

IPR2025-00207

Declaration of Marwan Hassoun, Ph.D. Under 37 C.F.R. § 1.68 in Support of
Petition for *Inter Partes* Review of U.S. Patent No. 8,984,228

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Arm Ltd, MediaTek Inc.,

Petitioners

v.

Daedalus Prime LLC,

Patent Owner

Patent No. 8,984,228
Case No. IPR2025-00207

DECLARATION OF MARWAN HASSOUN, Ph.D., UNDER 37 C.F.R. § 1.68
IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF U.S.
PATENT NO. 8,984,228

TABLE OF CONTENTS

I. INTRODUCTION	1
II. BACKGROUND AND QUALIFICATIONS	3
III. MATERIALS CONSIDERED FOR THIS DECLARATION.....	5
IV. SUMMARY OF OPINIONS	5
V. OVERVIEW OF THE ‘228 PATENT AND ITS PROSECUTION HISTORY	6
A. Overview of the ‘228 patent.	6
B. Overview of the Prosecution History of the ‘228 patent	20
C. Overview of the IPR Filed Against The Child Patent Of The ‘228 Patent	23
VI. LEVEL OF ORDINARY SKILL IN THE PERTINENT ART	25
VII. CLAIM CONSTRUCTION	26
VIII. UNDERSTANDING OF THE LAW	27
IX. SCOPE AND CONTENT OF THE PRIOR ART	32
A. Agarwal	32
B. Sinharoy	53
C. Blumrich	56
D. Sistla	56
X. DETAILED INVALIDITY ANALYSIS	57

A. Structure Of The Challenged Claims And My Analysis Below.	58
B. Ground 1: Claims 1-2, 4-7, and 11-14 Are Obvious Over Agarwal.....	59
1. Overview Of Agarwal.....	59
2. Agarwal Discloses The Elements Of Claims 1-2, 4-7, And 11-14. .	59
a) Claim 1	59
i. 1[pre]: “An apparatus comprising:”	59
ii. 1[a]: “a multicore processor including a plurality of cores” ..	60
iii. 1[b]: “a shared cache memory”	64
iv. 1[c]: “an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor”	68
v. 1[d]: “a caching agent to perform cache coherency operations for the plurality of cores and the IIO module”	74
vi. 1[e]: “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.”	87
b) Claim 2: “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.”	94
c) Claim 4: “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.”	97

- d) Claim 5: “5. The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.. 101
- e) Claim 6: “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.” 103
- f) Claim 7: “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.”.....110
- g) Claim 11: 114
- a. Preamble: “11. A method comprising:” 114
- b. [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:” 114
- c. [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” 116
- d. [11c] “directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.” 118
- h) Claim 12: “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.” 118

a. Claim 13: “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” 123

b. Claim 14: “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.” 125

C. Ground 2A: Claims 1, 4, 6, 7, and 11-14 Are Obvious Over Sinharoy.....127

3. Overview Of Sinharoy 127

4. Sinharoy Discloses The Elements Of Claims 1-2, 4-7 And 11-14.127

i) Claim 1 127

vii. 1[pre]: “An apparatus comprising:” 127

viii. 1[a]: “a multicore processor including a plurality of cores”128

ix. 1[b]: “a shared cache memory” 130

x. 1[c]: “an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor” 130

xi. 1[d]: “a caching agent to perform cache coherency operations for the plurality of cores and the IIO module” ...138

xii. 1[e]: “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.”	153
j) Claim 4: “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.”	179
k) Claim 6: “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.”	182
l) Claim 7: “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.”	187
m) Claim 11:	188
a. Preamble: “11. A method comprising:”	188
b. [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:”	188
c. [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”	190
d. [11c] “directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.”	192

n) Claim 12: “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.”	192
a. Claim 13: “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”	195
b. Claim 14: “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.”	197
D. Ground 2B: Claims 2, 3, and 16-18 Are Obvious Over Sinharoy Combined With Blumrich.....	198
5. Overview of Blumrich	198
6. Motivation to Combine Sinharoy and Blumrich	198
7. Sinharoy Combined With Blumrich Discloses, And/ Or Renders Obvious, Claims 2, 3, and 16-18.	203
o) Claim 2: “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.”	203
p) Claim 3: “3. The apparatus of claim 2, wherein the table of requests includes at least one dedicated entry to store a posted transaction.”	207
a. Claim 16:	208
i. 16[pre] “A system comprising:”	208

- ii. 16[a] “a first multicore processor including a first plurality of cores,” 210
 - iii. 16[b] “a first shared cache memory,;” 210
 - iv. 16[c] “a first integrated input/output (IIO) module to interface between the first multicore processor and a first plurality of IO devices coupled to the first multicore processor, and;” 210
 - v. 16[d] “a first caching agent to perform cache coherency operations for the first plurality of cores and the first IIO module,;” 211
 - vi. 16[e] “wherein the first caching agent includes a table of requests having a plurality of entries each to store a pending request for one of the first plurality of cores or the first IIO module, including at least one dedicated entry to store a pending request corresponding to a posted transaction.” 211
 - vii. 16[f] “a second multicore processor including a second plurality of cores,” 212
 - viii. 16[g] “a second shared cache memory,” 213
 - ix. 16[h] “a second IIO module to interface between the second multicore processor and a second plurality of IO devices coupled to the second multicore processor, and” 213
 - x. 16[i] “a second caching agent to perform cache coherency operations for the second plurality of cores and the second IIO module; and” 214
 - xi. 16[j] “a peripheral controller coupled to at least one of the first and second multicore processors.” 215
- b. Claim 17: “17. The system of claim 16, wherein the first caching agent is to receive an allocation transaction from the first IIO module, the allocation transaction comprising a request to

directly store data of the allocation transaction into the first shared cache memory, and directly store the data of the allocation transaction into the first shared cache memory.” 216

c. Claim 18: “18. The system of claim 17, wherein the data in the shared cache memory is to be accessed via at least one of the first plurality of cores a number of times, without sending a transaction to a system memory coupled to the first multicore processor.” 217

E. Ground 2C: Claim 5 Is Obvious Over Sinharoy Combined With Sistra.....218

1. Overview of Sistra 218

2. Motivation to Combine Sinharoy and Sistra, including with Blumrich. 218

3. Sinharoy Combined With Sistra Discloses, And/ Or Renders Obvious, Claim 5.....222

q) Claim 5: “5. The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.” 223

F. Ground 2D: Claim 19 Is Obvious Over Sinharoy Combined With Sistra and Blumrich.....223

r) Claim 19: “19. 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.” 223

XI. SECONDARY CONSIDERATIONS DO NOT REBUT THE PRIMA FACIE CASE OF OBVIOUSNESS	225
XII. CONCLUSION.....	225

I, Marwan Hassoun, Ph.D. hereby declare as follows:

I. INTRODUCTION

1. I have been retained as an expert witness on behalf of Arm Ltd. and MediaTek Inc. (collectively “Petitioner” or “Petitioners”) for the above-captioned Petition for *Inter Partes* Review (“IPR”) of U.S. Patent No. 8,984,228 (“the ‘228 patent”). My billing rate for my time in connection with this IPR at my standard consulting rate of \$750 per hour. My compensation is in no way dependent on the outcome of this matter.
2. I have been asked to provide my opinions regarding whether claims 1-7, 11-14, and 16-19 (“the Challenged Claims”) of the ‘228 patent are invalid as obvious. In preparing this declaration, I have reviewed the ‘228 patent, the file history of the ‘228 patent, and numerous prior art references from the time of the alleged invention.
3. I have been advised and it is my understanding that patent claims in an IPR are given their meaning applying the district court “*Phillips*” approach to claim construction. As detailed below, I have analyzed the Challenged Claims under the *Phillips* construction. In the Understand Of The Law section below, I have stated the principles from *Phillips* that I have been advised should guide my analysis.

4. In forming the opinions expressed in this declaration, I relied upon my education and experience in the relevant field of the art, and have considered the viewpoint of a person having ordinary skill in the relevant art, as of December 13, 2011 that is the priority date listed on the face of the '228 patent. My opinions directed to the invalidity of the Challenged Claims are based, at least in part, on the following prior art publications which I understand qualify as prior art under pre-AIA 35 U.S.C. §§102A), (b), and/or (e):

Exhibit No.	Description
Ex-1005	U.S. Patent No. 7,805,575 (“ <i>Agarwal</i> ”), filed May 25, 2007 and issued September 28, 2010; prior art under §§102A), (b), and (e).
Ex-1006	B. Sinharoy et al., “IBM POWER7 Multicore Server Processor,” IBM Journal of Research and Development, vol. 5, no. 3, pp. 1:1-1:29 (2011) (“ <i>Sinharoy</i> ”); prior art under §102A),
Ex-1007	U.S. Patent Appl. Pub. No. 2011/0219188 (“ <i>Blumrich</i> ”), filed January 18, 2011 and published September 8, 2011; prior art under §§102A) and (e).

Exhibit No.	Description
Ex-1008	U.S. Patent Appl. Pub. No. 2008/0126750 (“ <i>Sistla</i> ”), filed November 29, 2006 and published May 29, 2008; prior art under §§102A), (b), and (e).

II. BACKGROUND AND QUALIFICATIONS

5. My background and expertise that qualify me as an expert in the technical issues in this case are as follows:
6. I earned a Bachelor of Science degree in electrical engineering from South Dakota State University in 1983, a Master of Science degree in electrical engineering from Purdue University in 1984, and a Ph.D. in electrical engineering, also from Purdue University, in 1988.
7. I have over 30 years of experience in academia and industry in integrated circuits, including field programmable gate arrays (FPGAs), processors and systems-on-chip (SOC) designs as well as products integrating high-speed and high-complexity analog, digital, and mixed signal circuits including, for example, processors, memories, analog-to-digital converters, digital-to-analog converters, and power management

circuits. My experience includes designing memories including cache memories for use by processors.

8. I was a Professor of Electrical and Computer Engineering at Iowa State University from 1988–2000, teaching and performing research in very-large-scale integration (VLSI) integrated circuits and am currently a Professor-of-the-Practice in the Electrical and Computer Engineering department at Tufts University teaching integrated circuits design.
9. In industry, I have held a variety of senior professional positions, at startups (e.g. RocketChips, KeyEye Communications) and leading semiconductor companies (e.g. Hewlett-Packard (HP), Xilinx).
10. I have been involved in building analog, digital, and mixed-signal integrated circuits products that involve processors, co-processors, analog-to-digital converters, digital-to-analog converters, and serializers-deserializers, and their integration into large complex integrated SOCs, FPGAs, and boards.
11. I am a named inventor on 59 issued patents and 4 pending patent applications related to highly integrated circuits with novel functionality in the areas of multi-gigabit serial transceivers, power conversion,

memories, analog-to-digital converters ADCs), digital-to-analog converters, RF devices, FPGAs, robotics, and digital signal processor filters. Additionally, I have over 70 publications that include papers on memories and processors.

III. MATERIALS CONSIDERED FOR THIS DECLARATION

12. In addition to my general knowledge, education, and experience, I considered the materials listed in **Exhibit B** in forming my opinions.

IV. SUMMARY OF OPINIONS

13. Based on my review of the '959 patent and its prosecution history, the other materials I have considered, and my knowledge and experience, my opinions are as follows:

- Claims 1-2, 4-7, and 11-14 are obvious over Agarwal (Ground 1).
- Claims 1, 4, 6, 7, and 11-14 are obvious over Sinharoy (Ground 2A).
- Claims 2, 3, and 16-18 are obvious over Sinharoy combined with Blumrich (Ground 2B).
- Claim 5 is obvious over Sinharoy combined with Sistla (Ground 2C).
- Claim 19 is obvious over Sinharoy combined with Blumrich and Sistla (Ground 2D).

V. **OVERVIEW OF THE ‘228 PATENT AND ITS PROSECUTION HISTORY**

A. **Overview of the ‘228 patent.**

14. The application that became the ‘228 patent was filed December 13, 2011 according to the face of the patent.

15. The ‘228 patent describes “Providing common caching agent for core and integrated input/output (IO) module.” ‘228 at Title.

16. In essence, the ‘228 patent recognizes what others recognized that, for cache memory accesses in a multi-core system that included input/output modules, a common caching agent could provide access to the cache for both processors and an I/O module within the multicore system. This was reflected in the Abstract:

In one embodiment, the present invention includes a multicore processor having a plurality of cores, a shared cache memory, an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor, and a caching agent to perform cache coherency operations for the plurality of cores and the IIO module. Other embodiments are described and claimed.

17. ‘228 patent at Abstract.

18. In its “Background,” the ‘228 patent describes existing cache solutions for multiprocessor systems and purported problems when IO components are added to such systems.

In modern multiprocessor (MP)-socket computer systems, various topologies are possible. Such systems often include many different semiconductor components realized as integrated circuits (ICs). The ICs include processors, memories, chipsets, input/output hubs (IOHs) and so forth.

As process nodes advance, greater amounts of functionality can be incorporated into a single semiconductor die. One integration trend is to integrate an IO component such as functionality of an IOH into a central processing unit (CPU) die. The main motivation for doing so is to reduce the bill of material (BOM) cost of a computer system, and enable small form factors while reducing overall power consumption.

But problems arise once an IO component is integrated on the same chip with a multiprocessor. Traditional IO integration treats the IO component as a separate caching agent, meaning that dedicated logic is associated with the IO component to handle cache coherency operations. When an IO agent is performing read/write operations to

main memory, it has to snoop the CPU side cache to maintain cache coherency. In MP systems, this becomes a major scaling problem. For example, in an 8 socket system, there are effectively 16 caching agents in the system, which can degrade performance. And the efforts to scale up a system to support these many caching agents are not trivial.

19. '228 at 1:7-30.

20. The '228 patent specifically identified an idea of using a common caching agent to manage cache coherency for both CPU and IO component accesses:

In various embodiments, an IO agent can be integrated into a multiprocessor socket such as a multicore processor for use in a MP server system, while avoiding scaling issues. More specifically, a CPU caching agent can be configured to support both CPU traffic and IO traffic as well, thereby resolving the scaling issue. Performance can be enhanced by bringing the IO agent and CPU core(s) closer. This IO agent includes functionality to provide an interface between one or more peripheral devices coupled to the processor by off-chip links such as Peripheral Component Interconnect Express (PCIe™)

links, and may take the place of a separate chipset component such as an IOH.

21. '228 patent at 1:53-64.

As seen, socket 110 0 includes a plurality of cores 120 00-120 0n. As examples, socket 110 0 can include 4, 8, or another such number of cores. Each of the cores can include various components including a processor pipeline having multiple stages such as a front end unit, one or more execution units, and a back end unit. In addition, one or more levels of cache memories can be present within the cores. Each of cores 120 may be coupled to a shared cache memory 130 0, which may be a last level cache (LLC). As seen, LLC 130 can include a caching agent 135 0. In various embodiments, this caching agent may be a combined caching agent both for the CPU as well as for an integrated IO agent. More specifically, socket 110 0 may include an IO module (hereafter an integrated IO module or IIO module). This IIO module may act as an interface to one or more off-chip peripheral or IO devices (not shown in FIG. 1) coupled to the processor, e.g., via PCIe™ links. Examples of such IO devices include storage devices, network interfaces, graphics cards, and so forth.

22. '228 patent, 2:8-35.

23. The '228 patent then describes exemplary components of its
caching agent:

Referring now to FIG. 2, shown is a block diagram of a caching agent in accordance with an embodiment of the present invention. As shown in FIG. 2, caching agent 200 may be logic interposed between one or more cores of a processor and a LLC 230. In addition, caching agent 200 may also provide an interface between an IIO module within a multicore processor and the LLC. As seen, caching agent 200 can include an ingress port 210 that includes multiple ingress queues 211-213. As seen, ingress port 210 may be coupled to an arbiter 215 which can arbitrate amongst incoming requests and provide them to a table of requests (TOR) 220. As seen, TOR 220 may be a buffer or other temporary storage for holding incoming requests. In the embodiment shown, TOR 220 may include at least one dedicated resource 225 for posted transactions. In one embodiment, dedicated resource 225 may be a single entry of the queue, although the scope of the present invention is not limited in this regard.

Still referring to FIG. 2, TOR 220 interfaces with LLC 230. More specifically, this LLC may be a bank or other portion of the LLC associated with the caching agent. To provide access to other locations within the processor via a ring interconnect 260, an ingress queue 240 and an egress queue 250 may be provided. Thus via caching agent 200, cores of the processor and an IIO module can maintain coherency without the need for additional caching agents. Although not shown for ease of illustration, understand that caching agent 200 may further include various coherency logic to perform cache coherency operations on behalf of the cores and IIO module, and to enable coherent access with regard to both the core transactions and IO device transactions as described herein.

24. '228 patent, 2:36-65.

25. The '228 patent then described one embodiment in which the common caching agent can have distributed portions.

Note that although a single structure is shown in FIG. 2 for ease of illustration, understand 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.

26. ‘228 patent, 2:65-3:3.

27. According to the ‘228 patent, such an approach for a common
caching agent between CPU and I/O devices provided certain benefits:

With this approach, the IIO module proxies through the CPU caching agent to access memory or other IO devices, therefore reducing the overhead of allocating dedicated resources for an integrated IO caching agent. This also reduces the amount of snoop traffic needed since a reduced number of caching agents per system can be realized. Thus in various embodiments, a system can include a single caching agent per multicore processor socket, where each socket includes multiple cores and an IIO module.

28. ‘228 Patent, 3:4-12.

29. As referenced above, the ‘228 patent purportedly solved a problem of
“separate caching agent[s]” for IO components by implementing a combined “distributed” caching agent. Claim 1 recites this alleged solution in element 1[e]:

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

30. '228 patent, 7:43-46.

31. In one embodiment, the '228 patent teaches the “distributed” caching agent corresponds to each “portion” managing a partition, bank or slice of the shared cache. E.g., '228 patent, 2:67-3:3 (“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.”).

32. As shown in the embodiment of FIG. 6, processor 700 includes a distributed configuration having partitions or slices each including a core 710 and a partition of a caching agent 715 and a LLC 720. Note that while distributed caching agents are shown, understand that these distributed portions form a single caching agent, and which is configured to handle cache coherency operations both for the cores as well as an IIO module 750.

33. '228 patent, 6:22-29. Thus, at least one embodiment of a “distributed” single caching agent having separate distributed portions associated with a slice / portion of LLC that collectively handle the cache coherency operations for the entire multicore and IIO module system.

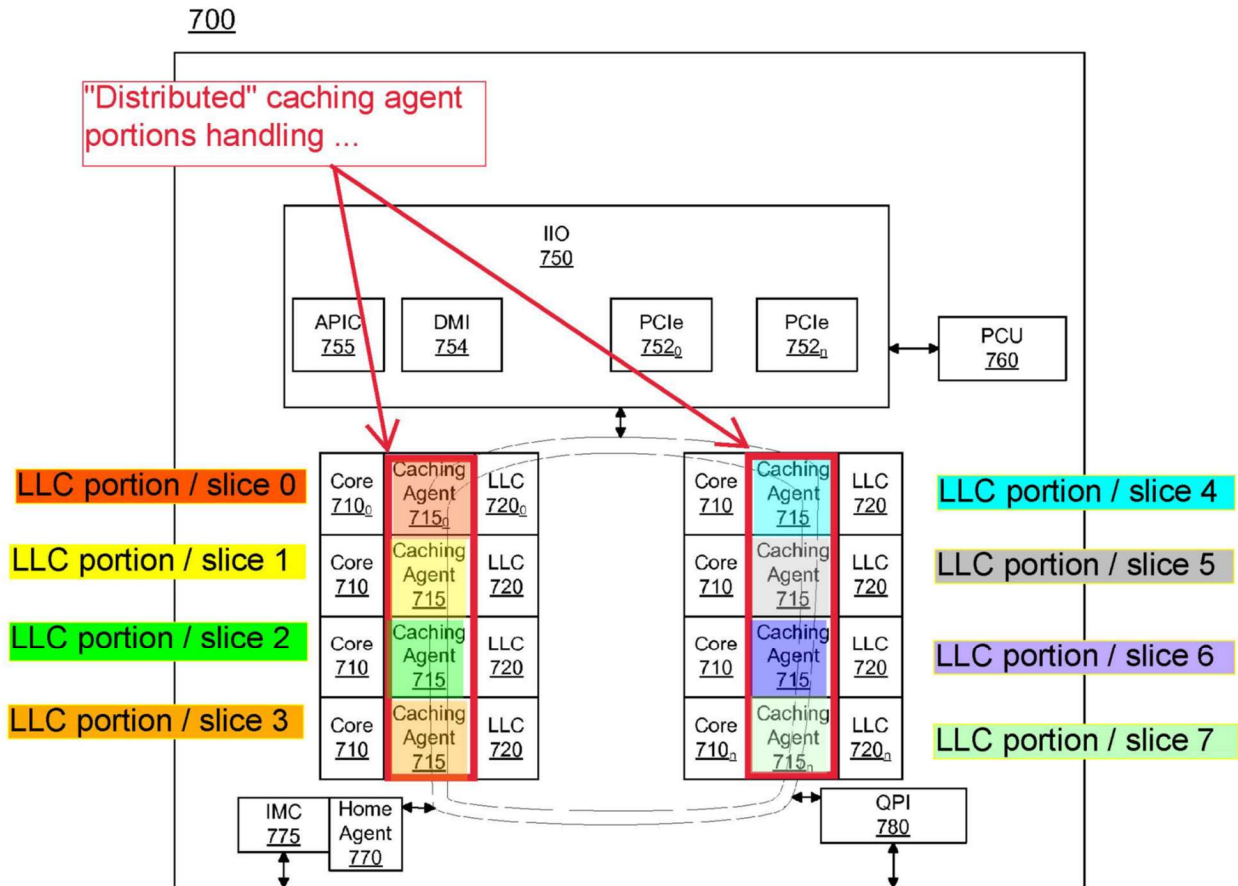


FIG. 6

34. '228 patent, Fig. 6; 6:20-39.

35. As issued, the '228 patent includes 19 claims. While I have been asked to opine only on the Challenged Claims as referenced above, I list all 19 claims in the '228 patent below for ease of reference:

US 8,984,228

1. An apparatus comprising:

a multicore processor including a plurality of cores, a shared cache memory, an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor,

and a caching agent to perform cache coherency operations for the plurality of cores and the IIO module,

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.

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.

3. The apparatus of claim 2, wherein the table of requests includes at least one dedicated entry to store a posted transaction.

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.

- | |
|---|
| 5. The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect. |
| 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. |
| 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. |
| 8. The apparatus of claim 1, wherein the caching agent is to send a drop ownership indication to the IIO module for a cache line owned by the IIO module for an IO device coupled to the multicore processor responsive to receipt of a snoop request in the caching agent for the cache line from another agent. |
| 9. The apparatus of claim 8, wherein the IO device prefetched the cache line. |
| 10. The apparatus of claim 8, wherein the another agent is one of the plurality of cores. |
| 11. A method comprising:

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;

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

directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.

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.

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.

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.

15. The method of claim 11, further comprising sending a drop ownership indication to the IIO module for a cache line owned by the IIO module responsive to receipt of a snoop request in the caching agent for the cache line from another agent.

16. A system comprising:

a first multicore processor including a first plurality of cores, a first shared cache memory, a first integrated input/output (IIO) module to interface between the first multicore processor and a first plurality of IO devices coupled to the first multicore processor, and a first caching agent to perform cache coherency operations for the first plurality of cores and the first IIO module, wherein the first caching agent includes a table of requests having a plurality of entries each to store a pending request for one of the first plurality of cores or the first IIO module, including at least one dedicated entry to store a pending request corresponding to a posted transaction;

a second multicore processor including a second plurality of cores, a second shared cache memory, a second IIO module to interface between the second multicore processor and a second plurality of IO

devices coupled to the second multicore processor, and a second caching agent to perform cache coherency operations for the second plurality of cores and the second IIO module; and a peripheral controller coupled to at least one of the first and second multicore processors.

17. The system of claim 16, wherein the first caching agent is to receive an allocation transaction from the first IIO module, the allocation transaction comprising a request to directly store data of the allocation transaction into the first shared cache memory, and directly store the data of the allocation transaction into the first shared cache memory.

18. The system of claim 17, wherein the data in the shared cache memory is to be accessed via at least one of the first plurality of cores a number of times, without sending a transaction to a system memory coupled to the first multicore processor.

19. 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.

B. Overview of the Prosecution History of the ‘228 patent

36.I have reviewed the prosecution history of the ‘228 patent in forming my opinions for purposes of this declaration.

37.As initially submitted, claim 1 recited the following:

What is claimed is:

- 1 1. An apparatus comprising:
- 2 a multicore processor including a plurality of cores, a shared cache memory, an
- 3 integrated input/output (IIO) module to interface between the multicore processor and at least
- 4 one IO device coupled to the multicore processor, and a caching agent to perform cache
- 5 coherency operations for the plurality of cores and the IIO module.

38.‘228 Pros. History, 2011-12-13 Application.

39. The examiner issued a rejection for the claims based upon U.S. Pat. Publication 2004/0139283 (“Arimilli”) for most claims and Arimilli combined with Ueda (U.S. Patent Application Publication 2008/0320236) for the remainder of the claims. ‘228 Prosecution History, Office Action, 2014-07-08.

40. In response, the Applicant amended the claims. For example, claim 1 was amended to include “distributed portions” that were identified in the original claim 4 as follows:

Listing of Claims:

Claim 1 (currently amended): An apparatus comprising:
a multicore processor including a plurality of cores, a shared cache memory, an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor, and a caching agent to perform cache coherency operations for the plurality of cores and the IIO module, 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.

41. ‘228 Pros. History, Amendment, 2014-10-08. In conjunction with the amended claims, the applicant argued that the prior art did not disclose the “distributed portions” that were part of the amendment.

Pending claims 1-3, 5-7, 9, 11, 17 and 20 stand rejected under 35 U.S.C. §102(b) over U.S. Publication No. 2004/0139283 Arimilli).

