails to establish that Agarwal discloses or renders obvious the “single caching agent ... including a plurality of distributed portions each associated with a corresponding one of the plurality of cores” required by Claim 1 of the ’228 patent. *Id.*, ¶ 71. Agarwal’s distributed directory controllers are not a “single caching agent,” and there is no teaching, suggestion, or motivation to modify Agarwal to arrive at the claimed invention. Accordingly, Claim 1 is not obvious in view of Agarwal. *CFMT, Inc. v. Yieldup Int’l Corp.*, 349 F.3d 1333, 1342 (Fed. Cir. 2003) (obviousness requires a suggestion of all limitations in a claim).

Claims 2 and 4–7 depend from Claim 1 and are, therefore, likewise not obvious in view of Agarwal. *Hartness Int’l. Inc. v. Simplimatic Engineering Co.*, 819 F.2d 1100 (Fed. Cir. 1987) (a claim that depends from a nonobvious independent claim is nonobvious because it contains all of the limitations of that independent claim plus a further limitation).

2. Claim 2 Is Not Disclosed or Rendered Obvious by Agarwal

Claim 2 reads:

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.

Ex. 1001, 7:47-49.

The “table of requests” (TOR) is described as a centralized structure within the caching agent that stores pending requests from both the plurality of cores and the IIO module and is part of the caching agent’s logic for managing and arbitrating requests, as shown in Figure 2.

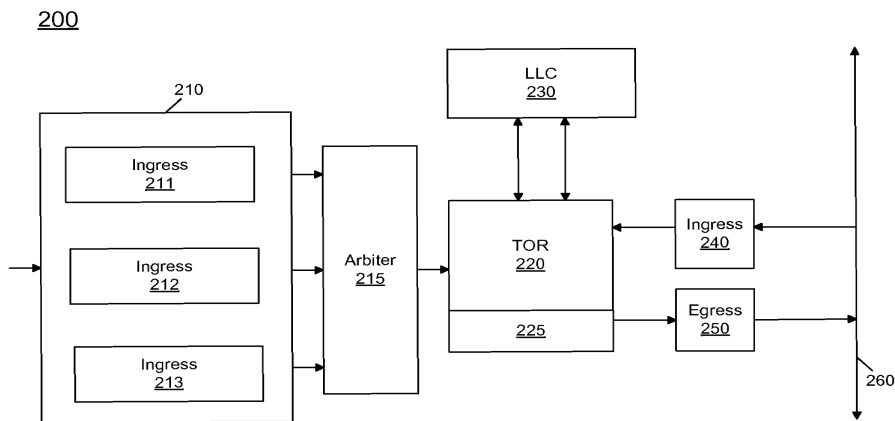


FIG. 2

Ex. 1001, 2:44–53, 2:66 – 3:3; Fig. 2; Ex. 2006, ¶ 76.

The petition argues that Agarwal discloses or renders obvious the TOR because Agarwal’s directory controller in each tile maintains a buffer

or queue of outstanding transactions. Pet., 24–25; Ex. 1003, ¶¶ 94–97, 204–213.

a) The “Table of Requests” Limitation Is Not Disclosed or Suggested by Agarwal

1. The ’228 Patent’s “Table of Requests” (TOR) Is a Specific, Centralized Structure.

The TOR should be understood as a single, shared resource within the caching agent and not a collection of separate, per-core or per-module buffers. Ex. 2006, ¶ 76. Figure 2 shows the TOR is a single, unified table within the caching agent, downstream of the arbiter (215) and upstream of the LLC (230), that arbitrates and manages requests from both the processor cores and the IIO module. All requests from the cores and the IIO module are funneled through this structure. Claim 2 does not state that “each core includes a table of requests” or that “each portion [of the single caching agent] includes its own table.” Instead, Claim 2 makes clear *the* caching agent (singular) includes a table of requests (singular) for all sources, which implies a unified, centralized structure within the caching agent that stores all incoming requests. *Id.*, and see, *Harari*, 656 F.3d at 1341.

2. Agarwal’s Buffers Are Not the Claimed “Table of Requests.”

In contrast to the recited TOR, in Agarwal each “small buffer” is a per-tile structure that tracks outstanding transactions for that tile’s directory

controller. Ex. 1005, 27:21–25. This makes sense because Agarwal describes an architecture that is fundamentally distributed, in which each tile manages its own directory state and outstanding transactions. *See, e.g., id.*, 27:21–25; 16:26–45; 27:21–25; Figs. 2, 6A/B. There is no disclosure in Agarwal of a centralized table or buffer that stores pending requests from both the processor cores and the I/O module, nor is there any suggestion that the per-tile buffers are shared resources in the manner required by Claim 2. Ex. 2006, ¶ 77-78.

3. Petitioner Conflates Distinct Structures

The Background section of the '228 patent notes that in systems where both cores and IIO modules issue requests, deadlocks can occur if posted requests (e.g., writes) are blocked by non-posted requests (e.g., reads), especially when both types of requests are funneled through a shared resource. The invention introduces the TOR within the caching agent, and it includes at least one dedicated resource (queue entry) for posted transactions. *See* Ex. 1001, 2:44–53; Fig. 2, el. 225. This dedicated resource ensures that posted requests are not blocked by non-posted requests, preventing deadlocks that would otherwise occur. Ex. 2006, ¶¶ 79-80.