Applicants respectfully traverse the rejection. Regarding claim 1, Arimilli fails to anywhere disclose or even suggest that a multicore processor include a single caching agent that includes multiple distributed portions each associated with a corresponding one of multiple cores of the multicore processor. Here, contended support is

from paragraph 40 of Arimilli. However, neither here nor anywhere else in the reference is there any disclosure of distributed portions of a single caching agent. Rather, regarding paragraph 40, all that is described is that a processing unit includes a cache hierarchy that may include separate level one (L1) caches for each processor core, and a level two (L2) cache shared by multiple processor cores. The L1 caches thus do not meet the recited subject matter regarding a caching agent to perform cache coherency operations for an integrated IO module. And clearly, the shared L2 cache disclosure fails to anywhere address that this be in the form of distributed portions. Nor is there any disclosure of distributing a caching agent as set forth in the claim. Rather, all that Arimilli discloses is that each cache (namely the L1 and L2 cache) includes cache array, directory and controller. Nonetheless, nothing here nor anywhere else discloses the distributed arrangement. Accordingly claim 1 and the claims depending therefrom are patentable over the cited art.

42. '228 Pros. History, Amendment, 2014-10-8.

43. Based upon this amendment and the arguments in the October 8, 2014

Amendment, the claims issued.

Declaration of Marwan Hassoun, Ph.D. Under 37 C.F.R. § 1.68 in Support of
Petition for *Inter Partes* Review of U.S. Patent No. 8,984,228

Notice of Allowability	Application No. 13/324,053	Applicant(s) LIU ET AL.	
	Examiner EDWARD DUDEK JR	Art Unit 2136	AIA (First Inventor to File) Status No

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--
All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to the reply filed on 08 October 2014.
☐ A declaration(s)/affidavit(s) under **37 CFR 1.130(b)** was/were filed on _____.

44. '228 Pros. History, Notice of Allowability, 2014-11-12.

C. Overview of the IPR Filed Against The Child Patent Of The '228 Patent

45. I have reviewed materials from IPR2023-01344 which was filed against U.S. Pat. No. 9,575,895. In particular, I have reviewed the Petition, the Patent Owner's Response, The Petitioner's Reply, and the Patent Owner's Sur-reply.

46. I note that the Agarwal reference discussed below is not part of the IPR2023-01344 proceedings and that the Sinharoy reference is part of the IPR2023-01344 proceedings.

47. I note that the claims of the '895 patent challenged in IPR2023-01344 have similarities to, and overlapping terms with the '228 patent claims. For example, the table below compares claim 1 of the '895 patent with claim 1 of the '228 patent:

'895 patent	'228 patent
1. A processor comprising: a plurality of cores, a shared cache memory, a memory controller to interface with a memory coupled to the processor, an integrated input/output (IIO) module to interface between the processor and an IO device coupled to the processor and a caching agent to perform cache coherency operations for the plurality of cores and the IIO module, wherein the processor is to receive an allocation transaction from the IO device and directly store data of the allocation transaction into the shared cache memory, wherein the caching agent is a single caching agent for the processor and includes a plurality of	1. An apparatus comprising: a multicore processor including a plurality of cores, a shared cache memory, an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor, and a caching agent to perform cache coherency operations for the plurality of cores and the IIO module, 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.

distributed portions each associated with	
a corresponding one of the plurality of	
cores.	

48. As is apparent from the comparison above, claim 1 of the ‘895 patent has, effectively, all of the elements of claim 1 of the ‘228 patent. While the ‘895 patent does not expressly recite “multicore processor,” the ‘895 patent does recite that its processor has “a plurality of cores” which appears to describe a “multicore processor.”

49. As I have discussed below, the IPR2023-01344 proceedings have discussed the meaning of the claim terms in the ‘895 patent related to “a single caching agent for the processor and includes a plurality of distributed portions each associated with a corresponding one of the plurality of cores.”

VI. LEVEL OF ORDINARY SKILL IN THE PERTINENT ART

50. I have been advised that there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including the educational level of active workers in the field at the time of the invention, the sophistication of the technology, the type of problems encountered in the art, and the prior art solutions to those problems.

51. It is my opinion that a person of ordinary skill in the art (“POSITA”) in December 2011 would have (1) held at least a bachelor’s degree in electrical engineering, computer engineering, physics, or a similar field, with a focus on semiconductor technology and/or microelectronics, and (2) had approximately two years of industry or academic experience in a field related to computer architecture design and development, including experience with cache management in multicore systems-on-a-chip (SoCs). A POSA would have gained knowledge of these concepts also through a mixture of training and work experience, such as by obtaining a Master’s degree in those fields with equivalent work experience being substituted for any formal education requirement.

VII. CLAIM CONSTRUCTION

52. After reviewing the claims and the terms therein, I note the following.

53. No specific claim term needs to be completely defined for purposes of my invalidity analysis because the prior art discussed herein because, as detailed below, the prior art discloses architectures and functionality corresponding to that disclosed in the ‘228 patent. Thus, any construction of a claim term that would encompass the ‘228 patent disclosure would also encompass the prior art discussed herein.

VIII. UNDERSTANDING OF THE LAW

54. In expressing my opinions and considering the subject matter of the claims of the '228 patent, I am relying upon certain basic legal principles that have been explained to me. I have been informed by counsel that a patent claim is invalid as anticipated under 35 U.S.C. § 102 if each element of that claim is present either expressly or inherently in a single prior art reference. I have also been informed that, to be an inherent disclosure, the prior art reference must necessarily disclose the limitation, and the fact that the reference might possibly practice or contain a claimed limitation is insufficient to establish that the reference inherently teaches the limitation.

55. I have been informed that a claimed invention is unpatentable under 35 U.S.C. § 103, if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. I have also been informed by counsel that the obviousness analysis takes into account factual inquiries including the level of ordinary skill in the art, the scope and content of

the prior art, and the differences between the prior art and the claimed subject matter.

56.I have been informed by counsel that the Supreme Court has recognized several rationales for combining references or modifying a reference to show obviousness of claimed subject matter. Some of these rationales include the following: A) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f) some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

57.Finally, I understand that secondary factors may be considered in an obviousness inquiry. I understand that these are also referred to as secondary considerations and may include evidence of long-felt need,

failure of others, skepticism of those of skill in the art, licensing, commercial success, recognition in the industry, acquiescence, evidence of copying, and unexpected results. While these secondary considerations must be taken into account, they do not necessarily control the obviousness determination.

58. Moreover, for objective evidence of nonobviousness to be accorded substantial weight, I understand that the proponent of such evidence must establish a “nexus” between the evidence and the merits of the patent at issue. Specifically, I understand that the secondary consideration must be tied to the merits of the claimed invention of the patent at issue. Where the alleged secondary consideration results from something other than what is both claimed and novel, I understand that there is no nexus to the merits of the claimed invention, and the secondary considerations are thus not indicative of nonobviousness.

59. With regards to the Phillips-based claim construction, I have been advised that the following legal principles were taken from the Phillips decision:

the words of a claim "are generally given their ordinary
and customary meaning." ... We have made clear,

moreover, that 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, i.e., as of the effective filing date of the patent application.

Importantly, the person of ordinary skill in the art is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification. ...

In some cases, the ordinary meaning of claim language as understood by a person of skill in the art may be readily apparent even to lay judges, and claim construction in such cases involves little more than the application of the widely accepted meaning of commonly understood words. ... In such circumstances, general purpose dictionaries may be helpful. In many cases that give rise to litigation, however, determining the ordinary

and customary meaning of the claim requires examination of terms that have a particular meaning in a field of art. Because the meaning of a claim term as understood by persons of skill in the art is often not immediately apparent, and because patentees frequently use terms idiosyncratically, the court looks to "those sources available to the public that show what a person of skill in the art would have understood disputed claim language to mean." ... Those sources include "the words of the claims themselves, the remainder of the specification, the prosecution history, and extrinsic evidence concerning relevant scientific principles, the meaning of technical terms, and the state of the art." ...

Quite apart from the written description and the prosecution history, the claims themselves provide substantial guidance as to the meaning of particular claim terms.

60. Finally, I am also guided by the following instruction which I understand to also be from the Phillips decision: “Importantly, the person of ordinary skill in the art is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification.”

IX. SCOPE AND CONTENT OF THE PRIOR ART

61. My analysis below of the invalidity includes citation to the specific references mentioned above: Agarwal, Sinharoy, Blumrich, and Sistla.

62. However, I note that the art of caches, including caches that managed cache coherency for both CPU’s and I/O components was well developed prior to the December 2011 filing date of the ‘228 patent.

A. Agarwal

63. Agarwal is a United States patent that predates the filing of the ‘228 patent by over 4 years.

64. *Agarwal* teaches the same solution for I/O cache coherency as taught in the ‘228 patent: a distributed caching agent in which individual portions of the caching agent collectively handle the different portions of a shared cache to form a collective single caching agent that handles both CPU and IO module cache coherency for multicore architectures with an

integrated IO module. Thus, *Agarwal* teaches the same solution as is taught and claimed in the '228 patent – except that *Agarwal* predates the '228 patent by 4+ years.

65. Because *Agarwal* is in the same field of endeavor as the '228 patent (e.g. cache management, including cache coherency architecture, for multicore processors including an integrated I/O module) and is reasonably pertinent to the problem the '228 patent addresses (e.g. cache coherency using a caching agent/ directory controller (i.e. coherence controller)), *Agarwal* is analogous art to the '228 patent.

66. Like the '228 patent, *Agarwal* discloses “[a] multicore processor” with “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” *Agarwal*, Abstract. *Agarwal* teaches:

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.

Directory based protocols are one way of maintaining cache coherence. In many previous systems, a directory is maintained alongside main memory. For example, the directory state and directory controller are both implemented as part of main memory, which is off-chip, and there is directory state associated with each memory line....

67.Agarwal, 1:34-53.

68.Agarwal, teaches that the directory controller and directory storage can be located off chip or it can also be distributed across “tiles” (or “nodes”) in a tiled multicore processor.

The directory can be located at a variety of places. In the examples described below, two of the places at which the directory for a given portion of memory is located are:

(1) A directory can be located at an on-chip memory interface or “shim” adjacent to an on-chip memory controller. This scheme is called shim cache coherence or the SCC scheme. In SCC, there can also be a higher level cache (e.g., L3 cache) which fronts the portion

of main memory controlled by that shim. This cache could also be centralized and front all of main memory.

(2) A directory can be located on a node (e.g., a core or tile) in the multicore system. The directory can be associated with that node's higher level cache or the directory can be standalone by itself. This second scheme is called proxy cache coherence or PCC.

69. *Agarwal*, 11:12-26.

70. Agarwal's Proxy Cache Coherence (PCC) embodiment uses the distributed cache coherence method:

2 Proxy Cache Coherence (PCC)

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., the 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.

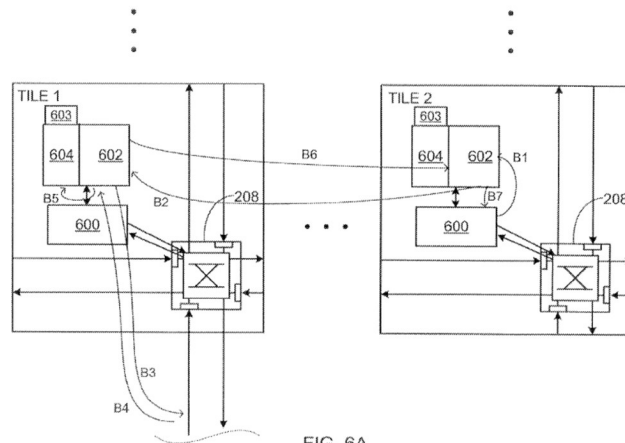
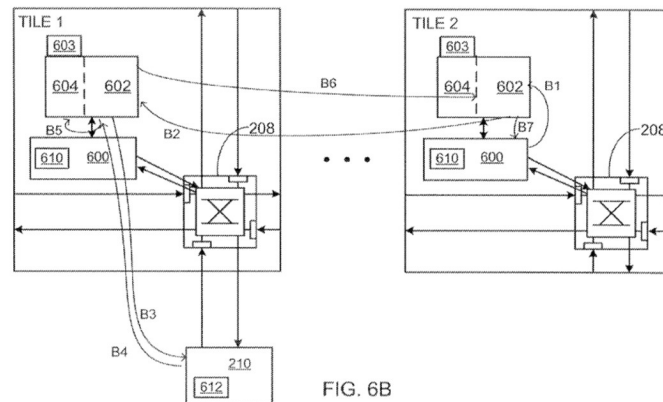


FIG. 6A



71. Agarwal, 16:26-45, FIG. 6A and FIG. 6B.

72. The directory controller(s) and directory storages, collectively, maintain cache coherence. For example, with reference to Figs. 6A and 6B, Agarwal teaches:

An example of the protocol handling for PCC is depicted in FIGS. 6A and 6B and is discussed next. Referring to FIGS. 6A and 6B, the steps for maintaining cache coherence in this example are as follows.

On tile 2, a processor 600 attempts to read data at address *a* from its cache 602 (Action B1). The cache 602 may be serving as an L2 cache after a miss in an L1 cache within the processor 600. After a cache miss (e.g., for a shared data read) on tile 2, the miss goes through as a message through the network to the home tile, tile 1 (Action B2). As described above, the home tile can be indicated by the TLB or some

other method). The message encodes information such as (1) which tile it originated from, (2) what the cache line address is, (3) whether it is a read or a write, (4) whether a local copy of the data exists in the tile's cache already, (5) whether the request is for a shared data item. The message might also encode whether the message corresponds to an acknowledgement that some coherence action has been successfully undertaken by the tile. This action might cause a first network message to go through. (For local data, the home tile might commonly simply be listed as the current tile itself, tile 2. The home tile can also be the same tile for shared data, although it is a rarer than for local data). In this case, on a cache miss, the controller can go directly to main memory, and there is not need to go to a separate home tile.

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.

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).

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.

In the case illustrated in FIG. 6A, the cache line is not present in the cache of tile 1. In this case the tile 1 sends a message to the memory controller responsible for that cache line asking for the line. If there is a common L3 cache in the system near the memory controller shim, then the request can be satisfied by the L3, and if the L3 does not exist, then the request is satisfied by the main memory Action B3).

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

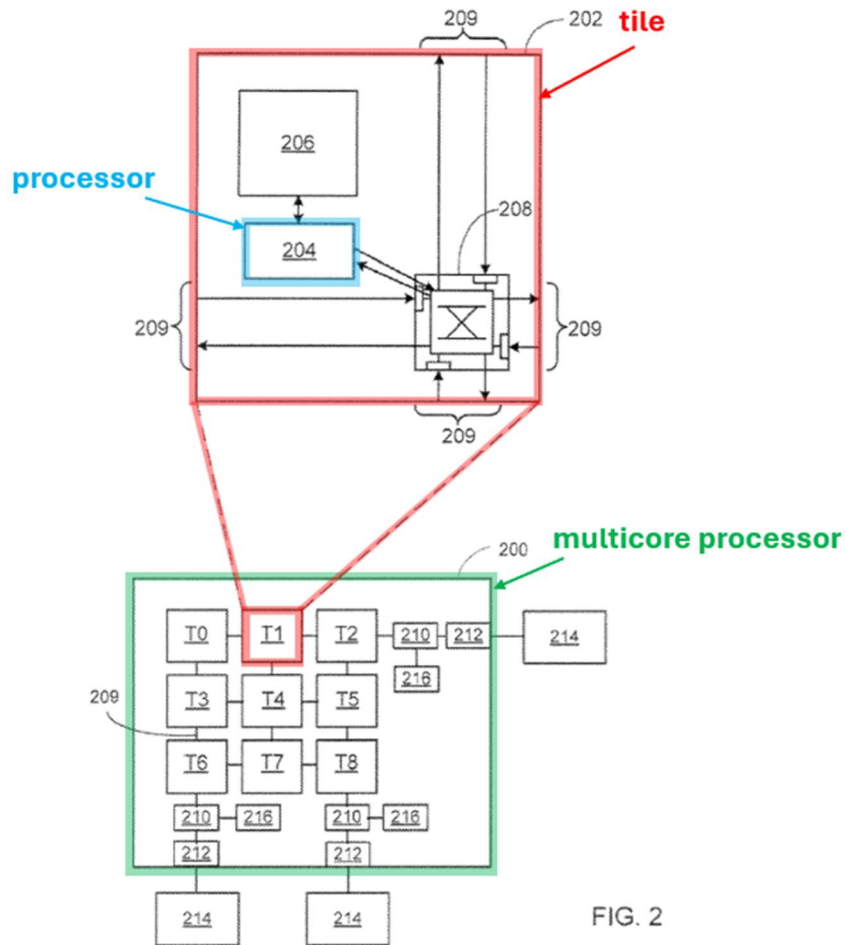
from Tile 1, the cache 602 on tile 1 will serve as the L2 cache for that access.

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.

73.Agarwal, 19:3-20:25.

74.As annotated Figure 2 below illustrates, *Agarwal* discloses a multicore processor 200 that includes, as one example, nine instances (T0-T8) of a tile 202:



75. *Agarwal*, Fig. 2 Annotated).

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.

76. *Agarwal*, 7:34-36. Fig. 2 represents an exemplary multiprocessor architecture.

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. Also shown are data paths 209 of a mesh network connecting the tiles through the switches.

77. Agarwal, 11:53-59.

78. As the annotated figure below further illustrates, each tile includes an L1 cache 610 within the processor 600 and an L2/L3 cache 602 that includes not only private L2 entries but also L3 entries that can be shared by all cores in the tiles:

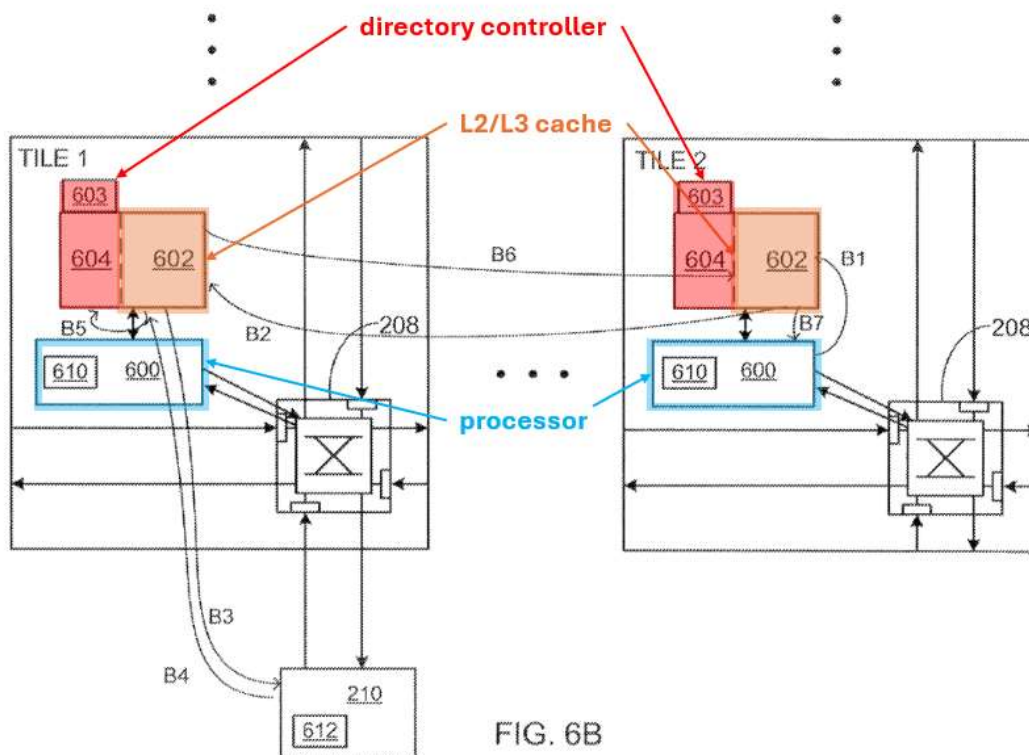


FIG. 6B

79. *Agarwal*, Fig. 6B Annotated).

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 602 on tile 1 will serve as the L2 cache for that access.

80. *Agarwal*, 20:1-16.

81. *Agarwal*'s PCC ("Proxy Cache Coherence") protocol operates to the same as the '228 patent's preferred embodiment in that cache operations including cache coherency are performed using individual portions of the

cache control to form one single caching agent as I have discussed above with respect to the '228 patent and in particular, Figs. 2 and 6 within the '228 patent).

82. Under *Agarwal*'s PCC protocol, a given memory location is assigned to the portion of the shared cache on one of the tiles in the multiprocessor, which is called its "home tile."

2 Proxy Cache Coherence (PCC)

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., the 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.

83. *Agarwal*, 16:26-45.

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.

84. *Agarwal*, 23:46-53.

85. The home tile performs cache coherency operations as a “proxy” for memory locations assigned to the shared cache portion / partition on the home tile, which gives rise to the term Proxy Cache Coherence. See *Agarwal*, 16:26-45 (quoted above)

86. The assignment of a home tile for a cache operation can be done using different approaches including “based on a function of the address such

as a hash function” or a TLB or an address mapping table AMT.

Agarwal, 27:46-28:8, 29:17-30:11, 23:6-53, 27:32-38, 17:42-67, 19:3-29.

87. In *Agarwal*, each tile includes a cache 602, a directory storage to store directory entries and a directory controller (i.e. coherence controller,) coupled to the directory storage.

When a processor (or node) (for example, node 2) requested a cache line which was not present in its cache, node 2's cache would take a cache miss and the request would be sent directly to the main memory location corresponding to the cache line. A coherence controller or the directory controller attached to that memory would check the directory entry stored along with the memory location (in the directory state) and would determine the action to take based on the request and the current state of that memory location. In a common scenario, the controller would simply send the requested data from memory to the node requesting it.

88. *Agarwal*, 2:47-54,

In another aspect, in general, a multicore processor has a directory and coherence controller that are associated with a memory shim and

DRAM controller which are both present on-chip, while the DRAM and directory store are present off-chip.

89.8:38-42,

Directory based protocols are one way of maintaining cache coherence. In many previous systems, a directory is maintained alongside main memory. For example, the directory state and directory controller are both implemented as part of main memory, which is off-chip, and there is directory state associated with each memory line (same size as a cache line, which corresponds to the unit in which memory is cached, also called a cache block).

90.1:46-50,

The directory can be located at a variety of places. In the examples described below, two of the places at which the directory for a given portion of memory is located are:

(1) A directory can be located at an on-chip memory interface or “shim” adjacent to an on-chip memory controller. This scheme is called shim cache coherence or the SCC scheme. In SCC, there can also be a higher level cache (e.g., L3 cache) which fronts the portion

of main memory controlled by that shim. This cache could also be centralized and front all of main memory.

(2) A directory can be located on a node (e.g., a core or tile) in the multicore system. The directory can be associated with that node's higher level cache or the directory can be standalone by itself. This second scheme is called proxy cache coherence or PCC.

91. *Agarwal*, 11:12-26.

In some examples, a directory coherence controller (DCC) 216 is located along with the shim 210 and the memory controller 212 (or DRAM controller) on the chip as illustrated in FIG. 2.

92. *Agarwal*, 12:28-32.

93. These entities manage the cache operations for the partition / slice / portion of shared cache assigned to that tile – just like the individual portions as shown above in Figure 6 of the '228 patent. This is shown, for example, in *Agarwal* at 27:39-28:9 (describing “I/O coherence” as an “extension to the proxy cache coherence”) and describing PCC generally for the distributed processing.

94. *Agarwal* explains that, in an extension of the PCC protocol called “ProxyIO coherence,” I/O coherence (i.e., cache coherence in the

presence of an integrated I/O module) can also be managed by the home tile.

5 I/O Coherence

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 “ProxyIO 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 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 optionally 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.

95.*Agarwal*, 27:39-28:9. As noted above, the ProxyIO coherence is an enhancement to the basic Proxy Cache Coherence.

2 Proxy Cache Coherence (PCC)

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.

96. *Agarwal*, 16:26-45.

97.A POSITA would have understood *Agarwal*'s Figures 2, 6A, 6B, and 9A-C describe the same invention. Figure 2 illustrates an “exemplary multicore processor 200 having a tiled processor architecture.” *Agarwal*, 11:53-54. Figures 6A and 6B use the exemplary architecture to illustrate the “proxy cache coherence (PCC) scheme.” *Agarwal*, 10:25-26. And Figures 9A-9C illustrate the exemplary architecture “with an I/O module” to explain how “an extension of the proxy cache coherence approach” operates to ensure “I/O coherence.” *Agarwal*, 10:31-32, 27:40-46.

B. Sinharoy

98.

99.*Sinharoy* is an IBM paper IBM describing IBM's “POWER7” processor chip. (*Sinharoy*, 1.) In IPR2023-01344, I have not seen any indication that the patent owner challenged the assertion that *Sinharoy* constitutes prior art to the '228 patent family.

100. *Sinharoy* appears to meet the requirements as they have been explained to me to constitute prior art to the '228 patent under at least pre-AIA 35 U.S.C. §102(a). For example, *Sinharoy* itself shows that it was published in the “May/June 2011” edition of the IBM Journal of

Research and Development (ISSN: 0018-8646), a well-known bimonthly serial publication that ran from 1957 to 2020. (Ex-1016, 1; Ex-1014, 1 (LoC Record at 1); Ex-1015, 1 (MARC Record at 1 (field 008, “001120d19572020nyubr pso 0 a0eng c,” indicating a discontinued serial publication (“d”) beginning in “1957” and ending in “2020” (emphasis added))).)

101. Prior to the '228 patent's priority date, at least two documents cite the May/June 2011 edition of the IBM Journal of Research and Development in which *Sinharoy* appears, confirming its public availability and accessibility. (Ex-1013 (cited reference no. [3]); Ex-1012 (cited reference no. [6]).)

102. This indicates that *Sinharoy* was publicly available prior to the '228 patent's priority date.

103. *Sinharoy* is analogous art to the '228 patent. Like the '228 patent, *Sinharoy* discloses a processor chip. (*Sinharoy*, 1.) The image below shows a POWER7 chip:

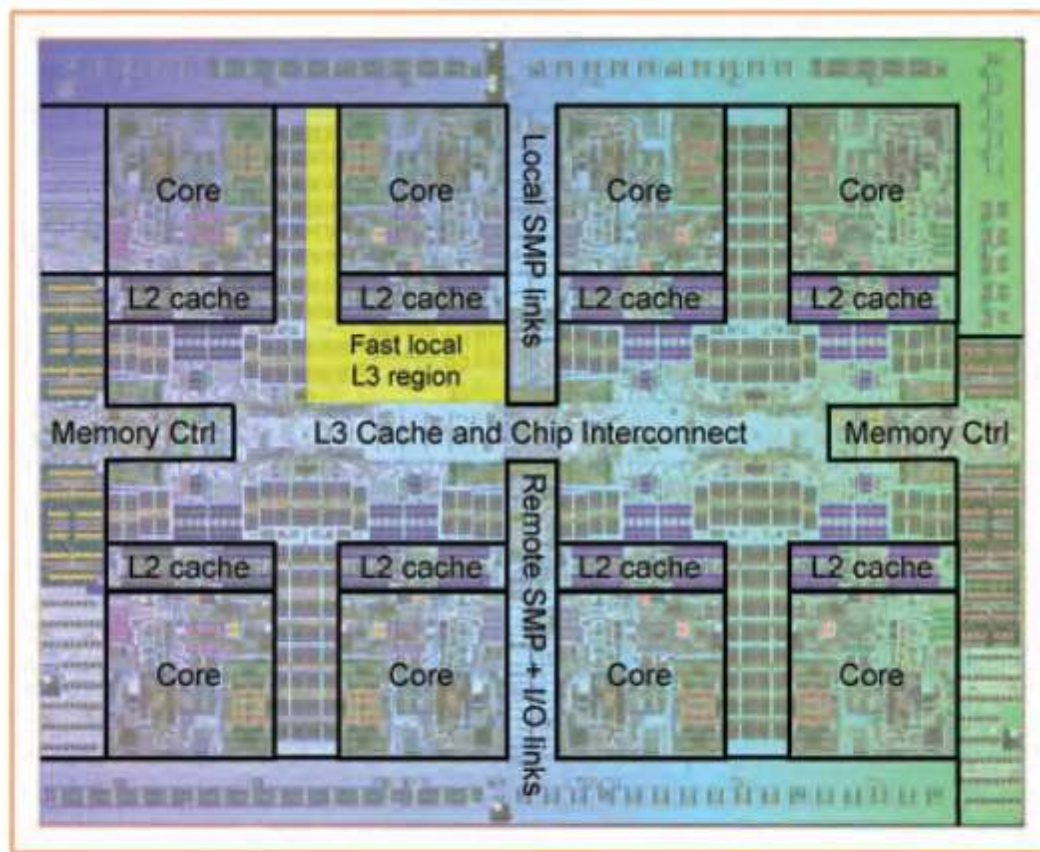


Figure 1

Die photo of the IBM POWER7 chip.

104. (*Sinharoy*, Fig. 1.)

105. *Sinharoy* describes “an eight-core processor” with “three levels of on-chip cache.” (*Sinharoy*, 1.) *Sinharoy*’s on-chip L3 cache “provides several advantages such as improved L3 latency” and “improved L3 bandwidth per core” (*Sinharoy*, 2.) Additionally, *Sinharoy* teaches that “[a]n I/O device performing a direct memory access (DMA) write

operation may target the operation to the cache, instead of to memory.”

(*Sinharoy*, 17.)

106.

C. Blumrich

107. *Blumrich* is analogous art to the '228 patent. Like the '228 patent, *Blumrich* discloses a “multiprocessor system,” (*Blumrich*, Abstract), and focuses on methods to “maintain cache coherence” (*Blumrich*, ¶[0124]).

108. In *Blumrich*, “the L2 caches serve as the point of coherence for all processors.” (*Blumrich*, ¶[0119].) As part of “[c]oherence tracking,” *Blumrich* discloses the use of a “request queue 302,” which “buffers incoming read and write requests.” (*Blumrich*, ¶[0128].) In one embodiment, this queue “is 16 entries deep,” although *Blumrich* notes that “other request buffers might have more or less entries.” (*Id.*) *Blumrich* teaches that, using this queue, “[t]he addresses of incoming requests are matched against all pending requests to determine ordering restrictions.” (*Id.*)

D. Sistla

109. Because *Sistla* is in the same field of endeavor as the '228 patent and is reasonably pertinent to the problem the '228 patent addresses, *Sistla* is

analogous art to the '228 patent.

Like the '228 patent, *Sistla* discloses a “processor” comprising “memory,” “a plurality of processor cores in communication with the cache memory,” and “a scalability agent unit that operates as an interface between an on-die interconnect and both multiple processor cores and memory.” (*Sistla*, Abstract.)

110. *Sistla*'s multicore processor includes “a plurality of core-cache clusters 310₁-310_N in communication with each other over an on-die interconnect 320.” (*Sistla*, ¶¶ [0030]-[0031].) Each core-cache cluster includes a “scalability agent,” including a “scalability agent protocol,” to maintain “memory coherency.” (*Sistla* ¶ [0018].) *Sistla* teaches that “on-die interconnect 320 is configured as a ring interconnect.” (*Sistla* ¶ [0030].).

X. DETAILED INVALIDITY ANALYSIS

111. For this Declaration, I have been asked to provide an opinion as to the Challenged Claims are invalid over the prior art as specified herein.

112. As part of my analysis, I have considered the scope and content of the prior art, described above. Additionally, I considered whether any differences between the alleged invention and the prior art (if any such

exist) are such that the subject matter, as a whole, would have been obvious to a person having ordinary skill in the art at the time of the alleged invention. I have also considered the level of ordinary skill in the pertinent art in performing my analyses. I have also considered the state of the art as of the time of the invention.

A. Structure Of The Challenged Claims And My Analysis Below.

113. The '228 patent has 3 independent claims each with one or more dependent claims.

114. As noted in the analysis below, the dependent claims overlap with each other. Also, as noted in the analysis below, some of the elements of certain independent claims overlap with elements in other independent claims and/or dependent claims.

115. It is my opinion that the prior art discloses all Challenged Claims or renders all Challenged Claims obvious.

B. Ground 1: Claims 1-2, 4-7, and 11-14 Are Obvious Over Agarwal.

116. As detailed below, it is my opinion that the claims 1-2, 4-7, and 11-14 are obvious over Agarwal. As detailed below, Agarwal discloses¹, or renders obvious, every element of claims 1-2, 4-7, ad 11-14.

1. Overview Of Agarwal

117. I have provided a detailed discussion above for Agarwal in Section IX.A

118. Below I perform an element by element comparison for Agarwal.

2. Agarwal Discloses The Elements Of Claims 1-2, 4-7, And 11-14.

a) Claim 1

i. 1[pre]: “An apparatus comprising:”

119. I have been advised that preamble can be limiting in certain circumstance but I do not need to make any opinions herein whether this preamble is limiting because *Agarwal* plainly discloses, or renders obvious the claimed apparatus as detailed in the elements below.

¹ As used herein, “discloses” encompasses what a reference would teach, suggest, or render obvious to a POSITA.

120. *Agarwal* discloses “a multicore processor,” which is an apparatus as detailed below. *Agarwal*, Abstract, 1:32-33 (“Cache coherence is a useful mechanism in multiple processor systems to provide a shared memory abstraction to the programmer.”).

In one aspect, in general, a multicore processor comprises a plurality of cache memories, and a plurality of processor cores, each associated with one of the cache memories. Each of at least some of the cache memories is configured to maintain at least a portion of the cache memory in which each cache line is dynamically managed as either local to the associated processor core or shared among multiple processor cores.

121. *Agarwal*, 3:15-23. *Agarwal*, Fig. 2 (shown in detail below for element 1[a]).

- ii. 1[a]: “a multicore processor including a plurality of cores”

122. *Agarwal* discloses, or renders obvious, this element.

123. As Figure 2 below shows, *Agarwal* discloses a multicore processor 200 with nine instances (T0-T8) of a tile 202:

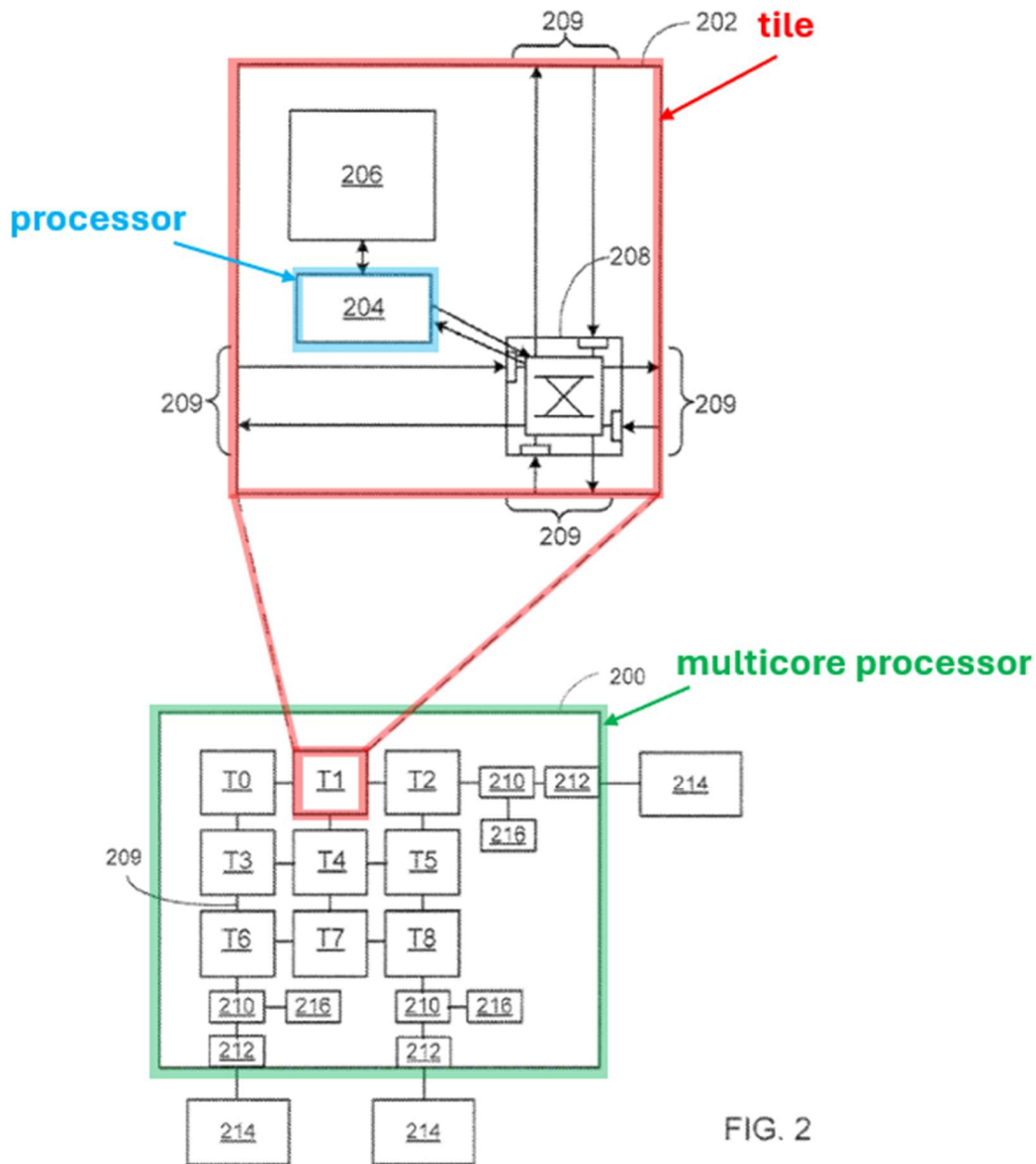


FIG. 2

124. *Agarwal*, Fig. 2 Annotated). See *Agarwal* 3:15-24 (quoted in preamble above);

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.

125. *Agarwal*, 7:34-40;

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.

126. *Agarwal*, 11:53-59.

127. Each tile 202 includes a processor 204 that corresponds to one of the claimed “plurality of cores.” *Agarwal*, 1:32-33 (“This invention relates to caching in ***multicore and multiprocessor computers***”), also, *Agarwal*, 3:6-14 (“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.”)

128. *Agarwal*’s Figure 6B further shows multicore processor 200 can have multiple tiles, each of which includes a processor 600 (i.e., “plurality of cores”):

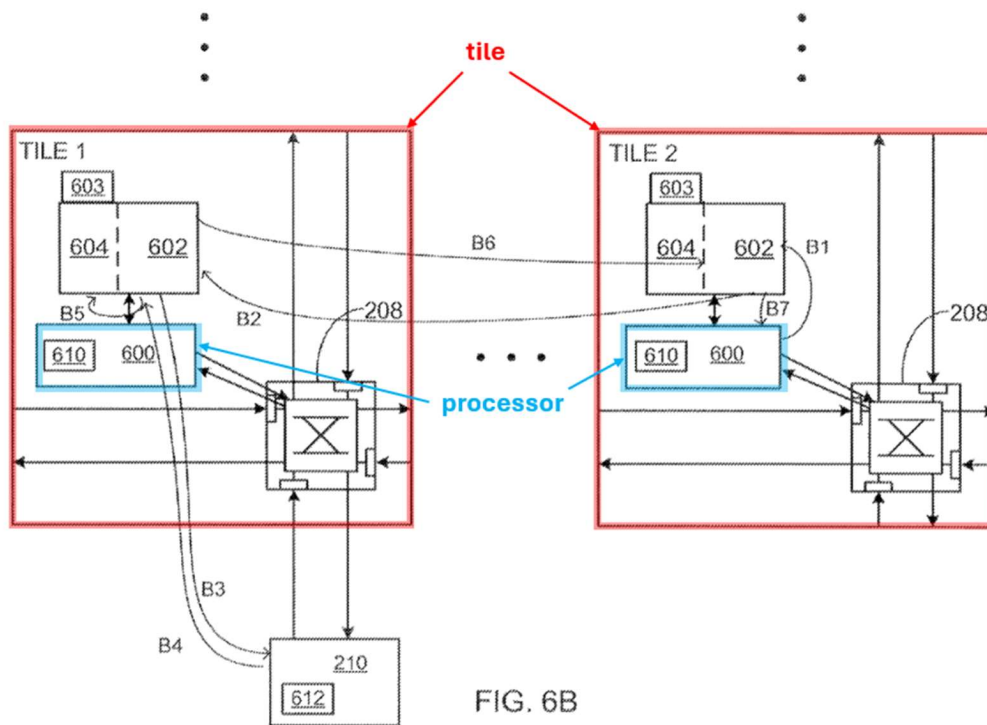


FIG. 6B

129. *Agarwal*, Fig. 6B Annotated), 20:1-4 (“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.”).

130. Thus, *Agarwal* discloses, or renders obvious, this element.

iii. 1[b]: “a shared cache memory”

131. *Agarwal* discloses, or renders obvious, this element. *Agarwal* discloses “a shared distributed cache.” *Agarwal*, 23:47-48.

132. *Agarwal* explains that the L2/L3 cache can be shared by multiple processor cores.

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.

133. *Agarwal*, 10:1-10,

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***.

134. *Agarwal*, 11:35-41,

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.

135. *Agarwal*, 18:39-47,

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...

136. *Agarwal*, 20:12-16,

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 function) to access *the data from the shared distributed cache* (which may in some cases be on the same tile).

137. *Agarwal*, 23:45-63.

138. For example, Figure 6B Annotated below) shows that each tile includes an L2/L3 cache 602 whose L3 entries can be shared by all cores across the various tiles (i.e., “a shared cache memory”) for a shared distributed cache:

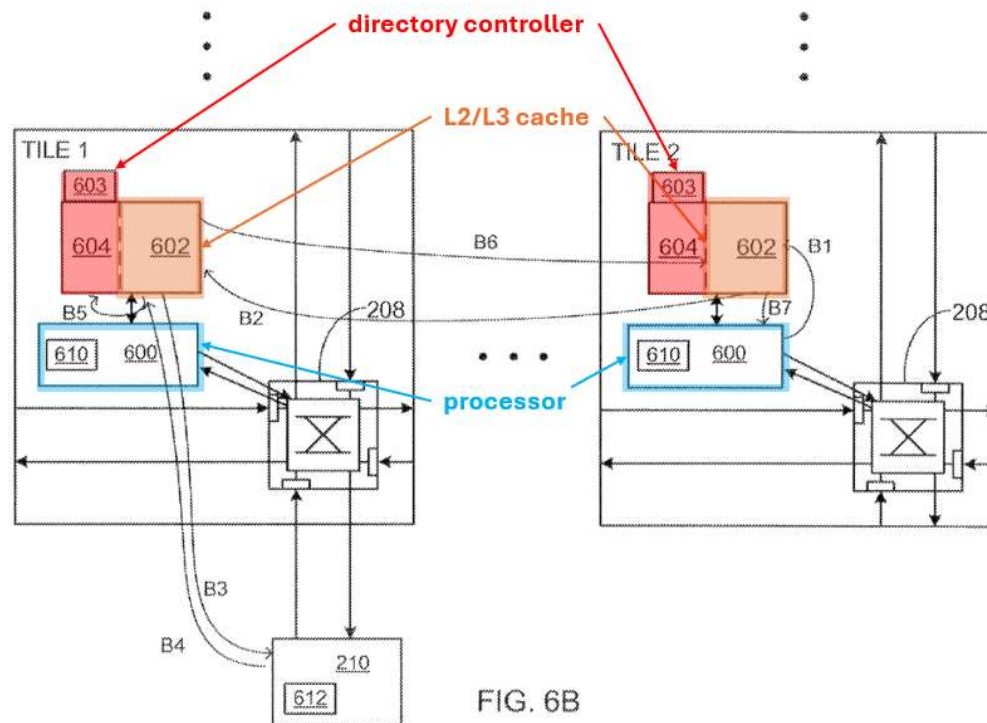


FIG. 6B

139.

140. *Agarwal*, Fig. 6B Annotated).

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 602 on tile 1 will serve as the L2 cache for that access

141. *Agarwal*, 20:1-16, 23:45-51 (quoted above).

142. *Agarwal* thus discloses a shared distributed cache memory, as claimed.

- iv. 1[c]: “an integrated input/output (IIO) module to interface between the multicore processor and at least one IO device coupled to the multicore processor”

143. *Agarwal* discloses, or renders obvious, this element.

144. *Agarwal*’s multicore processor ([1a]) includes an “I/O module” which is an integrated input/output (IIO) module as claimed. E.g., *Agarwal*, 10:31-32 (“FIGS. 9A-9C are diagrams of a multicore chip with an I/O module.”); *see also id.* at Figs. 9A-9C.

145. According to *Agarwal*, this IIO module can access memory.

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.

146. *Agarwal*, 27:40-45 (“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***.”).

147. *Agarwal* explains that the multicore processor must therefore be able to maintain cache coherence in the presence of the 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 “ProxyIO coherence.”***

148. *Agarwal*, 27:42-46.

149. *Agarwal*’s Figures 9A-9C illustrate cache coherence operations of the multicore processor ([1a]) with an I/O module.

150. As Figure 9B shows, multicore processor 900 includes “I/O module 912” that is integrated onto the same multicore processor 900 as the plurality of processor cores.

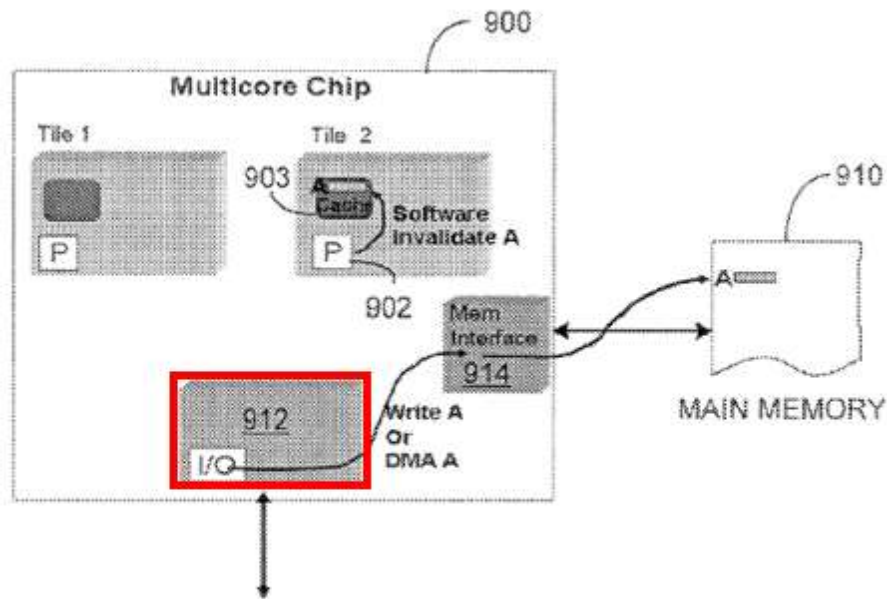


FIG. 9B

151. Agarwal, Fig. 9B Annotated),

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.”

152. *Agarwal*, 28:11-22,

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.

153. *Agarwal*, 29:5-7.

154. Figure 9B above also shows L2/L3 cache 903 in each tile (equivalent to cache 602 in Fig. 6B).

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.

155. *Agarwal*, 28:44-47.

156. I/O module 912 interfaces between the multicore processor and an IO device coupled to the multicore processor. For example, *Agarwal's* 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).”

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).

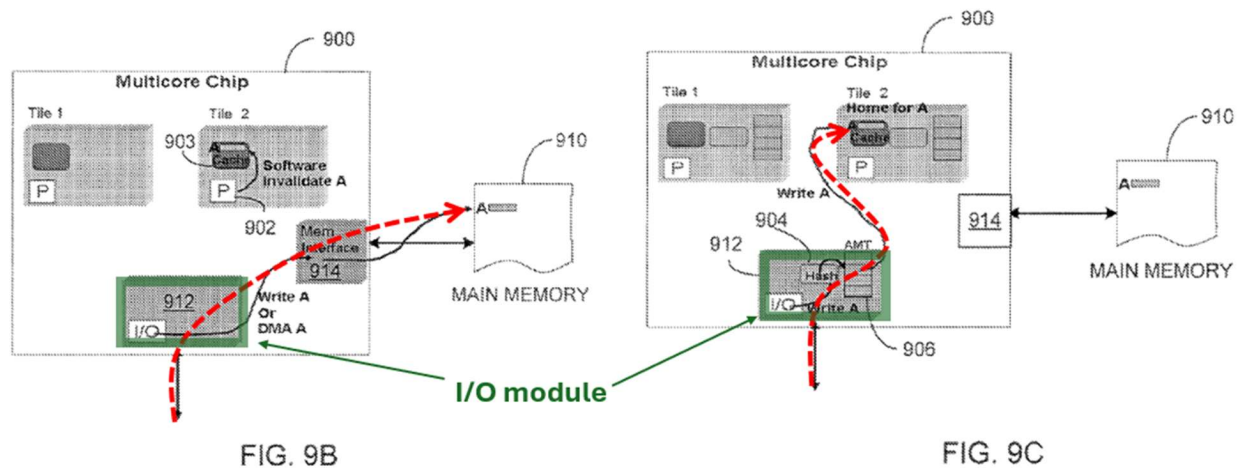
In this situation, if some tile has the same data location A) cached in its cache, a coherence protocol ensures that these caches are managed appropriately (e.g., invalidated or updated). This situation is shown in FIG. 9B. This situation is case 1: the “memory coherence case.”

157. *Agarwal*, 28:11-22.

158. *Agarwal* explains that I/O module 912 supports transactions that send data from an off-chip IO device (1) directly to “the main memory” (i.e., direct memory access (DMA) transactions described in Fig. 9B and 28:11-22 (quoted above), 28:33-43 (“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*) or (2) *directly into the L2/L3 cache (i.e., direct cache access (DCA) transactions* described in Fig. 9C and 28:23-32 (“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.”), 29:5-7 (quoted above).

159. The annotated figures below show both transactions and illustrate how I/O module 912 interfaces between multicore processor 900 and an IO device (not shown) coupled to the multicore processor:



160. *Agarwal*, Fig. 9B (left, annotated showing memory coherence case),
Fig. 9C (right, annotated showing the cache coherence case).

161. In the cache coherence case, the I/O module may maintain the necessary information (such as an AMT or hash function) to specify which tile is the home tile for an access such that it can directly send any cache accesses to that tile for cache coherence. *Agarwal*, 30:5-11, 27:40-28:9.

162. Thus, *Agarwal* discloses this element.

- v. 1[d]: “a caching agent to perform cache coherency operations for the plurality of cores and the IIO module”

163. *Agarwal* discloses, or renders obvious, this element.

164. *Agarwal* 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.

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.

165. *Agarwal*, 1:34-39.

166. To address this, *Agarwal* teaches several cache coherence management techniques, including Shim Cache Coherence (SCC) and ***Proxy Cache Coherence (PCC) protocols***. *Agarwal*, 16:26-20:34, 27:39-30:4.

167. Both SCC and PCC are directory based protocols used to maintain cache coherence by employing a directory controller (i.e. coherence controller).

Directory based protocols are one way of maintaining cache coherence. In many previous systems, a directory is maintained alongside main memory. For example, the directory state and

directory controller are both implemented as part of main memory, which is off-chip, and there is directory state associated with each memory line (same size as a cache line, which corresponds to the unit in which memory is cached, also called a cache block).

168. *Agarwal*, 1:46-50.

Directory based cache coherence approaches are generally more scalable and hence may be preferable over other approaches, such as snoopy cache approaches, for large-scale architectures.

169. *Agarwal*, 2:21-24.

When a processor (or node) (for example, node 2) requested a cache line which was not present in its cache, node 2's cache would take a cache miss and the request would be sent directly to the main memory location corresponding to the cache line. ***A coherence controller or the directory controller*** attached to that memory would check the directory entry stored along with the memory location (in the directory state) and would determine the action to take based on the request and the current state of that memory location. In a common scenario, the controller would simply send the requested data from memory to the node requesting it.

170. *Agarwal*, 2:47-54; see also *Agarwal*, 8:38-42 (quoted above).