In Agarwal, however, each tile has its own directory controller, and there is no centralized queue or table that receives requests from both cores and IIO modules. Instead, each directory controller manages state for its assigned address region independently. *Id.*, ¶ 81. The petition’s argument that Agarwal’s per-tile buffer is equivalent to the ’228 patent’s TOR ignores the requirement that the TOR is a shared structure for *all* sources, not a local buffer. There is no such shared structure in Agarwal. *Id.*

b) The Petition’s Reliance on “Obviousness” Is Legally and Technically Insufficient

1. “Obvious to Include a Buffer” Is Not Enough

The petition argues that it would be “obvious” to include a buffer or queue for outstanding requests, Pet. 24, but a mere buffer is not the same as the claimed “table of requests” that is shared, centralized, and arbitrates among both core and IIO requests. *Id.*, ¶ 82.

While buffers are sometimes used to track outstanding requests, and a processor, core, or module may have its own local buffer for this purpose, these are simple, per-source structures. *Id.*, ¶ 83. Indeed, Agarwal describes such a local buffer. But the “table of requests” in the ’228 patent is not just *any* buffer. It is a specific, shared structure within the caching agent that is designed to store pending requests from both the plurality of cores and the

IIO module. It is not per-core or per-module, but a unified table. It is part of the caching agent, not just a generic architectural feature. And it is structured to handle requests from multiple sources in a unified way. *Id.*, ¶ 83.

Obviousness under 35 U.S.C. § 103 requires that the prior art, either alone or in combination, would have led a POSITA to the claimed invention as a whole—not just to a generic concept (like “use a buffer”), but to the specific structure and function claimed. The ’228 patent’s TOR is designed (as explained in the context of the invention) to address system-level problems such as deadlock and ordering between posted and non-posted requests from multiple sources. A generic buffer or queue for outstanding requests does not, by itself, solve these problems or provide the same system-level coordination. *Id.*, ¶ 84.

2. No Evidence of a Need for or Benefit of a Unified Table Handling Both Core and IIO Requests

Agarwal’s architecture does not require, suggest, or benefit from a unified table of requests as claimed. *Id.*, ¶¶ 85-86. In Agarwal’s distributed arrangement, each tile has its own directory controller. Ex. 1005, Figs. 2, 6A/B; 16:38–45. Outstanding requests are tracked locally. *Id.*, 27:21–25 (“The tile can also keep a small buffer that contains a list of outstanding transactions along with an acknowledgement counter associated with

each.”). Accordingly, there is no architectural need for a single, unified table that tracks requests from all cores and I/O modules, and a POSITA would not adopt a modification that deliberately departs from that scheme. Rather than routing requests to the appropriate controller based on address mapping, a unified table would require those routings to be aggregated at a single entity, introducing scalability issues and defeating the purpose of the distributed approach. Ex. 2006, ¶ 86.

3. The Petition’s Use of Secondary References (e.g., Blumrich) Is Misguided

Petitioner argues a buffer would be beneficial and cites Blumrich for the idea of a request queue. Pet., 25. But in Blumrich, these track requests at a local—i.e., an individual directory controller or cache—level and are not shared structures for multiple sources. Ex. 1007, [0028], [0030] (request queue 302 is attached to each L2 slice, so it is not centralized). Hence, adopting such a queue to modify Agarwal’s directory controller might provide a local buffer for tracking requests handled by a tile’s directory controller (the buffer would be local to each tile), but there still would not be a single, unified table within a single caching agent that stores pending requests from both the plurality of cores and the IIO module. Instead, the architecture would remain distributed, with each directory controller

managing requests for its assigned memory region. Thus, modifying Agarwal to include a per-tile buffer (even if inspired by Blumrich) does not create a shared, unified table of requests as required by Claim 2. Ex. 2006, ¶¶ 87-89. The petition’s contention that Claim 2 is obvious in view of Agarwal should be rejected.

3. Claim 6 Is Not Obvious in View of Agarwal

Claim 6 reads:

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.

Ex. 1001, 7:64-67.

An “allocation transaction,” in the context of the ’228 patent, is a transaction initiated by an IO device (via the IIO module) that causes data to be directly stored (allocated) into the shared cache memory of the multicore processor, rather than first being written to system memory. Ex.1001, 3:45–58; Fig. 3; Ex. 2006, ¶ 91. The “allocation transaction” thus makes IO data quickly accessible to the processor cores, reducing memory access overhead. Petitioner argues that Claim 6 is obvious in view of Agarwal. Pet., 28-30.

a) *Agarwal Does Not Disclose the Required Transaction Chain*

Agarwal describes two cases for I/O module transactions: A “memory coherence case,” and a “cache coherence case.” In the memory coherence case, an I/O module writes data to main memory. The process is initiated from a tile (i.e., a core) and cache lines are updated accordingly, if present. In the cache coherence case, an I/O module writes data directly into a tile’s cache, and memory and other caches are updated according to the coherence protocol. Ex. 1005, 28:9–32; 29:5–30:11; Ex. 2006, ¶ 93. Petitioner equates the “cache coherence case” to the “allocation transaction” of claim 6. Pet., 29-30.