171. SCC and PCC can be implemented in different locations.

The *directory can be located at a variety of places*. In the examples described below, two of the places at which the directory for a given portion of memory is located are:

(1) A directory can be located at an on-chip memory interface or “shim” adjacent to an on-chip memory controller. This scheme is called shim cache coherence or the SCC scheme. In SCC, there can also be a higher level cache (e.g., L3 cache) which fronts the portion of main memory controlled by that shim. This cache could also be centralized and front all of main memory.

(2) *A directory can be located on a node (e.g., a core or tile) in the multicore system*. The directory can be associated with that node's higher level cache or the directory can be standalone by itself. This second scheme is called *proxy cache coherence or PCC*.

172. *Agarwal*, 11:12-26. See also *Agarwal*, 12:28-32 (quoted above).

173. Both SCC and PCC also employ directory storage that collectively with the directory controller operate as the claimed “caching agent” to

perform cache coherency operations for the plurality of cores and the IIO module.

174. The two approaches differ as to where those components are located.

Under the SCC protocol, a directory controller and directory storage are located adjacent to an on-chip memory controller. *Agarwal*, 11:15-21.

175. The PCC protocol is particularly relevant herein because under *Agarwal*'s PCC protocol, the directory controller and directory storage are distributed across the tiles in the multicore processor.² *Agarwal*, 11:22-26 (quoted above).

176. A particular cache operation is managed by one of the tiles in the multiprocessor, which would be the "home tile" for that cache memory operation.

2 Proxy Cache Coherence (PCC)

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*

² The directory storage 604 "can be included within the same memory that provides the cache 602, or can be included in a separate memory module on the tile." *Agarwal*, 16:39-42. The "directory controller 603 is coupled to the directory storage 604." *Agarwal*, 16:44-45.

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., the 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*.

177. *Agarwal*, 16:26-45.

For the shared distributed cache, each memory line has an assigned home location in the cache 206 of one of the tiles.

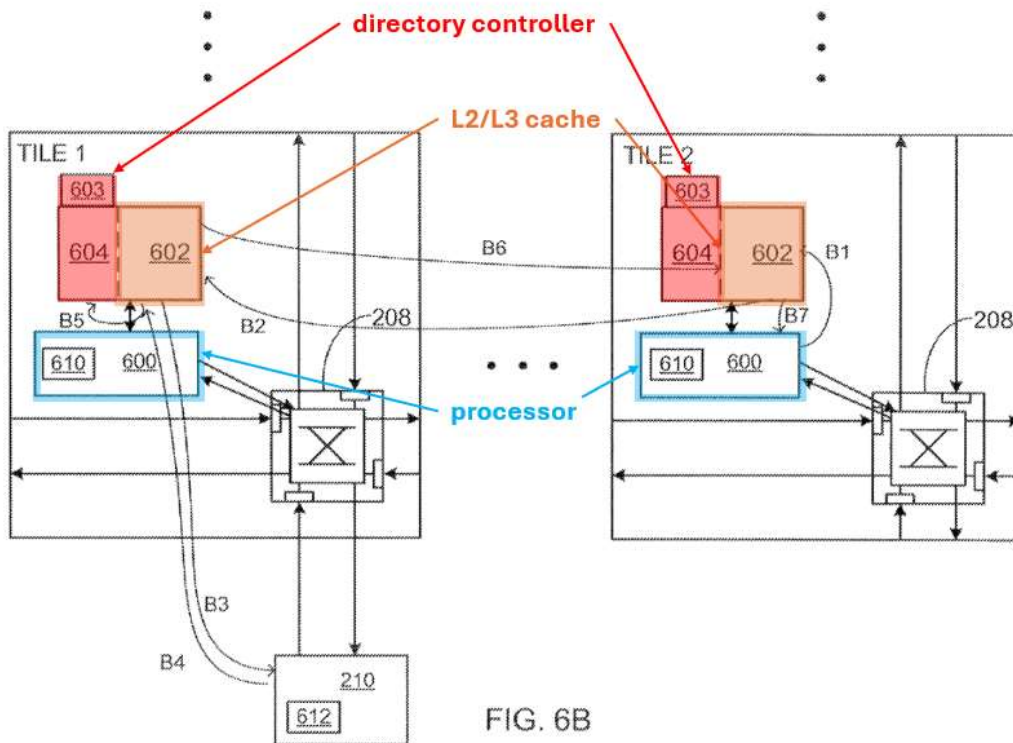
178. *Agarwal*, 23:51-53.

179. The home tile, including its directory controller and directory storage, performs cache coherency operations as a “proxy” for memory locations assigned to the tile, giving rise to the term Proxy Cache Coherence (PCC). *Agarwal*, 16:26-45 (quoted above).
180. *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,” (‘228 patent, 2:66-3:3)).
181. In *Agarwal*, 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. *Agarwal*, 27:39-28:8, 29:3-30:12.
182. For example, *Agarwal* discloses that the SCC and PCC schemes work in a similar manner with one difference is the directory controller (*i.e.* coherency controller) and directory storage being centralized in SCC (*e.g.* DCC 216 with directory adjacent to an on-chip memory controller shim, *Agarwal*, 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, *Agarwal*, 11:22-26, FIG. 6A, 16:36-45). See *Agarwal*, 16:27-34 (quoted above), 11:12-26 (quoted above).

183. *Agarwal*'s distributed caching portions also manage access to the cache by IO components – in the same manner as taught/claimed in the '228 patent. *Agarwal* explains that, in an extension of the PCC protocol called "ProxyIO coherence," the home tile can also manage cache coherence in the presence of an I/O module ("I/O coherence"). *Agarwal*, 27:39-28:9; see also *Agarwal*., 16:26-45. This is the same proxy I/O coherence concept described 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, 6:5-7, 6:35-37.

184. *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:



185. *Agarwal*, Fig. 6B Annotated), 20:1-25.

Each directory controller 603 and directory storage 604 manages cache coherence for the memory locations assigned to that home tile regardless of whether the cache access is from a tile (CPU) or an I/O module. *Agarwal*, 29:23-29 (“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.”) See also *Agrawal* 27:39-28:9 (previously quoted).

186. 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 tile that is the “home tile” for the shared cache partition that includes that cache line (the “home tile”) manage the coherence. All the directory controllers 603 and directory storages 604, collectively for all tiles (managing the collective memory partitions), thus manage cache coherence for the entire shared cache and memory space). Thus, Agarwal’s distributed components teach, or render obvious, a “single” caching agent managing cache coherence for the plurality of processor cores and the IIO module in the same manner as the ‘228 patent teaches for distributed caching agent partitions managing memory partitions / slices. ‘228 patent, Fig. 2, 2:66-3:3, Fig. 6, 6:20-29, 6:33-35.

187. *Coherence operations for the “plurality of cores”*: Figure 9A describes *Agarwal*’s cache coherence operations for the plurality of cores:

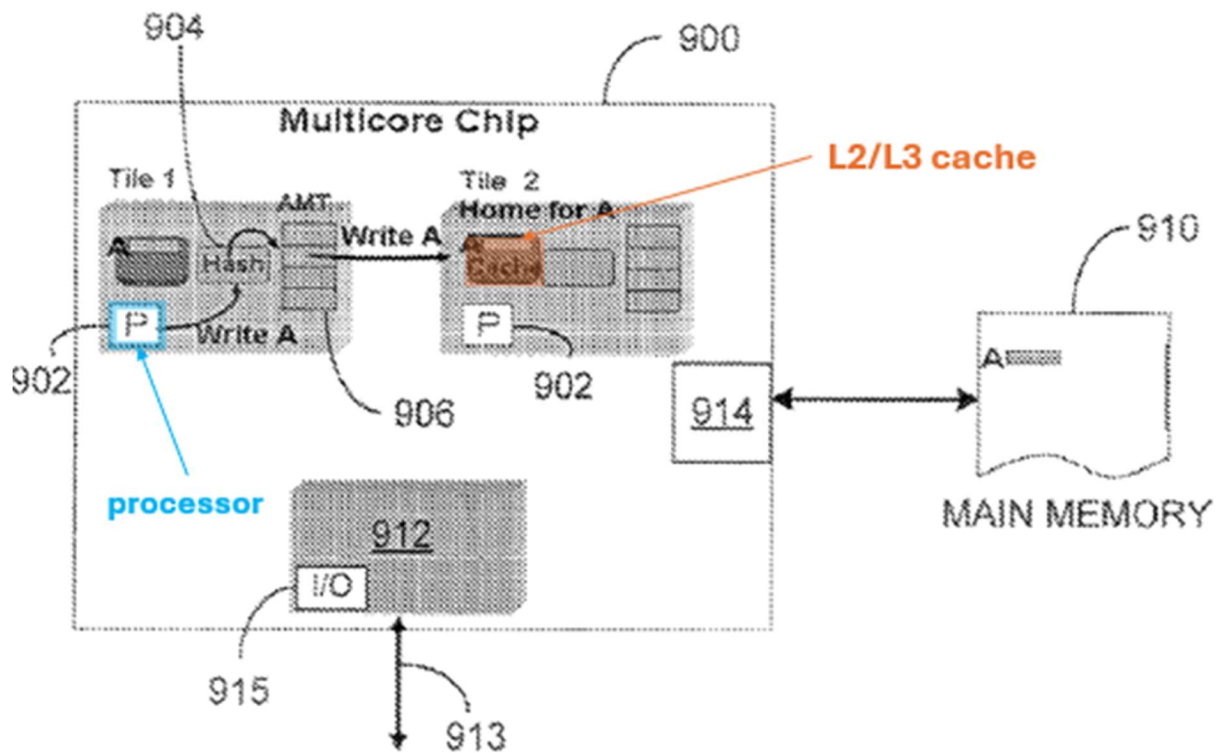


FIG. 9A

188. *Agarwal*, Fig. 9A Annotated). When Tile 1's processor 902 writes data to memory location having address A, Tile 1 sends the write request to the home tile of address A (Tile 2) based on a hash function and an address mapping table AMT) or TLB. *Agarwal*, 27:46-28:8, 29:17-30:11, 23:6-53, 27:32-38, 17:42-67, 19:3-29.

189. The home tile (Tile 2) performs cache coherence operations as a proxy and writes the data into a shared cache line located in the L2/L3

cache in Tile 2. *Agarwal*, 27:46-28:8; *see also Agarwal*, 20:1-16, Fig. 6B.

(shown and quoted above).

190. ***Coherence operations for the IIO***: Figure 9C describes *Agarwal*'s

ProxyIO cache coherence operations for the integrated I/O module:

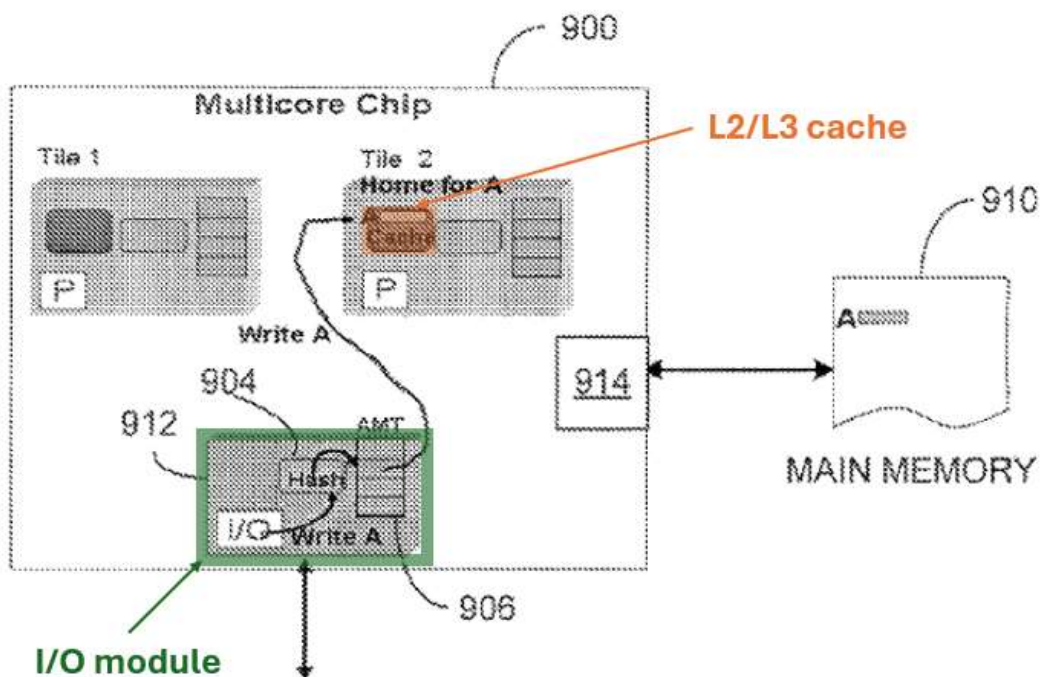


FIG. 9C

191. *Agarwal*, Fig. 9C Annotated), 29:5-30:8; 27:39-28:9. *Agarwal*

explains that I/O module 912 includes circuitry implementing a hash function and address mapping table AMT).

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.

192. *Agarwal*, 29:17-28. For example, I/O module 912 can deliver a write transaction to the home tile of the address A (Tile 2) based on a hash function and an address mapping table as quoted above in 29:17-28. Once the home tile (Tile 2) receives the write transaction, it performs cache coherence operations as a proxy to I/O module 912 and writes the data into the shared portion of Tile 2's L2/L3 cache. *Agarwal*, 29:5-28.

193. Thus, in the same manner as the '228 patent teaches, *Agarwal* teaches distributed caching agent portions each managing the partition / portion /

slice of shared cache assigned to that portion (e.g., the “home tile” for the shared distributed cache portion in *Agarwal*).

194.

- vi. 1[e]: “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.”

195. *Agarwal* discloses this element. As detailed in [1d], *Agarwal*’s caching agent has implements a shared distributed caching agent with portions associated with each of the cores.

196. *Agarwal* operates like the ‘228 patent to form a single caching agent with distributed portions. Like the ‘228 patent, *Agarwal*’s individual caching agent portions each handle cache operations for a portion / partition of the overall shared distributed cache. Thus, collectively, the portions form a single caching in the same manner as the ‘228 patent taught Albeit, 4+ years after *Agarwal*).

197. As discussed for [1d], *Agarwal* discloses a PCC protocol. (*Supra* [1d]; *Agarwal*, 16:26-20:34, 27:39-30:4). Under this PCC 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 – 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 distributed caching agent portions. *Supra* [1d]. Each tile has a directory controller 603 and directory storage 604 coupled to a corresponding processor core.

Referring to FIG. 6A, *each of the tiles (e.g., the 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.*

198. *Agarwal*, 16:35-45, Fig. 6A.

An example of the protocol handling for PCC is depicted in FIGS. 6A and 6B and is discussed next. Referring to FIGS. 6A and 6B, the steps for maintaining cache coherence in this example are as follows.

On tile 2, a processor 600 attempts to read data at address a from its cache 602 Action B1). The cache 602 may be serving as an L2 cache after a miss in an L1 cache within the processor 600. After a cache miss (e.g., for a shared data read) on tile 2, the miss goes through as a message through the network to the home tile, tile 1 Action B2). As described above, the home tile can be indicated by the TLB or some other method). The message encodes information such as (1) which tile it originated from, (2) what the cache line address is, (3) whether it is a read or a write, (4) whether a local copy of the data exists in the tile's cache already, (5) whether the request is for a shared data item. The message might also encode whether the message corresponds to an acknowledgement that some coherence action has been successfully undertaken by the tile. This action might cause a first network message to go through. (For local data, the home tile might commonly simply be listed as the current tile itself, tile 2. The home tile can also be the same tile for shared data, although it is a rarer than

for local data). In this case, on a cache miss, *the controller can go directly to main memory, and there is not need to go to a separate home tile.*

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.

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).*

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.

In the case illustrated in FIG. 6A, the cache line is not present in the cache of tile 1. In this case the tile 1 sends a message to the memory controller responsible for that cache line asking for the line. If there is a common L3 cache in the system near the memory controller shim, then the request can be satisfied by the L3, and if the L3 does not exist, then the request is satisfied by the main memory Action B3).

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 602 on tile 1 will serve as the L2 cache for that access.

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.

199. *Agarwal*, 19:30-20:25; *See also Agarwal*, Fig. 6B.

200. *Agarwal's* multiple directory controllers and directory storage blocks in the multicore processor, collectively, perform cache coherence operations for the shared cache memory through the tiles' distributed

portions managing cache access for different portions / partitions / slices of the shared cache. Thus, Agarwal discloses a plurality of distributed portions of a “single caching agent” for the whole multicore processor.

201. 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, 2:67-3:3. Thus, the claimed correspondence is with the *core*, not necessary the bank or slice.

202. *Agarwal* teaches this association explicitly as noted above for Figs 6A-6B. For example, Agarwal teaches “[a]t least some of the directory ***controllers are associated with a processor core....***” Agarwal, 3:42-44, claim 9. Similarly, Agarwal 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.***” Agarwal, 4:7-8, 6:35-36, claims 14, 63. 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”).

Supra [1d].

203.

- b) Claim 2: “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.”

204. *Agarwal* discloses this claim.

205. The ’228 patent specification defines a table of requests (TOR) as “As seen, **TOR 220 may be a buffer or other temporary storage** for holding incoming requests. In the embodiment shown, **TOR 220 may include at least one dedicated resource 225 for posted transactions**. In one embodiment, **dedicated resource 225 may be a single entry of the queue**, although the scope of the present invention is not limited in this regard.”” ’228 patent, 2:46-53.

206. *Agarwal* explains each “tile can . . . **keep a small buffer that contains a list of outstanding transactions** with an acknowledgement counter associated with each [outstanding transaction].” *Agarwal*, 27:21-25;

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*.

Each buffer entry can maintain a count of outstanding transactions.

207. *Agarwal*, 16:1-5, *See also Agarwal*, 20:53-54 (“The PCC protocol can also use *all of the mechanisms described in the SCC protocol*.”).

208. *Agarwal* identifies this “small buffer” as being part of “*cache state that is used to store directly state*” *Agarwal*, 27:11-25. Moreover, in *Agarwal*, “*each directory controller ... is configured to maintain directory state....*” *Agarwal*, 3:39-41, 4:9-12, claim 15, claim 8. *Agarwal* teaches a “*directory storage 604 for directory state*.” *Agarwal*, 16:39-42. Thus, a POSITA would have understood this buffer is part of the directory controller / directory storage (part of the “caching agent”), which maintains coherence for the incoming requests or, it would be obvious to include this buffer as part of the directory controller / directory storage for each tile to allow for fast / efficient access in coherency operations.

209. A POSITA further would have understood that a buffer would beneficially be used to operate *Agarwal*'s directory controller, so it can track outstanding requests and that this buffer is beneficially part of the directory controller / directory access for timely and efficient access in managing coherence for cache access requests. *See Blumrich* ¶[0118]-[0128] (disclosing the use of a request queue for directory based coherence as detailed in the discussion of Blumrich above).

210. Using this buffer, the directory controller stores the list of outstanding transactions (i.e., "pending requests") and associated acknowledgment counters involving the plurality of cores and I/O module discussed above in [1a], [1c], and [1d]. (*Supra* 1a], [1c], and [1d].)

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).

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.

211. *Agarwal*, 27:11-25;

In some protocols, the directory also maintains a counter for each directory entry/cache line to count the number of outstanding acknowledgements it must receive associated with that cache line. The *directory can maintain a counter for all possible directory entries*.

212. *Agarwal*, 5:32-37.

213. Thus, *Agarwal* discloses this claim.

c) Claim 4: “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.”

214. *Agarwal* discloses this claim.

215. A POSITA would have understood that “a last level cache (LLC),” as claimed, refers to the highest-level cache shared by all cores. *E.g.*, ‘228 patent, 2:14-17 (“one or more levels of cache memories can be present within the cores. ... a shared cache memory 130, ... may be a last level cache (LLC)”), 6:30-36 (“In general, each core 710 0-710 n may include low level caches in addition to various execution units and additional processing elements. In turn, the various cores may be coupled to each other and to a shared cache memory formed of a plurality of units of a LLC 740 0-740 n via caching agent 715 0-715 n. In various embodiments, LLC 740 may be shared amongst the cores and IIO module 750.”).

216. As discussed for [1b], *Agarwal* discloses an L2/L3 cache 602 for each tile in the multicore processor with the L3 cache lines being shared among all the tiles (i.e., “a last level cache (LLC)”). *Supra* [1b].

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 602 on tile 1 will serve as the L2 cache for that access.

217. *Agarwal*, 20:1-16, Fig. 6B.

218. Each tile's L2/L3 cache 602 is contained within that tile. E.g., *Agarwal*, 20:12-16 ("*cache 602 can serve as both a local L2 cache for the local tile and as a shared L3 cache for other tiles*,"); 23:45-50 ("*cache 206 ... 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), ...*"); Fig. 2.

219. *Agarwal* repeatedly references the "core associated with [one of the] the cache memory/memories." *Agarwal*, Abstract, 3:63-64, 4:28, 4:42-



221. The shared L3 cache lines of the L2/L3 cache 602 “serve . . . as a *shared L3 cache for other tiles.*” Agarwal, 20:12-16; *see also* Agarwal, 23:45-51 A “shared distributed cache.”).

222. Collectively, the shared L3 cache lines constitute “a plurality of distributed portions each associated with a corresponding one of the plurality of cores” because the shared L3 cache lines are distributed

across all cores (e.g., Tiles 1 and 2 above), each L3 partition being associated with a single core.

d) Claim 5: “5. The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect..

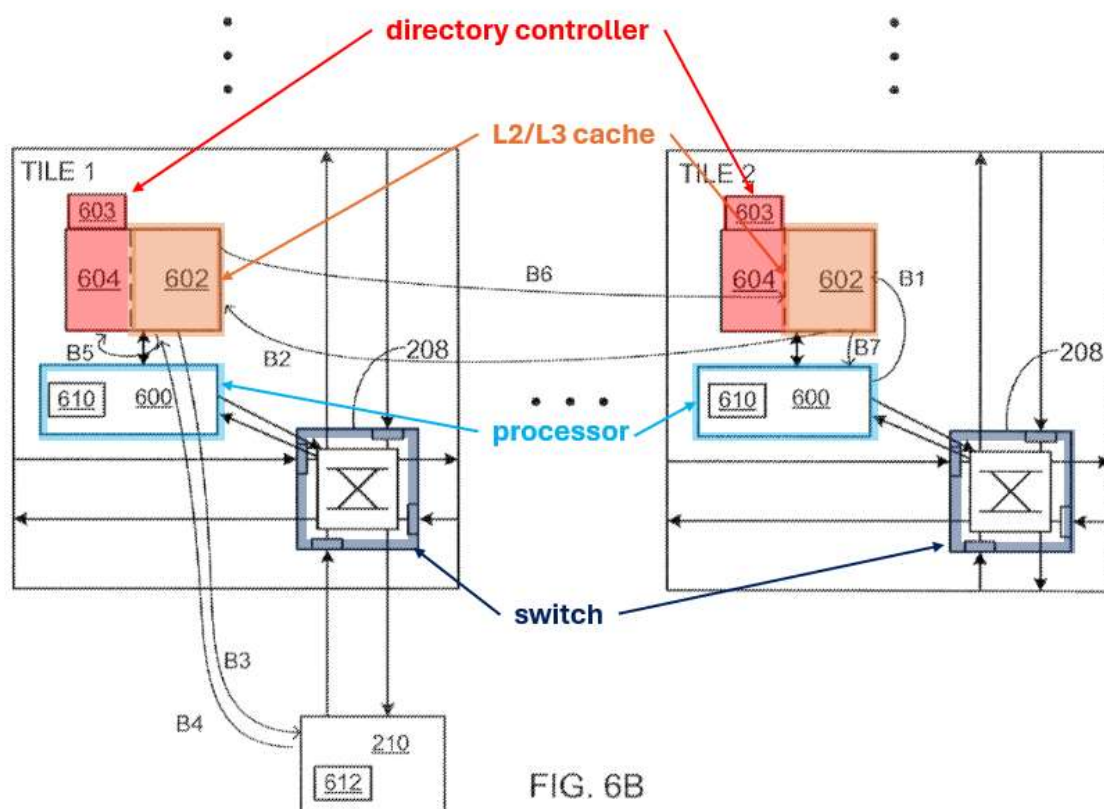
223. *Agarwal* discloses this claim.

224. In one exemplary embodiment, *Agarwal* explains that a mesh network may connect (via data paths) all the tiles—including the processor cores, L2/L3 caches, and directory controllers—through switches 208.

Also shown are ***data paths 209 of a mesh network connecting the tiles through the switches***. In some tiled architectures the switch 208 can include multiple switching modules for different independent networks connecting the tiles, such as static networks (in which routing is controlled by statically programmed switch instructions) or dynamic networks (in which messages are dynamically routed at runtime). 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.

225. *Agarwal*, 11:53-64, Fig. 2.



226. *Agarwal*, Fig. 6B Annotated). *Agarwal* also explains that the tiles (including directory controllers) “are **interconnected over a network that includes ... a ring....**” *Agarwal*, 3:45-47, 7:16-18, 11:66-12:2, claim 10. *Agarwal* specifically explains that cache coherency can similarly be

maintained using such a ring interconnect topology for the multicore processor. *Agarwal*, 20:63-67.

e) Claim 6: “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.”

227. *Agarwal* discloses, or renders obvious, this claim.

228. 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, 3:22-25; *see also id.*, 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”).

229. *Agarwal* discloses this feature. For example, *Agarwal* describes two alternatives of an I/O module writing into memory (the “memory coherence case” or writing into the cache (the “cache coherence case”).

5.1 I/O Coherence Examples

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.”

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.”*

230. *Agarwal, 28:11-32.*

231. Figure 9C shows the “cache coherence case.”

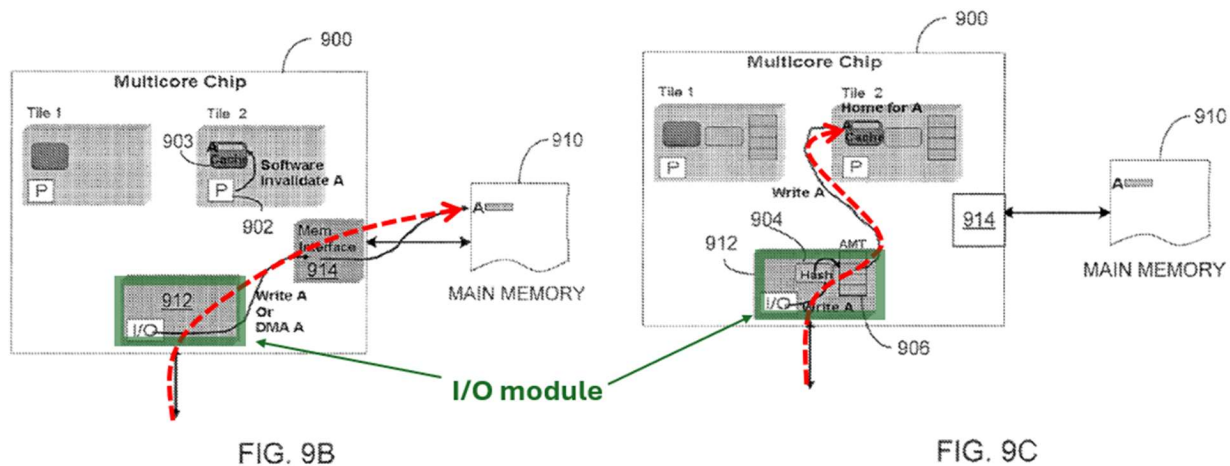


FIG. 9C illustrates an example in which a ProxyIO coherence protocol is used for situations in which an *I/O module 912 writes or reads data to or from a tile's cache....* The home tile accepts the write transactions *from the I/O module 912 just as it would write transactions from a tile.* The home 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.”

232. *Agarwal*, 29:5-28. See also 1[c], 1[d] *supra* (describing *Agarwal* caching agent managing cache reads / writes from tiles and I/O devices). *Agarwal*, 27:39-28:8, 28:23-32 (“An I/O module is also able to ... store data directly into the case of a give tile...”), 29:29-30:11, Fig. 9C. *Agarwal*’s multicore processor allocates a shared L3 cache line of the L2/L3 cache to the write data from the I/O device.

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.

Coherence processing at the home tile for memory access requests from the *I/O module 912 (“I/O coherence”)* includes a set of actions that are similar to those for PCC. However, there are some alternative coherence processing that can occur at the home tile for handling requests from the I/O module 912, as described below:

(1) In normal processing for requests from other tiles, coherence transactions write the modified byte or word in the home tile's cache. If that cache line was not present in the home tile, then the background data in that cache line are fetched from main memory and written into the cache line as well. With I/O coherence, the whole

cache line is typically written in one action, because I/O writes generally involve blocks of data that are typically bigger than a single byte or word. Thus, in the case of a request from an I/O module 912, the home tile does not need to fetch background data from main memory.

(2) 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 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.

(3) Whether it is a read or write transaction initiated by an I/O module 912, the I/O module 912 is not recorded as a sharer of the cache the line in the home tile's coherence directory. The directory does not need to keep a record of the I/O module 912 in the directory entry since the I/O module 912 does not cache the data, rather it just passes the data in or out over an I/O interface. Thus, there is no need for a directory entry since there is no need to invalidate the data at the I/O module 912 at a later point.

Various alternative can be implemented in the coherence protocol. For example, the I/O module 912 does not need to maintain the AMT and hash function if a DMA command from a tile to the I/O module 912 includes an indication of which tile to send the writes/reads to. Though, in some cases, it is useful to use the AMT and hash function to enable a fine-grain data smearing feature.

233. *Agarwal*, 29:5-30:11.

234. In the context of claim 6, *Agarwal* 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”). *Agarwal*, 29:5-30:11 (quoted above). The directory controller, in turn, allocates a shared cache line to the targeted cache. *Id.*

235. *Agarwal* thus discloses claim 6.

- f) Claim 7: “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.”

236. *Agarwal* discloses, or renders obvious, this claim.

237. As discussed for [1c] and [1d], *Agarwal*’s multicore processor supports I/O write transactions directly into the main memory in the “memory coherence case.”

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.”

238. *Agarwal*, 28:11-22,

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.

239. *Agarwal*, 28:33-43, Fig. 9B.

240. Alternatively, Agarwal's multicore processor writes transactions directly into the shared L2/L3 cache in the "cache coherence case"
Agarwal, 27:39-28:8,

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."*

241. *Agarwal*, 28:23-32, 29:5-30:11 (quoted above), Fig. 9C.

242. Claim 7 covers the "cache coherence case."

243. In *Agarwal*'s "cache coherence case," the I/O write data is written directly into the shared L2/L3 cache of the appropriate home tile (e.g., Tile 2 in Fig. 9C). *Agarwal*, 28:23-32, 29:5-30:11 (both quoted above).

244. Figure 9C below shows this process:

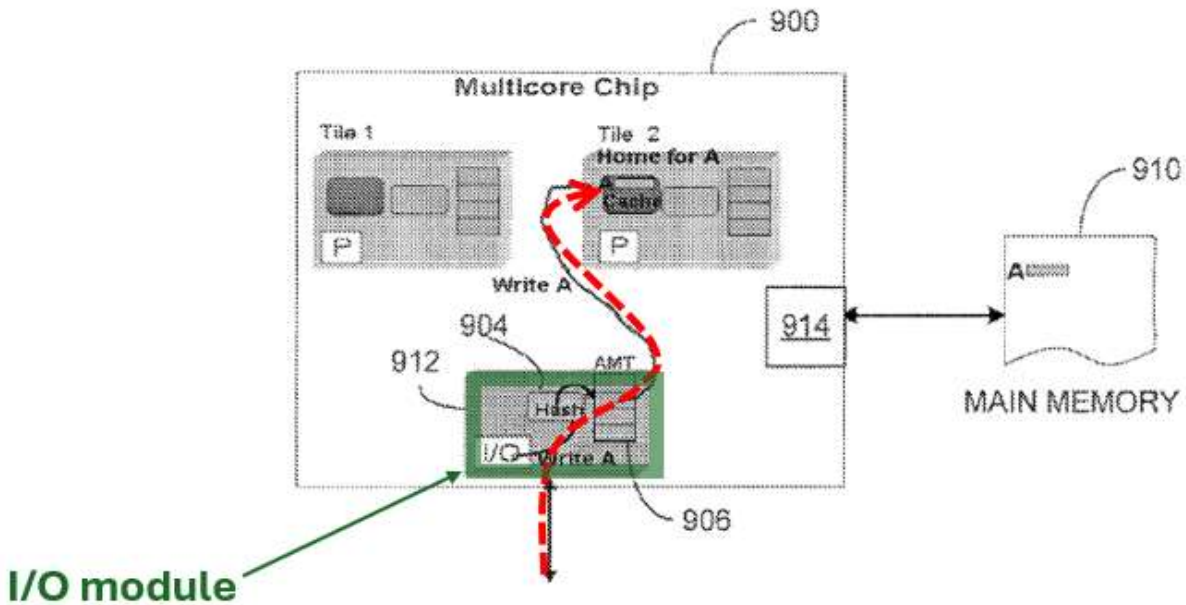


FIG. 9C

245. *Agarwal*, Fig. 9C Annotated).

246. Thus, in the “cache coherence case,” *Agarwal*’s directory controller and directory storage (i.e., “caching agent”) directly store data of the allocation transaction in Tile 2’s shared cache memory as detailed in 27:39-28:8, 28:26-32; 29:5-30:11..

247. Furthermore, as detailed for claim 6, *Agarwal* details two distinct scenarios: the “memory coherence case” (Fig. 9B) and the “cache coherence case” (Fig. 9C) In *Agarwal*’s description of the “cache coherence case” in Fig. 9C, the directory controller and directory storage do not initiate a memory transaction to a main memory (i.e., “system

memory”) unlike the system memory transaction shown in Fig. 9B. *Agarwal*, 29:5-30:11. Describing two such distinct scenarios discloses that the cache coherence case does not write into the system memory as is done in the memory coherence case.

g) Claim 11:

a. Preamble: “11. A method comprising:”

248. *Agarwal* discloses the preamble to the extent it is limiting. *Agarwal* discloses a method of maintaining cache coherency for a plurality of distributed cache regions that share the same memory space. *Agarwal*, 4:55-64, 7:22-31.

b. [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:”

249. *Agarwal* discloses, or renders obvious, this element.

250. *Agarwal* discloses a “multicore processor” as detailed above for claim element 1[a] above. (*Supra* claim element 1[a].)
251. *Agarwal* discloses “an integrated input/output (IIO) module of a multicore processor” as detailed above for claim element 1[c]. (*Supra* claim 1[c].)
252. *Agarwal* discloses “a shared cache memory of the multicore processor” as detailed above for claim element 1[b]. (*Supra* claim 1[b].)
253. As discussed for claim 6, *Agarwal* discloses “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.” (*Supra* claim 6. The same evidence provided for claim 6 thus demonstrates that *Agarwal* discloses the identical feature of element 11[a].
254. Element [11a] additionally recites “the allocation transaction comprising a request to directly store data of the allocation transaction into a shared cache memory of the multicore processor.” As discussed for claim 6, *Agarwal* discloses that an I/O device sends a request (i.e., “allocation transaction”) directly to the L2/L3 cache of a home tile,

coordinated via directory controller 603 and directory storage 604.

(*Supra* claim 6; *Agarwal*, 27:39-28:8, 28:23-32, 29:5-30:11, Fig. 9C.

255. And as discussed for claim 7, this transaction comes as a result of a request from an IO device. *Supra* claim 7; *Agarwal*, 29:5-30:11. The same evidence provided for claims 6 and 7 thus demonstrate that *Agarwal* discloses the identical feature of 11[a].

- c. [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”

256. *Agarwal* discloses, or renders obvious, this element.

257. *Agarwal* discloses “a caching agent of the multicore processor” as detailed above for claim element 1[d]. (*Supra* claim 1[d].)

258. Furthermore *Agarwal* discloses “the caching agent shared by a plurality of cores of the multicore processor and the IIO module” as detailed above for claim element 1[e]. (*Supra* claim 1[e].)

259. As discussed for [1d] and [1e], *Agarwal* discloses a “shared” single caching agent with distributed portions that maintain cache coherency for the plurality of cores and IIO module. (*Supra* [1d], [1e]).

260. As discussed for claim 6, *Agarwal* discloses “wherein the caching agent [of the multicore processor] is to receive an allocation transaction from the IIO module.” *Supra* claim 6. The discussion of claim 6 above explains that (1) *Agarwal*’s I/O module receives a request (“allocation transaction”) from an IO device coupled to the multicore processor, and (2) *Agarwal*’s directory controller 603 and directory storage 604 for the home tile (part of the distributed “caching agent”) receives that request from the I/O module. *Supra* claim 6. In other words, the I/O module “forwards” the IO device’s request (“allocation transaction”) to the directory controller 603 and directory storage 604 of the appropriate home tile (i.e., to the “caching agent”). *Supra* claims 6, 7. *Agarwal* thus discloses forwarding the allocation transaction to a caching agent of the multicore processor.

261. Claim 11[b] also recites: “the caching agent is shared by a plurality of cores of the multicore processor and the IIO module.” But as discussed for [1d] and [1e], *Agarwal* discloses a single caching agent with

distributed portions that maintain cache coherency for the plurality of cores and I/O module. *Supra* [1d], [1e]. The same evidence provided for claims 1 and 6 thus demonstrate that *Agarwal* discloses this feature of element 11[b].

- d. [11c] “directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.”

262. *Agarwal* discloses, or renders obvious, this element. As discussed for claim 7, *Agarwal* discloses that “directly storing the data of the allocation transaction into the shared cache memory of the multicore processor.” The same evidence provided for claim 7 thus demonstrates that *Agarwal* discloses the features of 11[c].

- h) Claim 12: “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.”

263. *Agarwal* discloses, or renders obvious, this claim.

264. A POSITA would understand that claim 12 essentially describes the purpose of a cache: that data can be maintained locally for access(es) so that the (slower) system memory does not have to be accessed if the data is in the cache.

265. The 228 patent agrees with this understanding when explaining this process “With the data now present in the LLC, it can be accessed a number of times for read and write operations by any of the cores of the processor as well as the initiating IO device or other IO devices coupled to the processor (block 340). Conventional cache coherency protocol operations and compliance with ordering rules can be performed when accessing this data to maintain coherency.” (‘228 patent, 3:65-4:4).

266. *Agarwal* discloses this concept. For example:

In response to a memory request, the home tile supplies the data contained in the address *a* (the relevant cache line denoted as <*a*>) to the requesting tile 2 if tile 1 has that cache line in its cache. If it does not, then tile 1 acts as a proxy for tile 2 and sends the request for address *a* to main memory (DRAM) (denoted as action A3). Main memory supplies the data <*a*> to tile 2 (denoted as action A4). Then

tile 2 supplies the data to the cache 602 of tile 1 (denoted as action A5). The cache 602 then supplies the data to the processor 600 Action A6). ... In this way, the other caches in the system serve as a higher level cache for Tile 2.

... However, *when the home tile has the data in its cache, the request to main memory can be eliminated*. ...Furthermore, notice that *once the home tile has the data, it can further supply the data to other tiles* (e.g., tile 3) *without having to go to main memory*.

267. *Agarwal*, 18:1-23.

268. Moreover, as discussed for claims 7 and 11, *Agarwal* discloses a “cache coherence case,” wherein I/O write data is written directly into the shared cache lines of the L2/L3 cache without initiating a main memory (i.e., “system memory”) update transaction. *Supra* claims 7, 11; *Agarwal*, 23:26-41, 28:23-32, 29:5-30:11, Fig. 9C.

269. Thus, tiles (with processor cores) can access (including both reading and writing) data as many times as needed until a write-back policy for writing copies of cache lines stored as shared L3 cache lines causes the cache line to be written to main memory. *Agarwal*, 24:61-64 (“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.”)

270. *Agarwal* provides further examples of how a processor core searches for and accesses data through the shared cache. First, the processor searches for data in the L1 or L2 caches.

4.1 Simple Read Flow

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.

4.2 Simple Write Flow

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.

271. *Agarwal*, 25:28-45.

272. If the search fails (i.e., a cache “miss” occurs) in the private L1 and L2 caches, the “home tile [for that memory access] checks its [L3] cache for the request block, and assuming hit, returns the data to the requesting tile.” *Agarwal*, 25:28-45.

273. If that search results in a hit, the data is returned—all without going out to the main memory. *Agarwal*, 25:35-37 (“The home tile checks its cache for the requested block, and assuming hit, returns the data to the requesting tile.”).

274. Future requests are processed in the same way and data is returned Assuming it stays in the cache) – all “without sending a transaction to a system memory coupled to the multicore processor” because the example above is the “simple read flow” which is not limited or restricted by previous accesses. *Agarwal*’s teaching that data, once present in the cache, can be accessed without accessing memory discloses this claim.

- a. Claim 13: “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”

275. *Agarwal* discloses, or renders obvious, this claim.

276. A POSITA would understand this claim to be, at a minimum, obvious because it describes typical cache operation in which evicted cache lines with modified data are written to memory to preserve the modifications.

277. As discussed for claim 12, *Agarwal*’s L2/L3 cache uses a write-back policy, wherein new data entered into a cache line is marked as “dirty” (i.e., “modified data,” as claimed). *Supra* claim 12;

The cache 206 uses a ***write-back policy for writing copies of cache lines stored as shared L3 cache lines***,

278. *Agarwal*, 24:61-64.

On the displacement of an L2 block, ***the block is either written back to memory (if dirty) or dropped (if clean)***.

279. *Agarwal*, 21:55-56,

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.

280. *Agarwal*, 19:47-52,

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).

281. *Agarwal*, 20:39-42.

282. Because *Agarwal* explains that cache 206 uses a write-back policy, *Agarwal* discloses the features of claim 13.

283. *Agarwal* teaches that if a “dirty” cache line is evicted, that data is written back to main memory (i.e., “system memory”), and any copies of that cache line are invalidated in the other tiles. *Agarwal*, 26:23-30 (“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.”). *See also Agarwal*, 21:55-56 (explaining “the block is either written back to memory (if dirty) or dropped (if clean)”).

284. This was and remains a well-understood and fundamental part of using a write-back policy.

285. *Agarwal* thus discloses that, “when the cache line includes modified data,” “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,” as claimed.

b. Claim 14: “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.”

286. *Agarwal* discloses, or renders obvious, this claim.

287. As discussed above for claims 7, 11 and 12, *Agarwal* discloses 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. *Supra* claims 7, 11, 12; *Agarwal*, 23:26-41, 28:23-32, 29:5-30:11, Fig. 9C All quoted and / or copied above).
288. The same evidence provided for claims 7, 11 and 12 thus demonstrates that *Agarwal* discloses the features of claim 14.

C. Ground 2A: Claims 1, 4, 6, 7, and 11-14 Are Obvious Over Sinharoy.

289. It is my opinion that Sinharoy, by itself, discloses or renders obvious claims 1, 4, 6, 7 and 11-14.

3. Overview Of Sinharoy

290. I have provided a detailed discussion above for Sinharoy in Section IX.A

291. Below I do an element by element comparison for Sinharoy.

4. Sinharoy Discloses The Elements Of Claims 1-2, 4-7 And 11-14.

i) Claim 1

vii. 1[pre]: “An apparatus comprising:”

292. I have been advised that preamble can be limiting in certain circumstance but I do not need to make any opinions herein whether this preamble is limiting because Sinharoy discloses a “multicore server processor” (i.e., an apparatus).

IBM POWER7 multicore server processor

The ***IBM POWER processor*** is the dominant reduced instruction set ***computing microprocessor*** in the world today, with a rich history of implementation and innovation over the last 20 years. In this paper, we describe the key features of the POWER7 processor chip. ***On the***

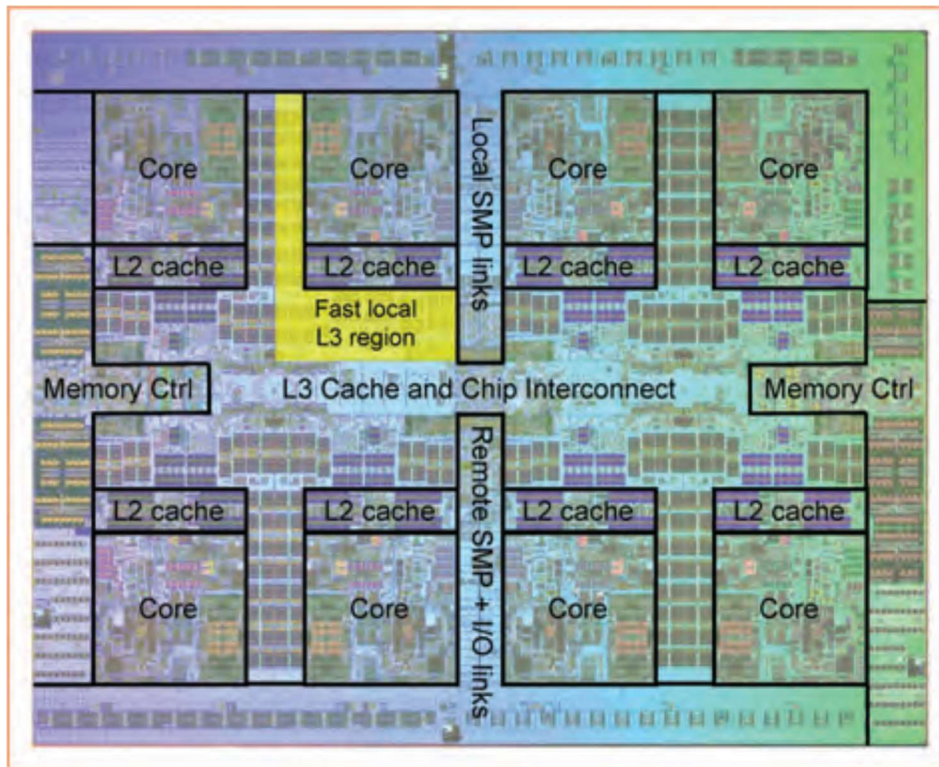
chip is an eight-core processor, with each core capable of four-way simultaneous multithreaded operation. Fabricated in IBM's 45-nm silicon-on-insulator (SOI) technology with 11 levels of metal, the chip contains more than one billion transistors. The processor core and caches are significantly enhanced to boost the performance of both single-threaded response-time-oriented, as well as multithreaded, throughput-oriented applications. The memory subsystem contains three levels of on-chip cache, with SOI embedded dynamic random access memory (DRAM) devices used as the last level of cache. A new memory interface using buffered double-data-rate-three DRAM and improvements in reliability, availability, and serviceability are discussed in reliability, availability, and serviceability are discussed

293. (*Sinharoy*, 1 (emphasis added), see also 23-25, Figs. 11-13).

viii. 1[a]: “a multicore processor including a plurality of cores”

294. *Sinharoy* discloses, or renders obvious, this element.

295. *Sinharoy*'s “multicore server processor” (i.e., “multicore processor”) includes “eight high-performance cores” (i.e., “a plurality of cores”). (*Sinharoy*, 1-2, see full quote from p. 1 in 1[pre] element above).



296. *Sinharoy*, Fig. 1.

297. “Figure 1 shows the **POWER7 chip**, which has **eight processor cores**, each with 12 execution units capable of simultaneously running four threads. To feed these eight high-performance cores, POWER7 has two memory controllers—one on each side of the chip. Each memory controller supports four channels of double-data-rate-three (DDR3) memory. These eight channels together provide 100 GB/s of sustained memory bandwidth. At the top and the bottom of the chip are the SMP

links, providing 360 GB/s of coherency bandwidth that allows POWER7
to efficiently scale to 32 sockets.” *Sinharoy*, 2 (emphasis added).

298. Thus, *Sinharoy* discloses, or renders obvious, this element.

ix. 1[b]: “a shared cache memory”

299. *Sinharoy* discloses, or renders obvious, this element.

300. *Sinharoy* discloses element [1b]. *Sinharoy*’s multicore server
processor includes a “32-MB-shared-L3 cache” (i.e., “shared cache
memory”). (*Sinharoy*, 2)

301. Figure 1 (shown above) shows the exemplary 4-MB slice of L3 cache
present in each of the 8 cores that is shared among the eight cores and the
I/O units. “Therefore, the POWER7 cache hierarchy includes ***a shared
32-MB L3 cache comprised of the 4-MB local L3 regions from the
eight cores that reside on the processor chip.***” *Sinharoy*, 17 (emphasis
added).

302. The details of the shared cache and its control are discussed below and
in *Sinharoy* at 17-22 and in Tbl 1.

303. Thus, *Sinharoy* discloses, or renders obvious, this element.

x. 1[c]: “an integrated input/output (IIO) module to
interface between the multicore processor and at

least one IO device coupled to the multicore processor”

304. *Sinharoy* discloses, or renders obvious, this element.

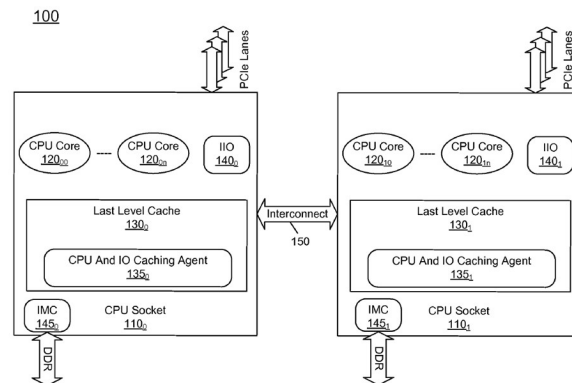


FIG. 1

‘228 patent at FIG. 1.

305. The ‘228 patent provides guidance about the claimed IIO module in the specification. “Referring to FIG. 1, shown is a block diagram of *a portion of a multiprocessor system* in accordance with an embodiment of the present invention. As shown in FIG.1, system 100 includes a pair of processor sockets 110₀ and 110₁. In the embodiment shown, *each socket 110 (generally) corresponds to a multicore processor*. For ease of discussion, reference will be made to the components within socket 110. In the embodiment of FIG. 1.....More specifically, socket 110₀ *may include an IO module (hereafter an integrated IO module or IIO module)*. This *IIO module may act as an interface to one or more off-*

chip peripheral or IO devices (not shown in FIG. 1) *coupled to the processor*, e.g., via PCIe™ links. Examples of such IO devices include *storage devices, network interfaces, graphics cards, and so forth.*” (Ex-1001, 1:65-2: 26.)

306. *Sinharoy* teaches two types of such IIO modules that provide off chip interfaces to the multicore IBM POWER 7 processor. “As shown in Figure 1, the POWER7 chip contains eight cores, each with an associated L2 cache and local L3 region, two memory controllers, *two I/O controllers*, and a *multichip SMP interconnect* capable of scaling up to 32 POWER7 chips into a single large SMP system. Consequently, each POWER7 chip *contains several coherence request, snoop, response, and notification interfaces and several data interfaces.*” (*Sinharoy*, 22, Figure 1 (duplicated above)).

307. The first type of IIO module is the “I/O controller.” *Sinharoy* discloses “two integrated I/O controllers,” each being an “integrated input/output (IIO) module,” as claimed. (*Sinharoy*, 22) These integrated I/O controllers connect (i.e., “interface between”) one or more input/output devices (e.g., PCI device) and the multicore processor via an I/O hub. (*Sinharoy*, 17, 22, Fig. 14 (illustrating a PCI adapter connected

to I/O hub). Sinharoy's "POWER7 I/O controllers also support cache injection, which enables DMA write traffic to be written directly into a cache, instead of to memory." (Sinharoy, 22.) As noted above, I have reviewed filings from IPR2023-01344. In that IPR, the patent owner did not challenge that *Sinharoy*'s two integrated I/O controllers disclosed the claimed IIO module. This supports my conclusion.