Petitioner is wrong. Agarwal does not disclose the full I/O device-to-I/O module-to-single caching agent flow required by Claim 6. In Agarwal, the I/O module (912) “includes an I/O interface 913 . . . 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).” *Id.*, 28:10–22 (emphasis added). That is, Agarwal’s “cache coherence case” concerns *core-initiated* DMA transactions, not transactions initiated by external IO devices. Ex. 2007,

69:18 – 70:20. Claim 6’s requirement that the allocation transaction be “received in the IIO module *from an IO device* coupled to the multicore processor” is missing. Ex., 2006, ¶ 94.

Neither the petition nor Dr. Hassoun’s declaration address this gap. Thus, Claim 6 has not been shown to be obvious in view of Agarwal.

4. Claim 11 and Its Dependent Claims Are Not Obvious in View of Agarwal

Claim 11 recites a method that includes:

[11a] “receiving an allocation transaction in an integrated input/output (IIO) module ... from an IO device ... the allocation transaction ... a request to directly store data ... into a shared cache memory”

[11b] “forwarding the allocation transaction to a caching agent ... the caching agent shared by a plurality of cores ... and the IIO module”

[11c] “directly storing the data of the allocation transaction into the shared cache memory”

Ex. 1001, 8:12-23.

Like Claim 6, “allocation transaction” semantics are baked into Claim 11’s steps. The “allocation transaction” is a request *from an IO device*,

received at the IIO, that is specifically for directly storing data into the LLC, bypassing system memory. This is more than a write that ultimately causes allocation” it is a transaction type and dataflow defined by the ’228 patent. Ex., 2006, ¶ 97.

ARM cites Agarwal’s “ProxyIO” example, in which an I/O module delivers a write to the “home tile,” whose directory controller then does normal coherence and can place data in the L2/L3 cache. Pet., 32. But, as detailed above, Agarwal does not teach an *I/O-originated “allocation transaction”* in the ’228 patent sense. Petitioner treats any I/O write that ends up in cache as an “allocation transaction,” which over-reads the term. Ex., 2006, ¶ 98. Agarwal’s ProxyIO has writes reaching cache memory but never teaches an I/O-originated “allocation transaction” with the specific semantics and flow the ’228 patent requires.

Further, Agarwal does not teach “forwarding ... to a caching agent shared by a plurality of cores and the IIO module,” step [11b]. Even without the “single” caching agent limitation found in claim 1, claim 11 still requires forwarding the allocation transaction to “*a* caching agent” that is *shared* by multiple cores and the IIO. The IIO proxies through the CPU caching agent.

That is, the IIO forwards the allocation transaction to the caching agent that services both CPU and IIO traffic. Ex. 1001, 3:45 – 4:31.

ARM’s mapping of Agarwal equates Agarwal’s per-tile, address-hashed “home tile” and its directory controller with the ’228 patent’s “caching agent,” asserting that sending an I/O write to a home tile counts as “forwarding ... to a caching agent” that’s “shared by a plurality of cores and the IIO module.” But this ignores Agarwal’s actual disclosure. Pet., 33.

In Agarwal, an I/O module delivers a transaction to whichever home tile chosen by a hash of the associated address and the directory controller on that tile handles coherence operations. Ex. 2006, ¶ 100. Agarwal does *not* describe this flow as forwarding to a caching agent that is a shared CPU/IIO agent; it’s just normal directory-based coherence at whichever home tile is associated with the target memory address. By contrast, the ’228 patent teaches proxying IIO traffic a shared CPU/IIO coherency agent, with the IIO explicitly forwarding the allocation transaction to that shared agent. That proxying step (and the identity of the recipient) matters for [11b] and is missing from Agarwal. *Id.*, ¶¶ 101-102. Petitioner’s directory as caching agent equivalence is a litigation construct, not a teaching of Agarwal, and it

does not satisfy the method step as claimed. Accordingly, Claim 11 is not obvious in view of Agarwal. *CFMT, Inc.*, 349 F.3d at 1342.

Claims 12-14 depend from Claim 11 and, therefore, are not obvious in view of Agarwal for at least the same reasons as Claim 11. *Hartness Int'l.*, 819 F.2d 1100.

5. Claim 14 is Not Obvious in View of Agarwal

Claim 14 recites:

“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.

Ex. 1001, 8:33-36.

The requirement of direct storing “without communicating with the system memory” at the time of the direct storing is narrower than merely “not sending a transaction” during later core accesses (Claim 12). The specification teaches that the invention bypasses system memory bandwidth during this direct store and writes back to memory only upon eviction. Ex. 1001, 3:45–4:11, Fig. 3, els. 330-360; Ex. 2006, ¶ 103 .

The petition cites Agarwal’s “cache coherence case” (Fig. 9C) to assert that I/O writes go “directly into the shared L3 cache.” Pet., 37, but

does not identify any explicit teaching in Agarwal that write occurs “without communicating with the system memory.” See Pet. §V.A.10, citing Agarwal 28:23–32; 29:5–30:11; Fig. 9C. Dr. Hassoun likewise treats Claim 14 summarily, cross-referencing his analysis for Claims 7/11/12, but does not identify where Agarwal affirmatively teaches the “without communicating” constraint, nor does he provide a reasoned motivation why a POSITA would implement a no-memory-communication store path. Ex. 1003, ¶¶ 286-288. In fact, Agarwal does not explicitly disclose “without communicating with the system memory” at the time of a direct cache write. The ProxyIO “cache coherence case” does have an I/O module “writ[ing] data directly into the L2/L3 cache,” but there is no mention or suggestion that this occurs without any communication with system memory as Claim 14 expressly requires. Citing a “write into L3” does not equal a disclosure that it is done without communicating with memory.