308. The second type of IIO module is the "SMP link." To network several POWER 7 chips to create a larger network SMP cluster, Sinharoy discloses a "cluster interconnect" or "multichip SMP interconnect" to a "fabric interface". ("Examples of such IO devices include ***storage devices, network interfaces, graphics cards, and so forth.***" (Ex-1001, 2:22-26). (*Sinharoy*, 22-23).

309. Sinharoy's Fig. 14 depicts an example of these I/O modules interfacing to I/O devices: An I/O controller via a "GX I/O bus" to an I/O hub that connects to a "PCI adapter" and the SMP links that interface to a network interface (e.g. "fabric interface").

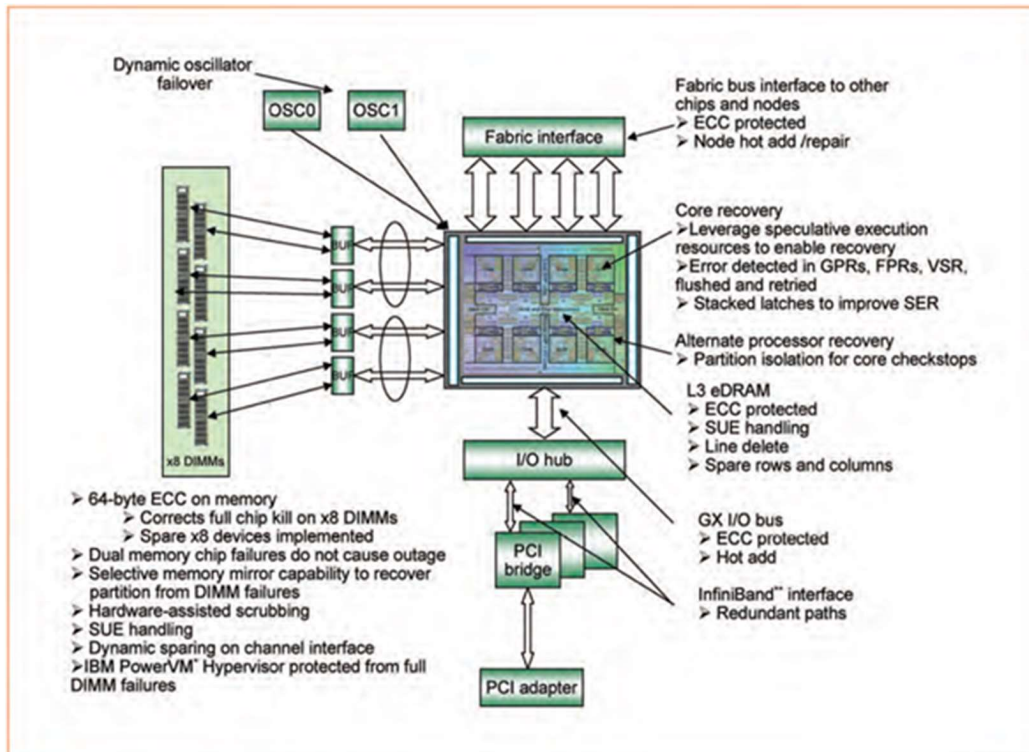


Figure 14

IBM POWER7 reliability and availability features.

310. (Sinhroy, Fig. 14.)

311. “The POWER7 chip supports *two integrated I/O controllers*.

Each of the controllers supports a proprietary 4-byte off-chip read interface and a 4-byte off-chip write interface, thereby *connecting the POWER7 chip to up to two I/O hub chips*. These *interfaces* operate at frequencies ranging from 1.25 to 2.5 GHz, depending on the I/O hub chips to which they are connected..... the POWER7 *I/O controllers* also

support cache injection, which enables ***DMA write traffic to be written directly into a cache, instead of to memory.***” *Sinharoy*, Col 2, p 1:22.

312. Fig. 13 shows the cluster network interconnects:

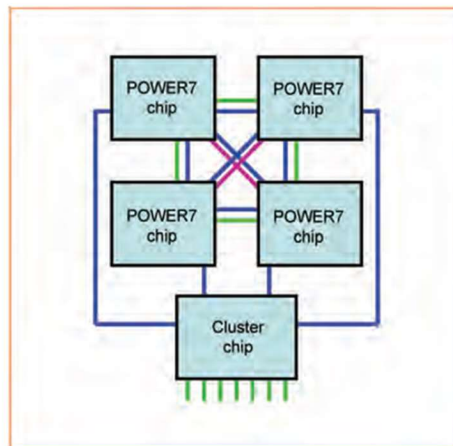


Figure 13

Cluster system first-level coherent nodal topology.

313. (*Sinharoy*, Fig. 13). *Sinharoy*’s network interface (“cluster interconnect” logic) teaches a network interface to interface to other POWER7 systems. .

314. “For massive multipeta-FLOPs high-bandwidth clustered systems, a specialized cluster interconnect chip has been developed. The chip couples the POWER7 coherence and data interconnect seamlessly ***into a cluster interconnect.*** In this way, ***32-way flat SMP systems are coupled to form 256-way SMP systems,*** which ***are interconnected to create a massive 512-K processor cluster.***” *Sinharoy*, Col 1, p 1:25.

315. “The cluster interconnect chip has an on- and off-chip coherence and data interconnect that interoperates with the POWER7 chip and provides an on-board memory coherence directory (MCD) that expands upon the dual-scope capabilities provided by the POWER7 13-state coherence protocol. The *MCD improves I/O subsystem throughput by qualifying and vectoring DMA reads and writes to a single target node*, instead of broadcasting them to the entire SMP system. It also *accelerates streaming memory write operations and reduces coherence scope misprediction latencies.*” *Sinharoy*, Col 1, p 1:25

316. “Each cluster interconnect chip incorporates up to *three 16PCI Express** Gen2 I/O subsystem interfaces*, a proprietary *clustering interconnect interface*, a high-bandwidth integrated switch and router, and a set of accelerators to process collective operations.” *Sinharoy*, Col 1, p 1:25.

317. “The cluster interconnect chip has four 10 B/s on-node *SMP links* to *directly connect to four POWER7 chips*. Figure 13 illustrates how *four POWER7 chips and one cluster interconnect chip are fully interconnected* to form a flat 32-way SMP nodal construct with an integrated cluster interface. As described in *Table 4*, the connections

between the four POWER7 chips are doubled, providing twice the nodal interconnect bandwidth as, compared with a standard system.” *Sinharoy*, Col 1, p 1:25.

318. Table 4 below highlights some of the network interfaces used in the cluster system.

Table 4 IBM POWER7 standard system and cluster system interfaces.

<i>Logical interface</i>	<i>Standard system</i>	<i>Cluster system</i>
POWER7 to POWER7 nodal interconnect	Use three of four POWER7 on-node SMP links	Use three of four POWER7 on-node SMP links
POWER7 to POWER7 nodal interconnect 2× bandwidth expansion	n/a	Use two POWER7 off-node SMP links and both GX I/O links
POWER7 to cluster chip nodal interconnect	n/a	Use fourth POWER7 on-node SMP link
System interconnect	Use one or two POWER7 off-node SMP links	Use seventh cluster chip off-node SMP/cluster links
I/O subsystem attachment	Use POWER7 GX I/O links	Use cluster chip PCI interfaces
Memory channels	Use eight POWER7 channels	Use eight POWER7 channels

319. (*Sinharoy*, Table 4, annotation added)

320. “Each cluster interconnect chip has seven 10 B/s *off-node SMP links*, enabling each nodal construct to directly *connect to seven other nodal constructs*, forming an eight-node 256-way SMP system, primarily to facilitate *coherent I/O sharing among the eight flat 32-way SMPs*.” *Sinharoy*, Col 1, p 1:25.

321. Thus, *Sinharoy* discloses this element.

- xi. 1[d]: “a caching agent to perform cache coherency operations for the plurality of cores and the IIO module”

322. *Sinharoy* discloses, or renders obvious, this element.

323. *Sinharoy*’s distributed “coherence fabric” comprises eight L3 cache controllers (i.e., “caching agent” as claimed). (*Sinharoy*, 19; *see also Sinharoy*, 17-19.)

324. *Sinharoy*’s multicore processor includes three cache layers. The L1 and L2 caches are private to each core, and the 32-MB L3 cache is shared by all cores. (*Sinharoy*, 19.)

Table 1 Comparison of POWER6 and POWER7 cache hierarchies.

POWER6 (assuming 3-GHz core)	POWER7 (assuming 4-GHz core)
64 KB store-through L1 D-cache 0.8ns latency, 80 GB/s private	32 KB store-through L1 D-cache 0.5ns latency, 192 GB/s private
4 MB store-in L2 cache ~5.0-ns latency, 160 GB/s private	256 KB store-in L2 cache 2.0-ns latency, 256 GB/s private
	4 MB partial victim local L3 region ~6.0-ns latency, 128 GB/s private
32 MB victim L3 cache ~35-ns latency, 80 GB/s shared by 2	32 MB adaptive victim L3 cache ~30-ns latency, 512 GB/s shared by 8

325. *Sinharoy*, Tbl. 1

326. This L3 cache comprises eight distributed 4-MB local L3 regions – with each region incorporated into a core to provide faster access for that corresponding core. (*Sinharoy*, 19.)

“The *shared 32-MB POWER7 L3 cache* is composed of *eight 4-MB L3 regions*. Each L3 region utilizes 128-byte lines and is eight-way set associative. *A given L3 region comprises a single centralized controller*, with four address hashed eDRAM cache data banks and four address hashed SRAM directory banks. *The local L3 region is tightly coupled to the L2 cache associated with a given core.*”

327. *Sinharoy*, 19.

“At a higher level, by incorporating the eDRAM L3 cache on the processor die, a 4-MB local L3 region enjoys roughly one sixth the access latency of an off-chip L3 cache, eliminating the need for a separate 4-MB on-chip SRAM cache to provide equivalent low latency access.”

328. *Sinharoy*, 19.

329. Figure 10 shows the 4 MB local L3 regions operating as a single caching agent with the relationship between the individual portions shown.

330. Example of their operation as a single caching agent is at least Figure 10 “Lateral castout(s)” shown between the various L3 regions (emphasized in red box below), “the capacity, data flow logic, and

control flow logic of a 4-MB local L3 region are shared between the L2 cache to which the local L3 region is coupled, ***and the other seven 4-MB L3 regions on the chip,*** “the POWER7 L3 replacement policy incorporates significant enhancements and ***exploits the semantic content of the 13 cache states to manage the aggregation of the 4-MB regions,***” and “when the core and L2 associated with a given L3 region are disabled, ***the remaining L3 regions enjoy full access to the L3 region.***”

331. “Storage accesses that miss both the L2 cache and the 4-MB local L3 region are broadcast to the coherence fabric and snooped by the memory controller, other L2 caches, possibly other L3 caches, and ***by the seven remote 4-MB L3 regions that comprise the remaining 28 MB of the on-chip L3 cache.*** Therefore, ***operations 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*** (see targeted cases below) or via the snoop dispatch ports (see state merge cases below).”

332. “As illustrated in Figure 10, the capacity, data flow logic, and control flow logic of a 4-MB local L3 region are shared between the L2 cache to which the local L3 region is coupled, **and the other seven 4-MB L3 regions on the chip**. The associated L2 cache utilizes the 4-MB local L3 region directly as a victim cache. The *other seven 4-MB L3 regions on the chip also use the 4-MB local L3 region as a victim cache under certain circumstances*. Note that when the core and L2 associated with a given L3 region are disabled, *the remaining L3 regions enjoy full access to the L3 region.*”

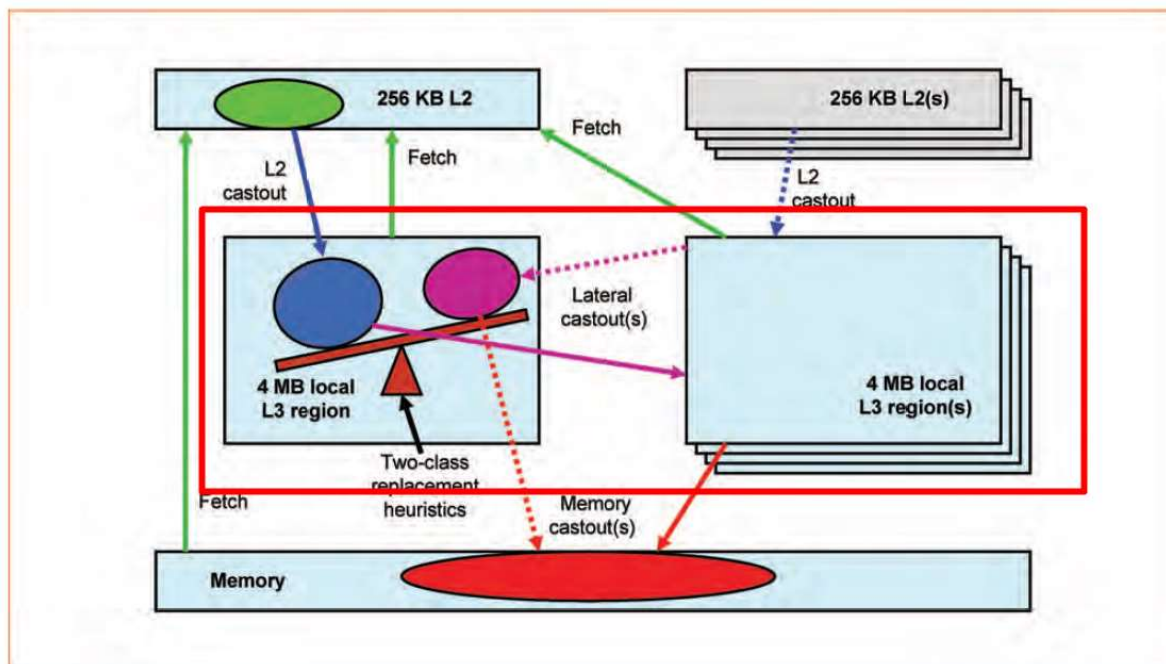


Figure 10

L3 region behavior.

333. *Sinharoy*, Fig. 10 (emphasis added).

334. *Sinharoy*'s lateral cast-out is an example of the collective operation of the distributed portions of the caching agent within *Sinharoy*. "When a lateral cast-out from an L3 region finds a copy of the line (in a subset of the valid states) in any of the other L3 regions within the lateral cast-out broadcast scope, *the states of the two copies are merged at the destination, and no data is moved*, saving energy (similar to the L2 cast-out case). Otherwise, *a set of heuristics weigh the usage pattern (both capacity usage and traffic volume) of the targeted region to determine whether to accept the lateral cast-out*. If the lateral cast-out is *accepted by the targeted L3 region and needs to allocate a line* (i.e., it does not find an invalid slot), it also preferentially selects the LRU line from the second class, only displacing a first-class line if no second-class lines exist. In order to allow for growth in the set of candidate second-class lines, *whenever a lateral cast-out is accepted by a given L3 region, the LRU first-class line begins a transformation that will eventually convert it to a most recently used second class*, thereby gradually enlarging the supply of second-class lines. Lateral cast-out operations that move data to a targeted L3 region *are managed in the L3 region by*

a pool of ten L3 write machines and ten associated L3 cast-out machines.”

335. *“Coherence operations for the plurality of cores.”* Sinharoy’s eight

L2 and L3 caches (one each per core) each includes a controller to help maintain cache coherency across all eight cores. (*Sinharoy*, 18-19.)

The private 256-KB POWER7 L2 cache utilizes 128 byte lines and is eight-way set associative. It is *comprised of a single centralized controller*, with two address-hashed cache data arrays.

336. *Sinharoy*, 18.

The shared 32-MB POWER7 L3 cache is composed of eight 4-MB L3 regions. Each L3 region utilizes 128-byte lines and is eight-way set associative. A given L3 region comprises a single centralized controller, with four address hashed eDRAM cache data banks and four address hashed SRAM directory banks.

The *centralized L3 region controller provides a single core/L2 dispatch port, a single lateral L3 region dispatch port, dual snoop dispatch ports (for even and odd cache lines), and a single pool of operational resources.*

337. *Sinharoy*, 19.

338. For *example*, each L2 and L3 controller supports the “13-cache-state protocol” defined in Table 2. (*Sinharoy*, 17, Tbl. 2.) These cache states support “coherent operations.” (*Sinharoy*, 17.)

The *adaptive L3 cache management policy routes instructions and data to the 4-MB local L3 regions of the core or cores that most actively utilize them*, enabling them to experience reduced latency, even cloning copies when multiple cores are actively sharing them but also destroying copies as they fall into disuse in order to preserve capacity. Since the POWER7 32-MB L3 cache resides on the processor chip, it provides lower latency access (even to 28 MB composed of nonlocal L3 regions) than the POWER6 off-chip 32-MB L3 cache.

As shown in Table 2, the *POWER7 L2 and L3 caches support the same 13-cache-state protocol* as the POWER6 design point [4]. While no new cache states have been added for POWER7, *new coherent operations are supported*. One *such operation is cache injection*. An *I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory*. If a given core’s L2 cache or local L3 region owns a copy of

the targeted cache line (i.e., holds the line in an M, ME, U, T, TE, TN, or TEN cache state), the data will be installed into the local L3 region.

339. *Sinharoy*, 17.

340. To maintain cache coherency, each L2 and L3 controller includes a pool of read/claim (RC) machines and write machines that perform cache coherency operations, such as fetches, read-write operations, and core prefetch operations. (*Sinharoy*, 18-19.) *Sinharoy* explains, for example, that:

Storage accesses that miss both the L2 cache and the 4-MB local L3 region ***are broadcast to the coherence fabric*** and snooped by the memory controller, ***other L2 caches, possibly other L3 caches, and by the seven remote 4-MB L3 regions that comprise the remaining 28 MB of the on-chip L3 cache.*** Therefore, operations 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 (see targeted cases below) or via the snoop dispatch ports (see state merge cases below). All of the cast-out scenarios are described in detail in the subsequent paragraphs.

As illustrated in Figure 10, *the capacity, data flow logic, and control flow logic of a 4-MB local L3 region are shared between the L2 cache to which the local L3 region is coupled, and the other seven 4-MB L3 regions on the chip.* The associated L2 cache utilizes the 4-MB local L3 region directly as a victim cache. The *other seven 4-MB L3 regions on the chip also use the 4-MB local L3 region as a victim cache under certain circumstances.* Note that when the core and L2 associated with a given L3 region are disabled, *the remaining L3 regions enjoy full access to the L3 region.*

341. Sinharoy, 19.

342. *“Coherence operations ... for the IIO module.”* Sinharoy’s cache architecture also supports “coherent operations” like “cache injection” that allows an I/O device to perform coherency operations (e.g., data reads or writes) targeted directly at the cache.

In addition to supporting the interrupt presentation function, *pipelined I/O high-throughput mode, and advanced partial DMA write management* pioneered in the POWER6 design, the POWER7 I/O controllers also support cache injection, which enables DMA write traffic to be written directly into a cache, instead of to memory.

343. *Sinharoy, 22.*

As shown in Table 2, the POWER7 L2 and L3 caches support the same 13-cache-state protocol as the POWER6 design point [4]. While no new cache states have been added for POWER7, *new coherent operations are supported. One such operation is cache injection.* An I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory. If a given core's L2 cache or local L3 region owns a copy of the targeted cache line (i.e., holds the line in an M, ME, MU, T, TE, TN, or TEN cache state), the data will be installed into the local L3 region.

344. *Sinharoy, 17.* The write machines in the L3 cache controller control the injections:

When prefetched data is staged to the L3 cache, **it is managed by a pool of ten write machines.** Each *write machine has an associated*

cast-out machine to manage the eviction of lines displaced by writes.

Note that these *write machines are also utilized by other operations such as cast-outs and cache-injections.*

345. *Sinharoy*, 19.

346. (*Sinharoy*, 17, 22) For example, when an I/O device sends a write command to the CPU caches via the *integrated* I/O controller, “[i]f a given core’s L2 cache or local L3 region owns a copy of the targeted cache line ... the data will be installed into the local L3 region.” (*Sinharoy*, 17.)

347. For the second IIO module identified above in element [1c], *Sinharoy* discloses the SMP links used for “multichip SMP interconnect” discussed in [1c] as an IIO module (e.g. “network interface.”) “As shown in Figure 1, the POWER7 chip contains ..._two I/O controllers, and a multichip SMP interconnect capable of scaling up to 32 POWER7 chips into a single large SMP system.” (*Sinharoy*, 22.) and “Each chip has four 10-B/s on-node **SMP links** associated with the vertical spine that emanates from the center of the chip toward the top edge. In a standard commercial system, *three of these four SMP links are connected to each of the three other chips that comprise a four-chip node.* In this manner, the

four chips are fully connected.” (Sinharoy, 23-24.) Sinharoy’s multicore processor maintains cache coherency for the cores and the IIO modules (integrated I/O controllers and SMP links) through an on-chip coherence interconnect that includes “***coherence request, snoop, response, and notification interfaces and several data interfaces.***” (Sinharoy, 22)

348. Sinharoy details the operation of the multichip SMP to provide a network interface to provide a multichip network on pages 22-23.

As shown in Figure 1, the POWER7 chip contains eight cores, each with an associated L2 cache and local L3 region, two memory controllers, two I/O controllers, and a multichip SMP interconnect capable of scaling up to 32 POWER7 chips into a single large SMP system. Consequently, each POWER7 chip contains several coherence request, snoop, response, and notification interfaces and several data interfaces.

These are tied together by a high-bandwidth intelligent on-chip coherence and data interconnect that physically occupies a horizontal equatorial trunk that runs across the center of the POWER7 chip. An emerging necessity, due to the multicore nature of the POWER7 design, the on-chip interconnect also manages the POWER7

capability to independently adjust individual core frequencies and accommodates the ensuing coherence and data flow variations.

....

The *on-chip coherence interconnect routes two sets (even and odd cache line) of coherence requests* through the horizontal trunk, inward toward the even/odd arbitration logic located at the center of the chip. Up to one even request and one odd request may be granted each on-chip bus cycle. Once granted, the *requests are broadcast within the chip* on the even/odd snoop buses outward toward the left and right edges of the chip. Requests that *will be routed to other chips are also sent to the multichip SMP interconnect* (discussed in the multichip interconnect section) via a *central vertical spine toward the top and bottom edges of the chip*. Requests that have *arrived from other chips are managed* by the even/odd arbitration logic and broadcast to the snoop buses.”

Coherence responses from the snoopers are routed inward along the horizontal trunk toward the even/odd coherence decision logic located at the center of the chip. For *requests that have been routed to other chips*, additional responses from the off-chip snoopers are *fed into the*

coherence decision logic. Once a *final coherence decision is made in response to a given request*, a notification is broadcast within the chip on the even/odd notification buses outward from the center toward the left and right edges of the chip. *Notifications that will be routed to other chips are also sent to the multichip SMP interconnect* via the central vertical spine toward the top and bottom edges of the chip.

349. Sinharoy, 22-23.

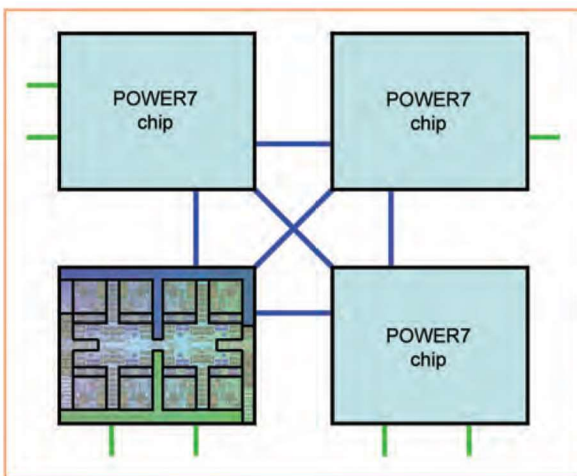


Figure 11

IBM POWER7 first-level nodal topology.

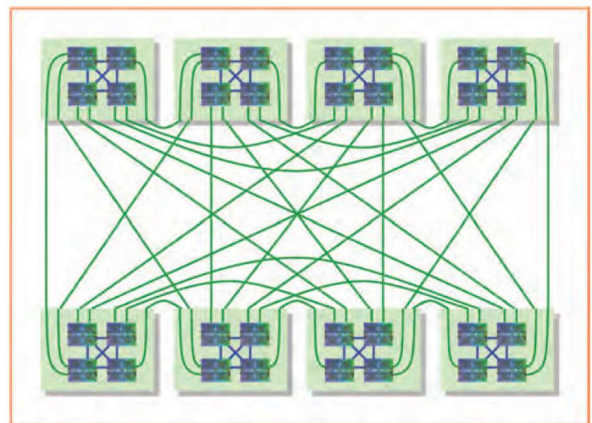


Figure 12

IBM POWER7 second-level system topology.

Sinharoy Figure 11 and Figure 12.

350. Sinharoy teaches the details of this “multichip interconnect” acting as the network interface.

Multichip Interconnect

The POWER7 system topology is built upon the structure that was developed for POWER6 systems. For standard commercial systems, up to 32 POWER7 chips *may be connected* to form a single 256-way SMP system.

Figure 11 depicts a *first-level nodal structure*, which *combines up to four POWER7 chips*. Each chip has *four 10-B/s on-node SMP links* associated with the vertical spine that emanates from the center of the chip toward the top edge. In a standard commercial system, *three of these four SMP links are connected to each of the three other chips that comprise a four-chip node*. In this manner, the *four chips are fully connected*.

A second-level system structure combines up to eight of the nodes. Each chip has two 10B/s off-node SMP links associated with the vertical spine that emanates from the center of the chip toward the bottom edge. As shown in Figure 12, in a standard commercial system, up to seven of the eight off-node SMP links (coming from the four POWER7 chips comprising a node) are connected to each of the seven other nodes that comprise an eight-node system.

Just as in POWER6 systems, the ***POWER7 SMP links are shared by coherence and data traffic*** and are protected by single-error-correct double-error-detect (SECDED Hamming code with additional parity) ECC. Unlike POWER6 systems, which enforce a strict 50% coherence to 50% data ratio or 33% coherence to 67% data ratio, the ***POWER7 SMP links enable a dynamic free-form allocation of coherence and data traffic***, enabling higher effective utilization of the links. Additionally, to maximize SMP link bandwidth, the on- and off-node SMP links do not operate at the on-chip frequency. They are independently tuned, typically operating at speeds ranging from 2.5 to 3.3 GHz, depending upon system packaging characteristics, and provide increased flexibility over the POWER6 SMP links.

351. *Sinharoy*, 23-24.

352. Thus, *Sinharoy* discloses this element.

- xii. 1[e]: “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.”

353. *Sinharoy* discloses, or renders obvious, element [1e].

354. *Sinharoy* teaches the same architecture that the ‘228 patent characterizes as a “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.”

355. In the ‘228 patent, the single caching agent: (1) handles cache requests and coherency for CPUs and I/O devices by implementing “various coherency logic” and (2) corresponds to an embodiment in the ‘228 patent’s, Figs. 2 and 6 (duplicated below).

356. For example, the ‘228 patent references “coherency logic”

Referring now to FIG. 2, shown is a block diagram of *a caching agent in accordance with an embodiment of the present invention.*

As shown in FIG. 2, *caching agent 200 may be logic interposed between one or more cores of a processor and a LLC 230.* In addition, *caching agent 200 may also provide an interface between an IIO module within a multicore processor and the LLC.* As seen, caching agent 200 can include *an ingress port 210* that includes multiple ingress queues 211-213. As seen, ingress port 210 may be coupled to an arbiter 215 which can arbitrate amongst incoming requests and provide them to a table of requests (TOR) 220. As seen,

TOR 220 may be a buffer or other temporary storage for holding incoming requests. In the embodiment shown, TOR 220 may include at least one dedicated resource 225 for posted transactions. In one embodiment, dedicated resource 225 may be a single entry of the queue, although the scope of the present invention is not limited in this regard.

Still referring to FIG. 2, TOR 220 interfaces with LLC 230. More specifically, *this LLC may be a bank or other portion of the LLC associated with the caching agent*. To provide access to other locations within the processor via a ring interconnect 260, an ingress queue 240 and an egress queue 250 may be provided. Thus *via caching agent 200, cores of the processor and an IIO module can maintain coherency without the need for additional caching agents*. Although not shown for ease of illustration, understand that *caching agent 200 may further include various coherency logic to perform cache coherency operations on behalf of the cores and IIO module*, and to *enable coherent access with regard to both the core transactions and IO device transactions* as described herein.

Note that although a single structure is shown in FIG. 2 for ease of illustration, understand 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.*

357. '228 patent, 2:36-3:30. Figure 6 is an embodiment of the distributed portions of the caching agent:

Referring now to FIG. 6, shown is a block diagram of a multicore processor in accordance with an embodiment of the present invention.

As shown in the embodiment of FIG. 6, processor 700 *includes a distributed configuration having partitions or slices each including a core 710 and a partition of a caching agent 715 and a LLC 720.* Note that *while distributed caching agents are shown, understand that these distributed portions form a single caching agent,* and which is *configured to handle cache coherency operations both for the cores as well as an I/O module 750.*

358. '228 patent, 6:22-29. The '228 patent's Fig. 6 shows the distributed caching agents associated with each core.

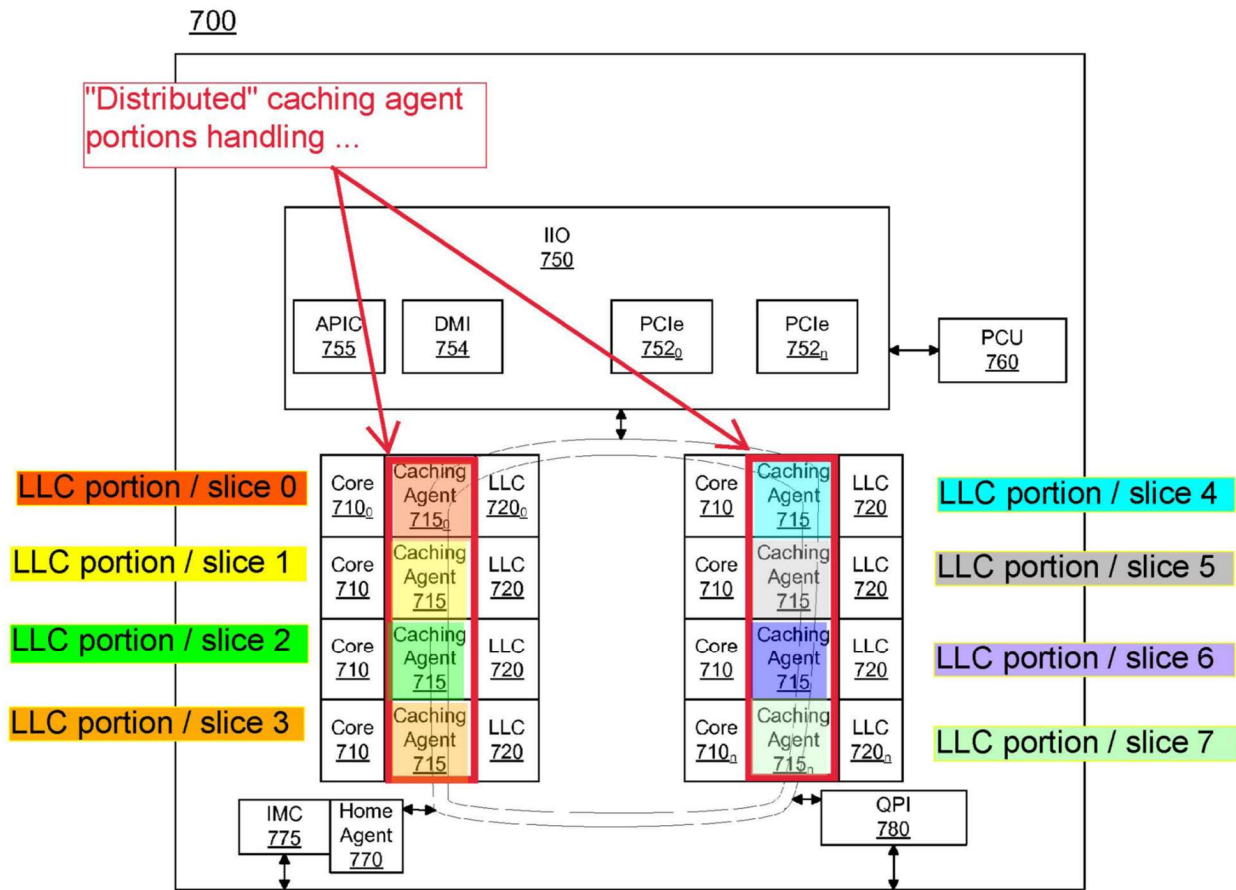


FIG. 6

359. Ex-1001, Fig. 6; 6:20-39.)

360. The '228 patent's Fig. 2 depicts a "block diagram" of one portion of the distributed caching agent in Fig. 6. (Ex-1001, 2:36-3:3.)

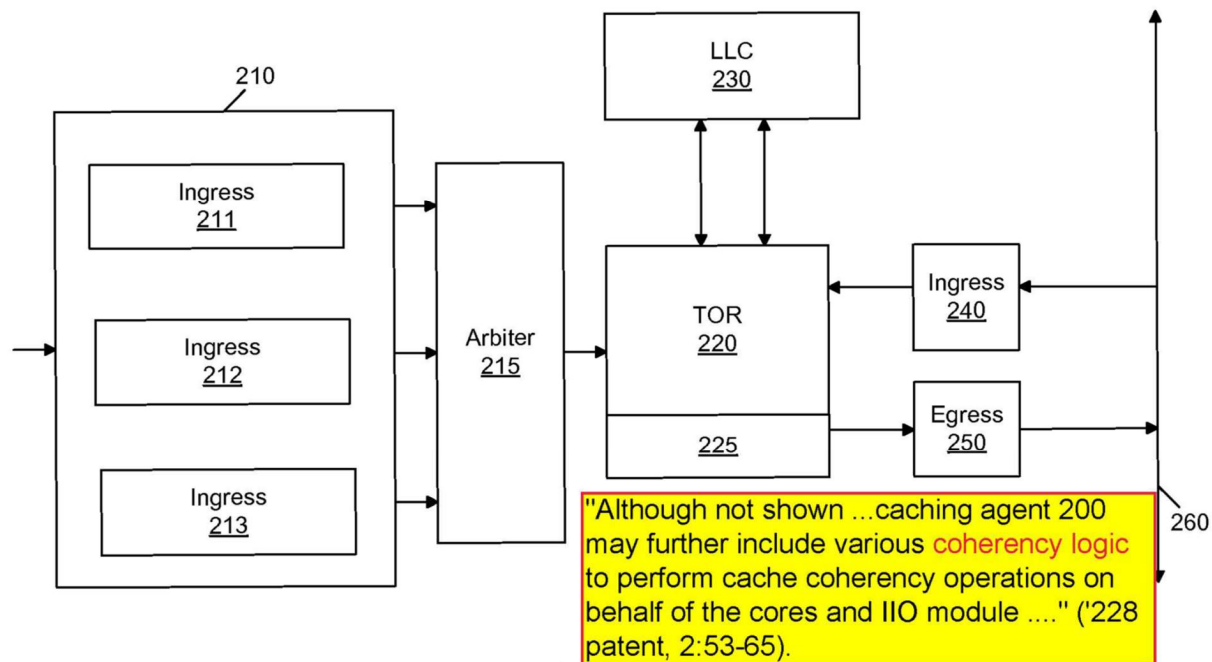
200

FIG. 2

361. (Ex-1001, Fig. 2.)

362. As annotated above in the yellow block which is a quotation from the '228 patent description of Figure 2, the Fig. 2 caching agent includes “coherency logic” to manage cache coherency. (Ex-1001, 2:53-65.)

363. Thus, each “distributed portion” caching agent in Fig. 6 above includes the Fig. 2 components shown above, including the “coherency logic” needed for cache coherency as recited in the claims. It is this “coherency logic” which is “not shown” that provides the ability to manage the cache coherency.

364. *Sinharoy's* architecture is from the same as the '228 patent's architecture above. *Sinharoy* teaches that each core includes: (1) a core, (2) a portion of the shared L3 cache; and (3) an L3 cache "controller." (*Sinharoy*, 17-19.)

Therefore, the POWER7 cache hierarchy includes *a shared 32-MB L3 cache comprised of the 4-MB local L3 regions from the eight cores that reside on the processor chip.*

365. *Sinharoy*, 17.

The shared *32-MB POWER7 L3 cache is composed of eight 4-MB L3 regions*. Each L3 region utilizes 128-byte lines and is eight-way set associative. *A given L3 region comprises a single centralized controller*, with four address hashed eDRAM cache data banks and four address hashed SRAM directory banks.

366. *Sinharoy*, 19.

367. Each of *Sinharoy's* L3 cache controller corresponds to one "caching agent 715" portion in the '228 patent's Fig. 6 above shown placed between each of the cores 710 and shared LLC cache regions 720. Like the '228 patent's "distributed" portions, *Sinharoy's* L3 cache controller cooperates with the other L3 cache controller's for the other cores to

provide a coherently managed cache across the distributed portions of the L3 cache.

368. Similarly, the architecture of the '228 patent's caching agent portions in Fig. 2 matches *Sinharoy*'s controller architecture. In the '228 patent, each core's caching agent includes various "ports" for ingress / egress to communicate with other cores and I/O ports, as well as "coherency logic" to manage coherency.

Referring now to FIG. 2, shown is a block diagram of a caching agent in accordance with an embodiment of the present invention. As shown in FIG. 2, *caching agent 200 may be logic interposed between one or more cores of a processor and a LLC 230. In addition, caching agent 200 may also provide an interface between an IIO module within a multicore processor and the LLC.* As seen, *caching agent 200 can include an ingress port 210* that includes multiple ingress queues 211-213. As seen, ingress port 210 may be coupled to an arbiter 215 which can arbitrate amongst incoming requests and provide them to a table of requests (TOR) 220. As seen, TOR 220 may be a buffer or other temporary storage for holding incoming requests. In the embodiment shown, TOR 220 may include

at least one dedicated resource 225 for posted transactions. In one embodiment, dedicated resource 225 may be a single entry of the queue, although the scope of the present invention is not limited in this regard.

Still referring to FIG. 2, TOR 220 interfaces with LLC 230. More specifically, this LLC may be *a bank or other portion of the LLC associated with the caching agent*. To *provide access to other locations within the processor* via a ring interconnect 260, an ingress queue 240 and an egress queue 250 may be provided. Thus *via caching agent 200, cores of the processor and an IIO module can maintain coherency without the need for additional caching agents*. Although not shown for ease of illustration, understand that *caching agent 200 may further include various coherency logic to perform cache coherency operations on behalf of the cores and IIO module, and to enable coherent access with regard to both the core transactions and IO device transactions as described herein*.

369. '228 patent, 2:36-2:65.

370. (Ex-1001, 2:34-65.)

371. *Sinharoy's "controller" also has various "ports" for ingress / egress to communicate with other cores and I/O ports as well as various coherency logic to perform coherency operations on behalf of the cores and IIO modules (e.g. I/O controllers and SMP links).*

The centralized L3 region controller provides a single core/L2 dispatch port, a single lateral L3 region dispatch port, dual snoop dispatch ports (for even and odd cache lines), and a single pool of operational resources.

Storage accesses that miss the L2 cache *access the 4-MB local L3 region via the core/L2 dispatch port*. Those that hit in the 4-MB local L3 region are *managed by a pool of eight read machines*. Prefetches and some L2 store misses also access the L3 region via the core/L2 dispatch port and *are managed by a pool of 24 lightweight RC machines*. When prefetched data is staged to the L3 cache, it is *managed by a pool of ten write machines*. Each *write machine has an associated cast-out machine to manage the eviction of lines displaced by writes*. Note that these *write machines are also utilized by other operations such as cast-outs and cache-injections*.

Storage accesses that miss both the L2 cache and the 4-MB local L3 region *are broadcast to the coherence fabric* and snooped by the memory controller, *other L2 caches, possibly other L3 caches, and by the seven remote 4-MB L3 regions that comprise the remaining 28 MB of the on-chip L3 cache*. Therefore, *operations 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 (see targeted cases below) or via the snoop dispatch ports (see state merge cases below)*. All of the cast-out scenarios are described in detail in the subsequent paragraphs.

As illustrated in Figure 10, *the capacity, data flow logic, and control flow logic of a 4-MB local L3 region are shared between the L2 cache to which the local L3 region is coupled, and the other seven 4-MB L3 regions on the chip*.

372. Sinharoy, 19.

373. *Sinharoy*'s controller provides the '228 patent's "coherency logic" that manages coherency among the different cores' L3 regions to form a "coherency fabric." *Sinharoy*, 19-21.

374. Indeed, *Sinharoy* discloses a more integrated set of distributed caching agent portions than is disclosed in the '228 patent. For example, *Sinharoy*'s "lateral cast-out" operation moves a cache line from one L3 portion to another L3 portion – but still preserves the cache line within the accessible L3 cache. *Sinharoy*, 19-20.

When *a lateral cast-out from an L3 region finds a copy of the line (in a subset of the valid states) in any of the other L3 regions within the lateral cast-out broadcast scope, the states of the two copies are merged at the destination, and no data is moved*, saving energy (similar to the L2 cast-out case). Otherwise, *a set of heuristics weigh the usage pattern (both capacity usage and traffic volume) of the targeted region to determine whether to accept the lateral cast-out*. If the lateral cast-out is *accepted by the targeted L3 region* and needs to allocate a line (i.e., it does not find an invalid slot), it also preferentially selects the LRU line from the second class, only displacing a first-class line if no second-class lines exist. In order to

allow for growth in the set of candidate second-class lines, whenever a lateral cast-out is accepted by a given L3 region, the LRU first-class line begins a transformation that will eventually convert it to a most recently used second class, thereby gradually enlarging the supply of second-class lines. *Lateral cast-out operations that move data to a targeted L3 region are managed in the L3 region by a pool of ten L3 write machines and ten associated L3 cast-out machines.*

When a line is evicted from an L3 region (due to an L2 cast-out, a lateral cast-out from another L3 region, a prefetch operation, or a cache injection operation), if the evicted line is second class and has a “dirty” state, it is written back to memory. If the evicted line is first class, *a set of heuristics weigh the cache state and the usage pattern (capacity) of the evicting cache to determine whether to laterally cast-out the line to another L3 region or to immediately write it back to memory* (if it is “dirty”). *If the line is laterally cast-out, another set of heuristics determine which of the other seven L3 regions to target based upon past history, reference pattern, and system firmware control. L3 cast-out operations are managed in the L3 region by a pool of ten L3 cast-out machines..*

375. *Sinharoy*, 20-21.

376. By using the other distributed portions of the L3 caches as “victim caches” (for lateral cast-out between L3 distributed portions), *Sinharoy*’s system manages the overall cache space as a single cache managed by a single caching agent controlling the cache lines across all of the distributed portions..

377. *Sinharoy*’s L3 cache controllers (caching agent portions) act collectively through their “shared” logic.

As illustrated in Figure 10, ***the capacity, data flow logic, and control flow logic of a 4-MB local L3 region are shared between the L2 cache to which the local L3 region is coupled, and the other seven 4-MB L3 regions on the chip.***

378. *Sinharoy*, p. 19, Fig. 10. This “sharing” of the “data flow logic” and the “control flow logic” corresponds to the “coherency logic” that the ‘228 patent describes for its Fig. 2 as distributed across different physical cores in the ‘228 patent’s Fig. 6.

379. *Sinharoy* also discloses that the multicore processor utilizes a “nonblocking-broadcast-based coherence-transport mechanism” providing distributed cache coherency management based on the “13-

cache-state protocol” shown in Table 2. (*Sinharoy*, 17, 23, Tbl. 2.) This “coherence-transport” mechanism uses the L3 dispatch ports and the SMP interconnect to manage the cache. (*Sinharoy*, 19.) *Sinharoy* discloses that when a core’s memory access misses both its L2 cache and its 4-MB local L3 cache region, snoop messages are broadcasted throughout this distributed coherence fabric to the other cache regions, and the snooped cache regions send snoop responses and, if needed, forward the shared data to the requester. (*Sinharoy*, 19.) This is a further form of the “coherency logic” that the ‘228 patent references for the “single caching agent.” (Ex-1001, 2:58-64.) (quoted above)

380. *Sinharoy*’s snooping between L3 cache portions facilitates distributed coherency operations and reduces the overall snooping – just like the ‘228 patent. For example, the L3 snooping allows for the cache to appear as a single entity to all of the CPUs and the I/O controllers where a request is routed to the distributed cache agent logic that also responds to that request and ensures coherency based on that request. *Sinharoy* 23 (fully quoted previously in paragraph 348) (“The on-chip coherence interconnect routes ... *coherence requests* through the horizontal trunk, inward *toward the even/odd arbitration logic located at the center of the*

chip. Once granted, the requests are broadcast within the chip on the even/odd snoop buses outward toward the left and right edges of the chip. Requests that have arrived from other chips are managed by the even/odd arbitration logic and broadcast to the snoop buses....Coherence responses from the snoopers are routed inward along the horizontal trunk toward the even/odd coherence decision logic located at the center of the chip...”)

381. Thus, *Sinharoy* operates just like the ‘228 patent in reducing snooping, for example, by the I/O component. (Ex-1001, 1:19-30).

As process nodes advance, greater amounts of functionality can be incorporated into a single semiconductor die. *One integration trend is to integrate an IO component* such as functionality of an IOH *into a central processing unit (CPU) die*. The main motivation for doing so is to reduce the bill of material (BOM) cost of a computer system, and enable small form factors while reducing overall power consumption. But problems arise once an IO component is integrated on the same chip with a multiprocessor. *Traditional IO integration treats the IO component as a separate caching agent*, meaning that *dedicated logic is associated with the IO component to handle cache coherency*

operations. When an IO agent is performing read/write operations to main memory, it has to snoop the CPU side cache to maintain cache coherency. In MP systems, this becomes a major scaling problem. For example, in an 8 socket system, there are effectively 16 caching agents in the system, which can degrade performance. And the efforts to scale up a system to support these many caching agents are not trivial.

382. ‘228 patent, 1:12-30.

383. In the ‘228 patent’s Fig. 6 shows that the cache is managed by a single distributed entity including for the IIO unit which does not have its own separate caching agent. The distributed caching agent’s “coherency logic” manages the cache for that access internally. However, to the IIO module, the cache appears as a single entity and the IIO module does not need to separately manage/access each core’s cache using a separate I/O agent for each core – which is the problem identified in the ‘228 patent’s Background quoted above.

384. As quoted previously from Sinharoy above, *Sinharoy* operates the same way that the ‘228 patent operates. *Sinharoy*’s I/O units (the two I/O controllers at bottom and the SMP interconnect at top) treat the cache as

a single integrated unit. *Sinharoy*'s distributed L3 caching agent logic manages the I/O units' access to the cache without needing its own caching agent/logic. For example, as quoted above in detail, *Sinharoy*'s L3 cache may need to perform lateral cast-out operations or other coherency logic operations, but those operations are abstracted from the I/O unit. As with the '228 patent, *Sinharoy*'s I/O devices do not need separate I/O agents to access cache associated with each core whether these requests are coming from the I/O controllers or the SMP links. The coherency is maintained by the same logic (see discussion and quotes above). In this manner, *Sinharoy* (*Sinharoy* 23, quoted above) operates in the same manner as the '228 patent identifies for a single caching agent. Ex-1001, 1:12-30 (quoted above).

385. In IPR2023-01344, the patent owner made two arguments against *Sinharoy* regarding this same claim term ("single caching agent ... including ... distributed portions"). *E.g.*, IPR2023-01344, paper 10 (05/28/2024) at 15-22. See also IPR2023-01344 paper 20 at 4-11 (10/22/2024) (Ex-1021).

386. First, PO argued that *Sinharoy*'s reference to "snooping" among L3 caches prevents *Sinharoy* from disclosing the '228 patents "single

caching agent.” For example, the patent owner stated: “This need for snooping of cache misses by all of the individual cache controllers of other cores’ respective L2 and L3 cache controllers indicates that no one single cache controller is or is acting as a single—that is, a solitary—caching agent for the processor, as recited by claim 1.” IPR2023-01344, paper 10 (05/28/2024) at 17.

387. The presence of “snooping,” however, cannot exclude *Sinharoy*’s architecture from disclosing the “single caching agent” because the ‘228 patent expressly acknowledges that snooping will still occur with the distributed cache agents / controllers. The ‘228 patent states that its “distributed” architecture “*reduces* the amount of snoop traffic....”

Note that although a single structure is shown in FIG. 2 for ease of illustration, understand 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.***

With this approach, the ***IIO module proxies through the CPU caching agent to access memory or other IO devices***, therefore reducing the overhead of allocating dedicated resources for an integrated IO caching agent. This also reduces the amount of snoop

traffic needed since a reduced number of caching agents per system can be realized.

388. Clearly, the ‘228 patent contemplates the presence of “snooping” even with its distributed architecture. Such snooping is shown throughout the ‘228 patent. *E.g.*, Ex-1001, Fig. 4 (“receiving snoop request ... grant ownership to snoop initiator”), 4:17-23 (“snoop request”), 4:38, 4:51-54. Indeed, claim 8 (dependent on claim 1) recites a “snoop request,” (similarly claim 15 (dependent on claim 11)).

389. For example, the ‘228 patent contemplates snooping as part of the caching agent’s functionality, for example: “Various elements ***both within an IIO module and CPU caching agent*** can be configured to avoid deadlocks and ensure reasonable performance. First, for PCIe™ ordering rule compliance and in order to sustain performance, a drop ownership indication can be ***provided by a CPU caching agent*** to send an indication back to the IIO to drop ownership of a cache line in the case that an ***external snoop request to an existing ownership request is detected.***” ‘228 patent 4:15-22.

390. Another example that the ‘228 patent contemplates the presence of “snooping” with its distributed architecture: “Referring now to FIG. 4,

shown is a flow diagram of a method in accordance with another embodiment of the present invention. As shown in FIG. 4, method 375 can be used to prevent deadlocks by ***causing an IIO to drop ownership*** of a given cache line when a conflicting request is received. Specifically as shown in FIG. 4, method 375 may begin by ***determining that a snoop request is received in a caching agent*** for a cache line that is owned by an IIO module (diamond 380). This ownership can be based on state information associated with the cache line, e.g., an ownership indicator as well as cache coherency state information, in some embodiments. When ***such request is received, the caching agent can send*** a drop ownership indication to the IO module to cause the module to drop ownership of the cache line (block 385). As further shown in FIG. 4, control next passes to block 390 where the ***ownership of the cache line can be granted to the initiator of the snoop***. In this way, ***this requestor*** can perform desired operations on the data. Then the data may later be accessed by the IO device. Although shown with this particular implementation in the embodiment of FIG. 4, understand the scope of the present invention is not limited in this regard.” ‘228 patent 4:32-50.

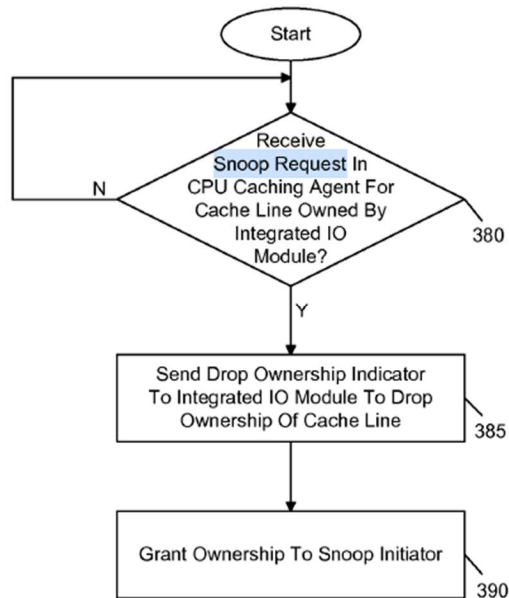
375

FIG. 4

‘228 patent FIG. 4.

391. Thus, PO’s arguments that some level of “snooping” excludes Sinharoy from meeting this claim element contradict the ‘228 patent.