Obviousness requires more than silence. That is, obviousness typically requires a showing that the feature is expressly or necessarily present in the prior art, or that a POSITA would have been motivated to adopt it (with a reasonable expectation of success), not merely that the art doesn’t say otherwise. *See, e.g., KSR*, 550 U.S. at 415-421. Claim 14’s timing matters.

The “without communicating” constraint attaches to the store step, not only to later accesses. Petitioner points to Claim 12-type behavior (multiple core accesses without sending a memory transaction) and then infers Claim 14 without identifying any passage where Agarwal affirmatively characterizes the direct I/O-to-cache write as entirely non-communicative with system memory. Pet., 37. That conflates different timing constraints. Ex. 2006, ¶ 106.

Further, while Agarwal identifies a memory-write case and a cache-injection case, the existence of the cache-injection alternative does not, without more, establish that the cache-injection path is implemented “without communicating with the system memory.” *Id.*, ¶ 107. The petition assumes that because data lands in cache, no memory communication occurs, but it does not cite any statement in Agarwal tying the cache-injection path to the no-communication constraint that claim 14 explicitly imposes.

Claim 14’s requirement that a direct store occur “without communicating with the system memory” is anchored in the ’228 patent’s express teaching of bypassing memory bandwidth on the store path. Ex. 1001, Fig. 3; 3:43–4:11. Petitioner has not demonstrated that this is taught or

suggested by Agarwal and the petition's conclusory and derivative treatment of Claim 14 does not support a conclusion of obviousness.

B. The Challenged Claims are Not Unpatentable in View of Sinharoy

1. Overview of Sinharoy

Sinharoy is a technical journal paper that describes IBM's POWER7 server processor; a device with an eight-core processor and a memory subsystem that contains three levels of on-chip cache. Ex. 1006, 1, 16. Each core has private L1 and L2 caches, and all cores share a logically unified 32MB L3 cache, physically implemented as eight 4MB local L3 regions, each with its own centralized controller. *Id.*, 16, 19. The chip also includes two integrated I/O controllers, supporting DMA cache injection, and a high-bandwidth on-chip coherence fabric that connects all cores, caches, memory controllers, and I/O controllers. *Id.*, 22-23; Ex. 2007, 71:13 – 4:5.

2. Claim 1 is Not Obvious in View of Sinharoy

Claim 1 requires a single caching agent for a multicore processor, implemented in a plurality of distributed portions each associated with a corresponding core. That single caching agent performs cache coherency

operations for all the cores and an I/O module. Ex. 1001, 7:36-36; Ex. 2006, ¶ 113.

Sinharoy's POWER7 architecture is fundamentally different: Sinharoy employs multiple independent controllers (for L2, L3, and I/O), each acting as a peer on a broadcast coherence fabric, with no single agent managing coherency for both CPU and I/O domains. Ex. 1006, 1-2, 16-21, 22-23; Ex. 2006, ¶¶ 109, 114. Each L3 region has its own controller and resources, Ex. 2007, 73:19 – 74:5, and each L3 region's controller provides its own dispatch ports and pools of state machines—confirming per-region controllers, not one agent spread across slices. Ex. 1006, 19, 23; Ex. 1003, ¶ 326; Ex. 2006, ¶¶ 109-110, 114. Coherency is enforced by a broadcast coherence fabric that all agents (memory controller, L2/L3 caches, I/O controllers) snoop, **as peers**. Ex. 1006, 19, 23; Ex. 2006, ¶ 114.

The individual controllers of Sinharoy's L3 cache regions are not described as distributed portions of a single caching agent. Rather, each L3 region controller operates independently, and the coherence protocol is enforced collectively by all peer agents via the broadcast fabric. Ex. 1006, 19, 23; Ex. 2006, ¶ 114. The I/O controller also participates as a peer, not as a subordinate or proxy of a CPU caching agent. Ex. 1006, 22-23. This

presents a distinct contrast to the architecture recited in Claim 1 of the '228 patent. Ex. 2006, ¶¶ 114-117. Contrary to the “single caching agent” of Claim 1, Sinharoy’s peer controllers do not present a unified control plane or a single agent responsible for all coherency actions for both CPU and I/O traffic.

Further, the message passing between the different controllers does not make these distinct units a “single” caching agent. Ex. 2006, ¶ 116. The controllers exchange information over the broadcast fabric, but each cache region’s controller still performs coherency operations locally and independently. The fabric does not maintain cache-line state for all cores/I/O. *Id.*

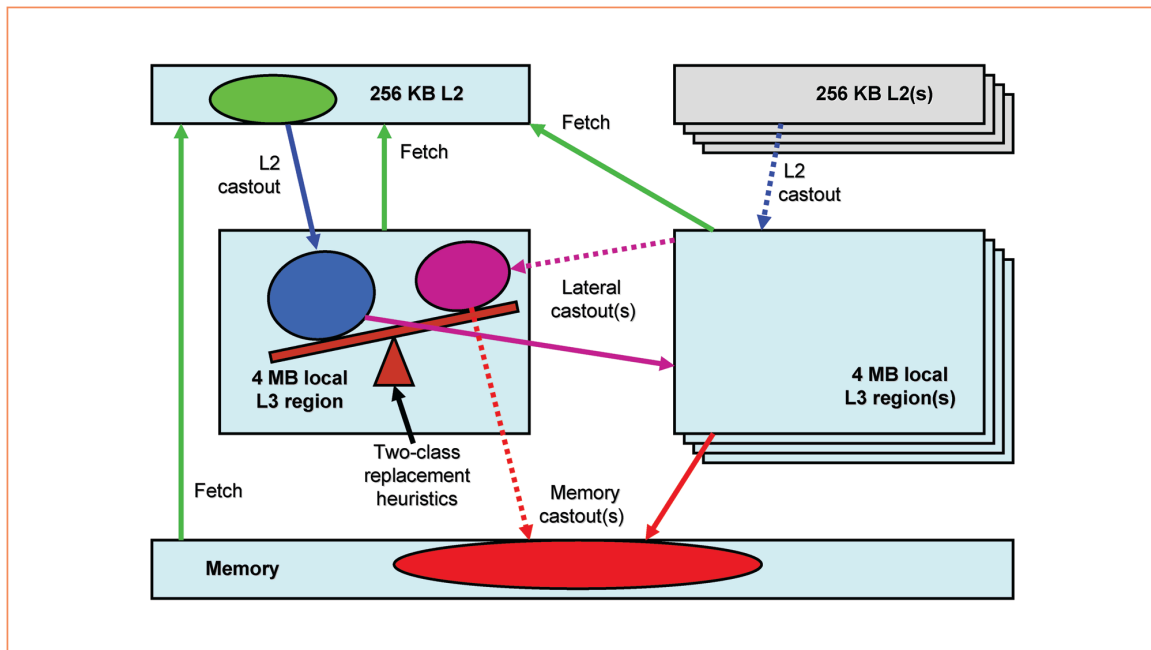
Thus, Claim 1 is not obvious in view of Sinharoy. Ex. 2006, ¶ 117. Further, because Claims 4, 6, and 7 depend from Claim 1, those claims are likewise not obvious in view of Sinharoy. *Hartness Int’l.*, 819 F.2d 1100.

3. Claim 4 is Not Obvious in View of Sinharoy

Claim 4 recites an association between a shared cache memory (LLC) that includes distributed portions, and a corresponding one of a plurality of

core. Ex. 1001, 7:53-56. Sinharoy fails to disclose such an association. Ex. 2006, ¶¶ 118-122.

The POWER7 L3 cache is divided into eight regions, but the individual regions are not tied one-to-one to specific cores. Instead, each L3 region is accessible by any of the cores. This is made possible by lateral L3 region dispatch ports included in each L3 region's individual cache controller. Ex. 2006, ¶ 120; Ex. 1006, 19; Ex. 2007, 74:6 – 75:13, 77:20 – 78:16.



In Figure 10, reproduced above, Sinharoy shows how a core can cast out data from an L2 cache to a 4MB L3 cache region. But rather than writing

back that data to memory, that 4MB L3 region can laterally cast out the data to another local 4MB L3 region and the original core (it's L2) can then fetch that data from the 4MB L3 region where the data now resides. The core is not limited to fetching from only an "associated" or "tightly coupled" L3 region. Ex. 1006, 19-20; Ex. 2006, ¶¶ 120-122.

The presence of a lateral ingress path for each L3 region is a hallmark of an architecture geared to accept non-local requests (i.e., from other cores). Ex. 2006, ¶ 120; Ex. 1006, 19. ("[O]perations that miss the 4-MB local L3 region but hit in the remaining 28 MB of the L3 cache access the cache via the snoop dispatch ports. L2 cast-out operations access the 4-MB local L3 region via the core/L2 dispatch port, whereas lateral cast-outs access a given 4-MB L3 region via the lateral L3 region dispatch port."). This is consistent with a shared (not per-core bound) LLC, and inconsistent with the claimed per-core association recited in Claim 4. Ex. 2006, ¶ 122.

Further, there is no motivation to repartition POWER7's L3 into per-core portions. Petitioner provides no reasoned explanation or expectation of success for modifying POWER7's address-homed, shared L3 into a per-core association as required by the claim. The lateral port in each POWER 7 L3 cache region exists precisely to handle non-local accesses and restricting the

L3 to the per-core association of Claim 4 would eliminate such access. As this would contradict the principle of operation articulated in Sinharoy, such a modification is non-obvious. Ex. 2006, ¶ 121; *In re Ratti*, 270 F.2d at 813.

4. Claim 6 is Not Obvious in View of Sinharoy

Claim 6 requires the caching agent receive an allocation transaction from the IIO module. The allocation transaction is received in the IIO module from an I/O device. Ex. 1001, 7:60-64. Sinharoy does not disclose this message flow. Ex. 2006, ¶¶ 123-125.

The ‘228 patent defines an allocation transaction as a request to directly store data into the LLC (i.e., to allocate a line in the LLC for I/O data), sent from the I/O device to the IIO, and forwarded by the IIO to the caching agent. Ex. 1001, 3:45 – 4:4.

In Sinharoy’s “cache injection,” DMA traffic is directed to the correct physical L3 slice based on an address-homing function. Ex. 1006, 17; Ex. 2007, 80:4 – 81:16. The two-hop message flow, I/O Device → IIO → Caching Agent, recited in Claim 6 is missing. Instead, the I/O controllers implement cache injection by writing into an L3 region when that region already owns the line. There is no intermediate recipient that functions as a

“single” caching agent to which the IIO forwards a distinct allocation request. Ex. 1006, 17, 22–23; Ex. 2006, ¶ 125. Accordingly, Claim 6 is not obvious in view of Sinharoy.

5. Claim 11 is Not Obvious in View of Sinharoy

Claim 11 recites similar allocation semantics as Claim 6: (1) receiving an allocation transaction in an IIO module from an I/O device, (2) forwarding the allocation transaction to a caching agent shared by the cores and IIO module, and (3) directly storing the data into a shared cache memory. Ex. 1001, 8, 11-21. As discussed above with respect to Claim 6, Sinharoy does not describe such a two-hop, device-originated allocation transaction that is forwarded to a caching agent. Ex. 2006, ¶¶ 126-127.

In Claim 11, the same allocation transaction (i.e., a request to directly store into LLC) that arrived in an IIO module is forwarded to the caching agent. In Sinharoy’s cache injection, DMA writes are direct into an L3 region, Ex. 1005, 17, 22–23, and there is no intermediate caching agent to which the IIO module forwards the write. Instead, the I/O controller implements cache injection locally, based on cache line ownership.

Accordingly, Claim 11 is not obvious in view of Sinharoy. Ex. 2006, ¶ 127. *CFMT, Inc.*, 349 F.3d at 1342.

Claims 12-14 depend from Claim 11 and, therefore, are not obvious in view of Sinharoy for the same reasons as Claim 11. *Hartness Int'l.*, 819 F.2d 1100.

C. Claims 2, 3, and 16 Are Patentable Over Sinharoy and Blumrich

Blumrich describes a multiprocessor system with an L2 cache-based point of coherence. Ex. 1007, Abst., [0018]-[0027], [0080]-[0081], [0092]-[0098]; Ex. 2006, ¶ 128.

1. Claim 2 is Not Obvious in View of Sinharoy and Blumrich

Claim 2 recites a table of requests (TOR) in the single caching agent to store pending requests for the plurality of cores and the IIO module. Ex. 1001, 7:47-49. In its proposed Sinharoy-Blumrich combination, Petitioner argues that Sinharoy's L2/L3 controllers, which provide pools of "read/claim (RC) machines," "write machines," and related resources, are the recited TOR, or, alternatively, that a POSITA would use a queue, as

taught in Blumrich to modify Sinharoy to include such a TOR. Pet., 60, 67-68. Petitioner is wrong.

First, the RC and write machines in Sinharoy are controller pipelines, not a unified table within a single caching agent. Ex. 2006, ¶¶ 130, 132. More specifically, they are state machines that manage sequences of cache coherence operations. The “machine” is what a controller uses to execute a transaction after it has been scheduled. *Id.*

In Sinharoy, rather than having a single controller handling all possible 13 coherence states (see Table 2, p. 17) simultaneously, the specialized RC and write machines manage different transaction types (read/claim vs. write). *Id.*; Ex. 1006, 19. Nowhere does Sinharoy describe these state machines as storing a population of pending transactions, and so they are not the TOR recited in Claim 2. Ex. 2006, ¶ 132.

The TOR of the '228 patent is “a buffer or other temporary storage for holding incoming requests,” with dedicated posted-transaction capacity. Ex. 1001, 2:44-52. In the embodiment of Fig. 2, ingress queues feed an arbiter, which then inserts incoming (core/I/O) transactions into the TOR before the LLC/controller logic acts on them. *Id.*, 2:53-65. Thus, Claim 2 requires a data structure that stores pending requests, i.e., a queue/buffer at the front

door of the caching agent shared by the requestors (cores and IIO). Ex. 2006, ¶ 131.

In Blumrich, the L2 contains a “request queue 302” that “buffers incoming read and write requests,” with a directory pipe and conflict-checking for speculative/TM operation. Ex. 1007, [0128]. This is a per-slice L2 queue, not an I/O queue nor an L3 caching-agent structure. Petitioner’s argument that a POSITA would have been motivated to modify Sinharoy’s L3 controller to include such a request queue and that this queue would serve as the claimed TOR, Pet., 67-68, is mistaken.

First, Blumrich’s request queue is not a table for mixed core/IIO requests in a caching agent. It buffers incoming read and write requests from the processor cores and is used for speculative execution and transactional memory management. Ex. 1007, [0128]-[0135]. There is no suggestion that this queue is used to buffer or store requests from an IIO module (i.e., from DMA or external I/O sources). Ex. 2006, ¶ 134.

Second, even if one considers Sinharoy’s L3 controller as handling both core and IIO traffic, via cache injection, there is no teaching of a unified ingress queue or table that stores pending requests from both sources before execution. The assertion that a POSITA would “obviously”

implement Sinharoy's L3 controller with a request queue like Blumrich's is a conclusion, not a reason for same. Sinharoy's "pool of operational resources" (RC/write machines) are execution resources, not ingress buffers, and are not described as holding pending requests from both sources, nor is there any reason for same. Blumrich does not teach or suggest that its request queue is suitable for, or should be adapted to, handle both core and IIO requests in a single table at the L3 controller level. Its queue is L2-centric and focused on speculative execution management. *Id.*, ¶ 135.

Third, a Sinharoy-Blumrich system would not have a TOR for both cores and an IIO module. Blumrich's queue is not shown to handle IIO requests and adapting it to do so would require significant architectural changes not suggested by either reference. Moreover, modifying Sinharoy's L3 controller to track all pending requests, including those from the IIO module is not a simple substitution. Sinharoy's architecture does not require or suggest a single ingress queue for both types of requests; its execution resources are provisioned to handle requests as they arrive, and there is no disclosure of a unified pending request table for both sources. *Id.*, ¶ 136.

The alleged motivation to combine, Pet., 67, does not address the specific technical challenges or architectural differences between L2

speculative request management (Blumrich) and L3 cache injection/multisource traffic (Sinharoy). There is no evidence that a POSITA would have reasonably expected that simply adopting a Blumrich-like queue as part of Sinharoy's L3 controller would result in a system that meets the claim—i.e., a single caching agent with a table of requests for both cores and IIO module. *Id.*, ¶ 137.

Thus, Claim 2 is not obvious in view of Sinharoy-Blumrich.

2. Claim 3 is Not Obvious in View of Sinharoy and Blumrich

Claim 3 further limits the recited TOR by requiring it to have “at least one dedicated entry to store a posted transaction.” Ex. 1001, 7:50-52. The specification explains that the “dedicated resource” can be an entry allocated for one-way, non-completion I/O writes (“posted”). *Id.*, 2:49-50. The dedicated entry ensures that posted transactions are not blocked by non-posted requests, which is important for deadlock avoidance and system performance. *Id.*, 5:41-45.

As noted above, Sinharoy and Blumrich do not teach or render obvious a caching agent that includes a table of requests to store pending requests for the plurality of cores and the IIO module. Furthermore, the

combination of these references does not teach or suggest the specific TOR with a dedicated entry specifically for posted transactions (i.e., write operations that do not require a completion/acknowledgment), as recited in Claim 3. Ex. 2006, ¶¶ 141-151.

As noted above, Blumrich's request queue 302 is a general-purpose buffer for incoming read and write requests from processor cores. Ex. 1007, [128]; Ex. 2007, 26:15-21. There is no teaching or suggestion that any entry in the queue is "dedicated" for posted transactions, nor does Blumrich discuss posted I/O semantics at all. All entries are available for any type of request, and there is no mention of reserving or dedicating an entry for a particular class of transaction (such as posted I/O writes). Ex. 2006, ¶ 142.

Sinharoy's "cache injection" does not rely on or even make use of a table of requests, let alone one with a dedicated entry for posted transactions. Ex. 1006, 17; Ex. 2006, ¶ 143. Moreover, Sinharoy's L3 controller uses execution resources (RC/write machines), but these are not described as a queue or table with reserved entries for posted transactions. Ex. 2006, ¶ 143.

The petition offers no specific motivation to combine Sinharoy and Blumrich, Pet., 68-69, and does not address specific technical challenges or architectural differences between L2 speculative request management

(Blumrich) and L3 cache injection/multisource traffic (Sinharoy) that a POSITA would face in attempting such a combination. Dr. Bagherzadeh explains that the POSITA would need to make architectural changes to accommodate a unified ingress buffer (the TOR), implement a specific data structure (e.g., a queue or array) that could hold and identify multiple pending requests, with at least one entry specifically reserved for posted transactions, configure the system to identify posted transactions and route them to the dedicated entry in the queue, and provide specific arbitration logic, integration paths and record keeping to accommodate these modifications. *Id.*, ¶¶ 145-150.

These are significant modifications that directly impact the design of the POWER7, yet the petition offers no details concerning same or what knowledge of a POSITA would be available to make such changes without the need for experimentation and while having an expectation of success. For example, no consideration is given to overcoming incompatibilities in transaction completion semantics. A general, unspecified motivation to “avoid deadlocks or improve performance” provides an insufficient basis to conclude such modifications would be obvious.

Thus, Claim 3 is not obvious in view of Sinharoy and Blumrich.

3. Claims 16-18 are Not Obvious in View of Sinharoy and Blumrich

Claim 16 recites a system with two multicore processors, one of which includes a caching agent including a TOR with multiple entries for core or IIO pending requests, including at least one dedicated entry for a posted transaction, plus a peripheral controller. Ex. 1001, 8:42-64.

Accordingly, all of the reasons for patentability of Claims 2 and 3 discussed above apply equally to Claim 16. Ex. 2006, ¶ 152.

Claims 17 and 18 depend from Claim 16 and are patentable over Sinharoy and Blumrich for the same reasons as Claim 16. *Hartness Int'l.*, 819 F.2d 1100.

D. Claim 5 is Patentable Over Sinharoy and Sistla and Claim 19 is Patentable Over Sinharoy, Blumrich, and Sistla

Claim 5 recites, “The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.” Ex. 1001, 7:57-59.

Claim 19 recites, “The system of claim 16, wherein the first caching agent is a single caching agent for the first multicore processor and includes a plurality of distributed portions each associated with a corresponding one

of the first plurality of cores, the first shared cache memory comprises a last level cache (LLC) including a plurality of distributed portions each associated with a corresponding one of the first plurality of cores, and wherein the plurality of distributed portions of the first caching agent are coupled via a ring interconnect.” *Id.*, 9:9-17.

Claim 5 is alleged to be obvious in view of Sinharoy and Sistla, US PG PUB 2008/0126750 A1 (Ex. 1008), and Claim 19 is alleged to be obvious in view of Sinharoy, Blumrich, and Sistla. Pet., 74 *et seq.* These challenges rely on Sistla’s teachings regarding the use of a ring interconnect. *Id.*

Sistla describes a system and method for aggregating core-cache clusters in a scalable manner to produce multi-core processors. Ex. 1008, Abst. A scalability agent operates as an interface between an on-die interconnect and both multiple processor cores and memory. A scalability agent protocol enables multiple core-cache clusters to appear as one caching agent to external devices, handle local cache coherence, and partition work among clusters. *Id.*, [0018]-[0019].

Petitioner alleges that Sistla’s disclosure concerning a “ring interconnect” would be an obvious feature to include in the architecture of Sinharoy and Sinharoy-Blumrich. Pet., 75-78. But Sistla’s “ring” couples

clusters, not distributed portions of a caching agent. Ex. 1008, [0018]-[0019], [0030]; Ex. 2006, ¶ 158, Ex. 2007, 27:11-23, 29:16 – 30:1.

According to Sistla, interconnect 320, which “is configured as a ring interconnect (but may be a 2D mesh)” connects core-cache clusters (310...310_n). The scalability agent (SA) is associated with each cluster and address-partitioned into separate “SA0...SA3” units. Ex. 1008, [0030], [0034]. The SA (i) issues on-die snoops, (ii) aggregates snoop responses, and (iii) performs address conflict checks so a socket “appears as one caching agent” to an external interconnect. *Id.*, [0040].

Thus, Sistla’s scalability agent is a coordination/snoop-aggregation layer, not an entity “that performs cache coherency operations for the plurality of cores and the IIO module;” i.e., it is not a “caching agent” as recited in the challenged claims. Ex. 2006, ¶ 159. This is confirmed by Sistla, since for write-class transactions the “SA only sequences the address portion” and “does not contain any data buffers.” *Id.*, 160; Ex. 1008, [0065].

Petitioner’s theory of obviousness depends on conflating Sistla’s SA with the caching agent recited in Claims 1 and 19. The reference itself does not support such an interpretation; hence, Petitioner’s theory collapses and

Claim 5 and 19 are not obvious in view of Sinharoy and Sistla, or Sinharoy, Blumrich and Sistla, respectively, for at least these reasons.

IV. CONCLUSION

For at least the foregoing reasons, the Board should find that Petitioner has failed to prove the unpatentability of any of the challenged claims.

Respectfully submitted,

Dated: December 17, 2025

Ascenda Law Group, PC
2150 N First St., Suite 420
San Jose, CA 95131
Tel: 866-877-4883
Email: tarek.fahmi@ascendalaw.com

/Tarek N. Fahmi/

Tarek N. Fahmi
Reg. No. 41,402

WORD-COUNT CERTIFICATE

I certify that the foregoing document complies with the type-volume limitation of 37 C.F.R. § 42.24(a) and (b) and contains 10,084 words (including words in drawings created by the Patent Owner or as annotations added by the Patent Owner to images reproduced in this document) based on the word count indicated by Microsoft® Word for Mac v. 16.85, excluding those portions exempted by the rule.

/Tarek N. Fahmi/

Tarek N. Fahmi

Reg. No. 41,402

CERTIFICATE OF SERVICE

The undersigned hereby certifies that a copy of the foregoing:

PATENT OWNER'S RESPONSE

and its accompanying exhibits (if any) were served on 12/17/2025, by filing this document through the PTAB P-TACTS System as well as by delivering a copy via email directed to the attorneys of record for the Petitioner at the following addresses:

Kevin Anderson
Patrick D. McPherson
DUANE MORRIS LLP
901 New York Ave. NW
Suite 700 East
Washington, D.C. 20001

Email:
KPAAnderson@duanemorris.com
PDMcPherson@duanemorris.com

Cory C. Bell
J. Preston Long
Alexander M. Boyer

Finnegan, Henderson, Farabow,
Garrett & Dunner, LLP
Two Seaport Lane, 6th Floor
Boston, MA 02210-2001

Email:
cory.bell@finnegan.com
jp.long@finnegan.com
alexander.boyer@finnegan.com

The parties have agreed to electronic service in this proceeding.

Respectfully submitted,

Dated: December 17, 2025

Ascenda Law Group, PC
2150 N First St., Suite 420
San Jose, CA 95131
Tel: 866-877-4883
Email: patents@ascendallaw.com

/Tarek N. Fahmi/

Tarek N. Fahmi
Reg. No. 41,402