392. As detailed above, Sinharoy’s lateral L3 cache reduces snooping (by I/O devices) that is identified as the problem that the ‘228 patent seeks to solve by providing the single caching agent that has distributed portions. Moreover, some amount of L3 snooping (or other coordination) is part of the logic that is necessary to communicate amongst the distributed L3

cache lines for coherency. To manage the cache coherently, the individual caching agent portions must be able to ascertain whether a particular memory address is resident in the cache or not and such a determination may be performed by snooping as part of the cache coherency logic.

393. The ‘228 patent abstracts this coordination as “various coherency logic to perform cache coherency operations on behalf of the cores and IIO module.”

Although not shown for ease of illustration, understand that caching agent 200 *may further include various coherency logic to perform cache coherency operations on behalf of the cores and IIO module, and to enable coherent access with regard to both the core transactions and IO device transactions as described herein.*

394. Ex-1001, 2:58-65. The ‘228 patent does not exclude L3 cache cross-snooping as part of the coherency logic.

395. The patent owner’s second argument in IPR2023-01344 against *Sinharoy*’s single caching agent is that the patent owner argues *Sinharoy*’s “individual cache controllers are **not** part of the same physical unit.” Ex-1021, p. 5.

396. Whether the controllers are “part of the same physical unit” is not a factor that is consistent with the ‘228 patent’s description of the distributed caching agent. As Fig. 6 shows above, in the ‘228 patent, the distributed caching agent portions are not shown as part of a the “same physical unit” as the patent owner argues. The eight separate “caching agents” in the ‘228 patent’s Fig. 6 (elements 715₀-715₇) “are *not* part of the same physical unit” (in the patent owner’s words) but are still taught in the ‘228 patent as being a single caching agent. (Ex-1001, 6:25-28.)

397. Despite not using the same words to describe its technology, *Sinharoy* shows a single caching agent as claims. For example, *Sinharoy*’s architecture implements the ‘228 patent’s “various coherency logic to perform cache coherency operations on behalf of the cores and IIO module” to allow the “cores... and an IIO module [to] maintain coherency without the need for additional caching agents.” (Ex-1001, 2:58-64) (fully quoted above).

398. Thus, *Sinharoy* discloses, or renders obvious, characterizing its architecture as a single caching agent with distributed portions as claimed.

399. The claim also recites that the caching agent is “associated with a corresponding one of the plurality of cores.”

400. As I detailed above for [1d], *Sinharoy* teaches that each core has a corresponding 4-MB L3 cache region managed by a “single centralized controller” *for that L3 cache slice* but that collectively form a single distributed “coherence fabric” (i.e., “a single caching agent”) that spans all L3 slices. (*Supra* [1d]; *Sinharoy*, 17-19, Fig. 1.)

401. *Sinharoy*’s eight processor cores each comprises, and supplies, an associated 4-MB L3 cache region for collective usage across all the cores (like the ‘228 patent Fig. 6 above). *E.g.*, *Sinharoy*, 17 (“32-MN L3 cache comprised of the 4-MN local L3 regions from the eight cores”); 16 (“a 32-KB L1 D-cache, a 256-KB L2 cache, and a 4-MB local L3 region **in** the POWER7 processor”),²³ (“Each core’s associated L2 cache and local L3 region...”); 22 (“POWER7 chip contains eight cores, each with an associated L2 cache and local L3 region...”).

402. *Sinharoy*’s Fig. 1 shows each core contains this L3 cache as described in *Sinharoy*’s narrative description:

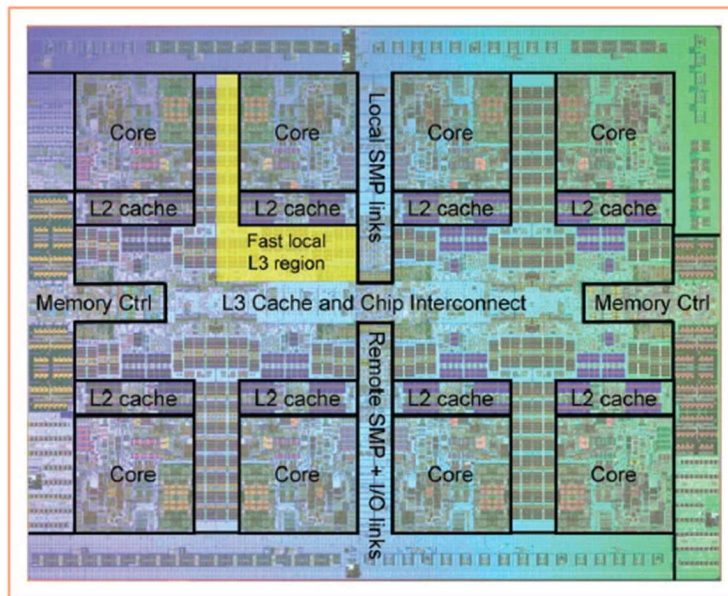


Figure 1

Die photo of the IBM POWER7 chip.

403. (*Sinharoy*, Fig. 1.).

404. For my analysis, it is not necessarily a physical proximity between *Sinharoy*'s core and L3 region on the chip as shown above that establishes the "associated with" element. Nor is the mere ability to access the shared cache but rather *Sinharoy*'s statements that each core includes / incorporates the L3 cache local region because, by definition, a "shared" cache is accessible to, and coupled to, all of the cores sharing that cache. Rather, it is *Sinharoy*'s statements above identifying the specific relationship between each of the 4-MB portions and their associated cores that establishes the claimed association.

405. Accordingly, each local 4-MB L3 cache region is “associated with” a corresponding core.

406. At a minimum, given *Sinharoy*’s overlap in architecture and functionality with that described in the ‘228 patent as being a “single caching agent,” *Sinharoy* renders obvious the claimed single caching agent. The shared L3 cache management is distributed across the different cores with “data flow logic and control logic” being “shared” between the L3 controller associated with each cache. This distributed sharing of the logic renders obvious the “single caching agent.”

407. *Sinharoy* thus discloses, including rendering obvious, this element.

j) Claim 4: “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.”

408. *Sinharoy* discloses, or renders obvious, this claim.

409. A POSITA would have understood that “a last level cache (LLC),” as claimed, refers to the higs data directly via a direct memory access

(DMA) write operation into the local L3 cache reghest level cache shared by all the cores.

410. *Sinharoy* discloses, as part of the “shared cache memory” discussed for [1b] above, a shared 32-MB L3 cache that comprises eight local regions (the 4-MB L3 regions) distributed across the die among the eight cores in the processor. (*Sinharoy*, 17, 19, Fig. 1, Tbl. 1.) The L3 cache is “a last level cache (LLC),” as claimed.

Each of the 4-MB L3 regions (“a plurality of distributed portions” of the “last level cache”) is associated with an L2 cache and an associated core (“corresponding one of the plurality of cores”). The yellow highlighting in Figure 1 below (included in the original) indicates the local L3 region is associated with its neighboring core, and each of the other seven cores similarly has an associated “local L3 region” neighboring it. Therefore, the POWER7 cache hierarchy ***includes a shared 32-MB L3 cache comprised of the 4-MB local L3 regions*** from the eight cores that reside on the processor chip.

411. *Sinharoy*, 17.

“The ***shared 32-MB POWER7 L3 cache is composed of eight 4-MB L3 regions***. Each L3 region utilizes 128-byte lines and is eight-way

set associative. A given L3 region comprises a single centralized controller, with four address hashed eDRAM cache data banks and four address hashed SRAM directory banks. *The local L3 region is tightly coupled to the L2 cache associated with a given core.*

412. (Sinharoy, 19.)

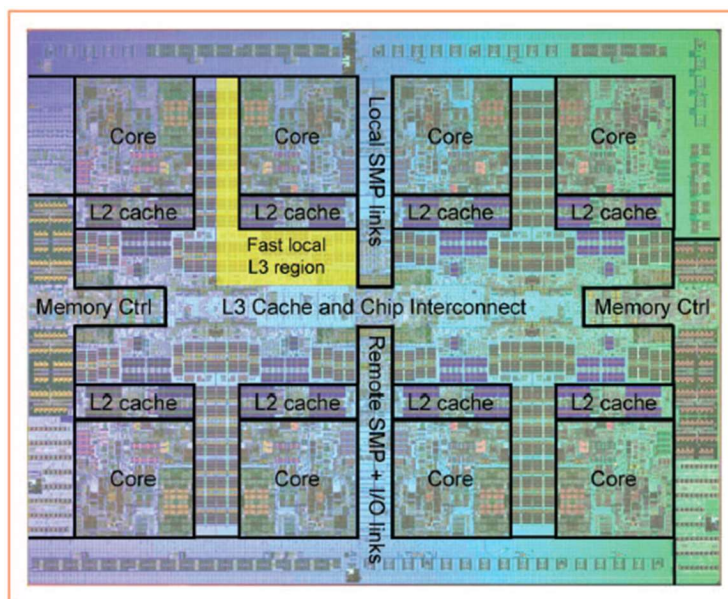


Figure 1

Die photo of the IBM POWER7 chip.

413. (Sinharoy, Fig. 1.) According to Sinharoy, the 4-MB L3 region and the coupled L2 share the same data and control flow logic, and the “associated L2 cache utilizes the 4-MB local L3 region directly as a *victim cache*.” (Sinharoy, 16-17, 19, Fig. 1.)

414. Further evidence of such an association is the fact that the each of the 4-MB local L3 regions is on the same power domain as its associated core and L2 cache: “Each POWER7 chiplet (i.e., a ***POWER7 core with its Level 2 (L2) and local L3 cache***) is *designed to be on a separate power domain, with asynchronous boundary with the PowerBus to which it is connected. This allows each chiplet to have independent voltage and frequency slewing* for advanced power management.” *Sinharoy*, 4.

415. And yet additional evidence that Sinharoy’s local L3 cache is associated with its corresponding core is disclosed through the description of operation of the core: “ “If a ***given core’s L2 cache or local L3 region*** owns a copy of the targeted cache line (i.e., holds the line in an M, ME, MU, T, TE, TN, or TEN cache state), the data will be installed into the local L3 region.” *Sinharoy*, 17.

416. Thus, *Sinharoy* discloses this element.

k) Claim 6: “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.”

417. Sinharoy discloses, or renders obvious, this claim.

418. 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)].”

To this end, *an IO device coupled to an IIO module* in accordance with an embodiment of the present invention *can generate an allocating transaction that can lodge data directly into the LLC.*

In this way IO data can be brought closer to the CPU cache, providing a more efficient mechanism to *lodge data from an IO device directly into a LLC of a processor and without the need for first storing the data in a system memory*, reducing bandwidth consumption. Still further, the need for a direct cache hint from the IO device to a core to cause a read request to obtain the data can be avoided.

Referring now to FIG. 3, shown is a flow diagram of a method in accordance with an embodiment of the present invention. As shown in FIG. 3, method 300 may be used to *perform allocation of data from an IO device coupled to a multicore processor directly into a cache memory of the processor.* More specifically, method 300 of FIG. 3 can be used to insert data into the

cache in a manner that avoids both: first providing the data to a system memory coupled to the processor (such as a dynamic random access memory (DRAM)) via an integrated memory controller of the processor; and the *need for sending a hint to a core of the processor to cause the data to be requested and stored into the cache.*

As seen in FIG. 3, method 300 may begin *by receiving an allocation transaction in an IIO module* from an IO device (block 310). For example, this *allocation transaction can be a request to write data directly into a cache memory, e.g., a LLC of the processor*, and it can be received from an IO device such as a peripheral device coupled to the processor via a PCIe™ link, for example.

419. '228 patent, 3:22-51

420. *Sinharoy* discloses the feature to directly lodge (inject) data into the cache.

421. *Sinharoy* discloses that the coherence fabric (i.e., “the caching agent”) receives an allocation transaction from an I/O device via an integrated I/O controller.

422. In particular, *Sinharoy* discloses a “cache injection” operation that allows an I/O device coupled to the processor to send a “write operation” (i.e., an “allocation transaction”) that writeion. (*Sinharoy*, 17, 22.)

As shown in Table 2, the POWER7 L2 and L3 caches support the same 13-cache-state protocol as the POWER6 design point [4]. While no new cache states have been added for POWER7, *new coherent operations are supported. One such operation is cache injection. An I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory.* If a given core’s L2 cache or local L3 region owns a copy of the targeted cache line (i.e., holds the line in an M, ME, U, T, TE, TN, or TEN cache state), *the data will be installed into the local L3 region.*

423. *Sinharoy*, 17.

424. Cache injection is a DMA/Write machines operation that applies to multiple modules including the I/O controllers and SMP links. For example, “*each POWER7 chip contains several coherence request, snoop, response, and notification interfaces and several data interfaces*” *Sinharoy*, 17.

425. “Storage accesses that miss the L2 cache access the 4-MB local L3 region via the core/L2 dispatch port. Those that hit in the 4-MB local L3 region are managed by a pool of eight read machines. Prefetches and some L2 store misses also access the L3 region via the core/L2 dispatch port and are managed by a pool of 24 lightweight RC machines. When prefetched data is staged to the L3 cache, it is *managed by a pool of ten write machines*. Each *write machine has an associated cast-out machine* to manage the eviction of lines displaced by writes. Note that *these write machines are also utilized by other operations* such as cast-outs and *cache-injections*.”

426. Sinharoy, 19. “An I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory.” *Sinharoy*, 17.

427. As discussed for [1c], integrated I/O controllers connect (i.e., “interface between”) one or more input/output devices (e.g., PCI device) and the multicore processor via an I/O hub. Accordingly, *Sinharoy*’s “coherence fabric” (i.e., the “caching agent”) receives an “allocation transaction” (e.g., write request) from one of the two integrated I/O

controllers (“IIO module”) when an I/O device seeks to perform a cache injection.

428. Similarly, the SMP network interconnect system (see discussion above in [1c], [1d], [1e]) is used “to facilitate coherent I/O sharing among the eight flat 32-way SMP” nodes to provide cache coherence. Sinharoy, 25.

429. Thus, Sinharoy discloses this claim.

- 1) Claim 7: “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.”

430. Sinharoy discloses, or renders obvious, this claim.

431. As detailed above for [1c], [1d], [1e], claim 6, *Sinharoy*’s L2 and L3 caches support a “13-cache-state protocol” that enable “new coherent operations” such as “cache injection.” (*Sinharoy*, 17.) During a cache injection, a DMA write operation (from any of the CPUs or IIO) may target a cache instead of the memory, (e.g., “[a]n I/O device performing a

direct memory access (DMA) write operation may target the operation to the cache, instead of to memory.” (*Sinharoy, 17; see also Sinharoy, 19, 22.*))

432. Sinharoy’s disclosure of “instead of to memory” confirms that this cache injection does not initiate a memory transaction. Sinharoy’s use of the words “instead of” and a POSITA would understand such language to disclose, including rendering obvious, that there is not a memory transaction associated with the cache injection.

433. Thus, Sinharoy discloses this claim.

m) Claim 11:

a. Preamble: “11. A method comprising:”

434. I have been advised that preamble can be limiting in certain circumstance but I do not need to make any opinions herein whether this preamble is limiting because Sinharoy discloses the claimed method in the operation of its disclosed processor as detailed below.

b. [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.”

435. *Sinharoy* discloses this element.

436. *Sinharoy* discloses a “multicore processor” as detailed above for claim
element 1[a] above. (*Supra* claim element 1[a].)

437. *Sinharoy* discloses “an integrated input/output (IIO) module of a
multicore processor” as detailed above for claim element 1[c]. (*Supra*
claim 1[c].)

438. *Sinharoy* discloses “a shared cache memory of the multicore
processor” as detailed above for claim element 1[b]. (*Supra* claim 1[b].)

439. *Sinharoy* discloses “receiving an allocation transaction in an
integrated input/output (IIO) module of a multicore processor from an IO
device coupled to the multicore processor” as detailed above for claim 6.
(*Supra* claim 6.)

440. Furthermore, as discussed for claims 6 and 7, *Sinharoy*’s cache
injection write operation (i.e., “the allocation transaction comprising a

request to directly store data of the allocation transaction into a shared cache memory of the multicore processor”) writes data directly via a direct memory access (DMA) write operation into the local L3 cache region as a result of a request from an IO device. (*Supra* claims 6-7)

- c. [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”

441. *Sinharoy* discloses this element.

442. *Sinharoy* discloses “a caching agent of the multicore processor” as detailed above for claim element 1[d]. (*Supra* claim 1[d].)

443. Furthermore *Sinharoy* discloses “the caching agent shared by a plurality of cores of the multicore processor and the IIO module” as detailed above for claim element 1[e]. (*Supra* claim 1[e].)

444. As discussed for [1d] and [1e], *Sinharoy* discloses a “shared” single caching agent with distributed portions that maintain cache coherency for the plurality of cores and IIO module. (*Supra* [1d], [1e]).

445. *Sinharoy* discloses “forwarding the allocation transaction to a caching agent of the multicore processor” as detailed above for claim 6. (*Supra* claim 6.)

446. *Sinharoy*’s coherence fabric (i.e., “caching agent”) receives a cache injection write operation (i.e., “allocation transaction”), which has been forwarded by an integrated I/O controller (i.e., “IIO module”) or SMP link to the coherence fabric so it can be received. (*Supra* claims 6, 7). For example, *Sinharoy* disclose “An *I/O device* performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory.” (*Sinharoy*, 17). A POSITA would understand that in order to “target” an operation, an I/O device performs the “forwarding” of the operation (i.e. “transaction”) or that it would be obvious to perform such operation. This understanding is further confirmed from the ‘228 patent (e.g. “As seen, this allocation *transaction* can be forwarded from the IIO to a CPU caching agent (block 320). Because in various embodiments *there is no separate caching agent for the IIO*, this *transaction can be directed to a common caching agent* that performs caching agent functionality both for processor cores and the IIO.” ‘228 patent 3:51-56).

447. *Sinharoy* thus discloses this element.

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

448. *Sinharoy* discloses this element.

449. *Sinharoy* discloses “directly storing the data of the allocation
transaction into the shared cache memory of the multicore processor” as
detailed above for claim 7. (*Supra* claim 7.) For example, *Sinharoy*’s
“POWER7 I/O controllers also support *cache injection*, which enables
*DMA write traffic to be written directly into a cache, instead of to
memory.*” (*Sinharoy*, 22.)

n) Claim 12: “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.”

450. Sinharoy discloses, or renders obvious, this claim. (See paragraphs 264-265 above, included herein by reference).

451. As discussed for claims 7 and 11, *Sinharoy*'s "cache injection" operation allows an I/O device to write data directly into the local L3 cache region instead of to memory. A POSITA would understand that Sinharoy supports a well-known write-back procedure for caches and therefore discloses access (including both reading and writing) data as many times as needed until a write-back needed (i.e. line has a "[dirty] state (i.e., "modified data," as claimed in claim 13)) for writing copies of cache lines stored as shared L3 cache lines causes the cache line to be written to main memory. (*Supra* claims 7, 11; *Sinharoy*, 17-21.) (e.g., "***Storage accesses that hit*** in the 4-MB local L3 region are ***managed by a pool of eight read machines....***" "When prefetched data is staged to the L3 cache, it is ***managed by a pool of ten write machines.*** Note that these ***write machines are also utilized by other operations*** such as cast-outs and ***cache-injections***" "the ***POWER7 L2 and L3 caches support the same 13-cache-state protocol*** as the POWER6 design point [4]. While no new cache states have been added for POWER7, ***new coherent operations are supported. One such operation is cache***

injection..... performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory,”

“Therefore, the following 128-byte operations all utilize a given cache data array for two 2 : 1 cycles: 1) core fetches; 2) cast-out reads; 3) snoop intervention reads, and 4) *write-backs of miss data*, as well as read–modify–write operations (performed on behalf of accumulated core stores) that affect more than 64-byte aligned values,” “if the evicted line is second class and *has a [dirty] state, it is written back to memory. If the evicted line is first class*, a set of heuristics weigh the cache state and the usage pattern (capacity) of the evicting cache to determine whether to laterally cast-out the line to another L3 region *or to immediately write it back to memory (if it is [dirty])*. If the line is laterally cast-out, another set of heuristics determine which of the other seven L3 regions to target based upon past history, reference pattern, and system firmware control. L3 cast-out operations are managed in the L3 region by a pool of ten L3 cast-out machines.”)

452. Once data is in *Sinharoy*’s shared cache, that data can be accessed without accessing the system memory – that is a cache’s purpose. The ‘228 patent does not describe anything new related to this claim and

merely states “[w]ith the data now present in the LLC, it can be accessed a number of times for read and write operations by any of the cores of the processor” (Ex-1001, 3:65-67.)

453. *Sinharoy* teaches that when a core needs to access a particular data location, the core checks first in its L2 cache, then in its local L3 cache, and then in the “remaining 28 MB of the on-chip L3 cache” – all without accessing system memory. *Sinharoy*, 19, Fig. 10. Thus, *Sinharoy* meets this claim.

454. A core that wants to access the I/O data resident in *Sinharoy*’s shared L3 cache will access that data without sending a transaction to system memory for as many accesses as the data remains in the cache.

- a. Claim 13: “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”

455. *Sinharoy* discloses, or renders obvious, this element.

456. In Sinharoy, “When *a line is evicted from an L3 region* (due to an L2 cast-out, a lateral cast-out from another L3 region, a prefetch operation, or a cache injection operation), if the evicted line is second class and *has a ‘dirty’ state*, it is *written back to memory*.” Sinharoy, at 21. Sinharoy’s Table 2 shows that “dirty” (i.e. “modified”) data can be “cast out” to memory. Sinharoy’s Figure 10 shows the cast-out (evicted / modified) lines being written to system memory:

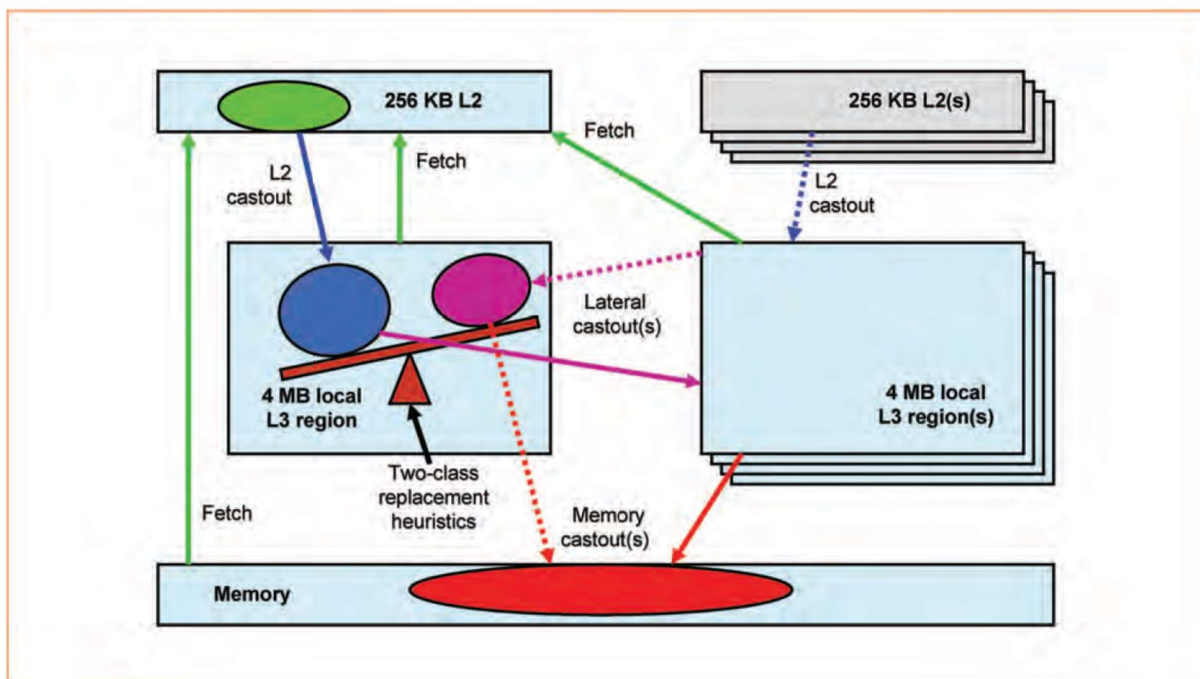


Figure 10

L3 region behavior.

457. Sinharoy, Fig. 10. Sinharoy’s “cast-out scenarios are described in detail” on pages 19-21. Sinharoy, 19-21. Those scenarios include

writing the evicted cast-out line to memory if it is dirty (modified).

Sinharoy, 19-21 (see quotes and discussion for claim 12, *Supra* Claim 12 above).

- b. Claim 14: “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.”

458. Sinharoy discloses, or renders obvious, this claim.

459. *Sinharoy* discloses claim 14 for the same reasons as cited above for claims 7, 11 and 12. (*Supra* [11], [12]). As discussed therein, *Sinharoy*’s cache injection directly stored data into the cache instead of the memory.

**D. Ground 2B: Claims 2, 3, and 16-18 Are Obvious Over Sinharoy
Combined With Blumrich.**

5. Overview of Blumrich

460. I have included an overview of Blumrich above in Section IX.

6. Motivation to Combine Sinharoy and Blumrich

461. One of ordinary skill would have been motivated to implement *Sinharoy*'s "coherence fabric" using *Blumrich*'s request queues. Like *Sinharoy*, *Blumrich*'s disclosure originated at IBM and a POSITA would be motivated to consider the combination of related technologies disclosed by the same company. (*Blumrich*, Cover.)

462. Each *Sinharoy* controller provides a "pool of operational resources." (*Sinharoy*, 19.) Accordingly, all incoming requests to an L3 controller must be arbitrated and processed using this pool of resources. *Sinharoy*'s pool of resources includes "a pool of eight read machines," "a pool of 24 lightweight RC [read/claim] machines," and "a pool of ten write machines." (*Sinharoy*, 19.) Read machines manage read hits; RC machines manage store misses and prefetches; and write machines manage data writes. (*Sinharoy*, 19.) Write machines also perform core prefetch operations, L2 cast-out operations, and I/O cache injection operations. (*Sinharoy*, 17-20.)

“The centralized L3 region controller provides a single core/L2 dispatch port, a single lateral L3 region dispatch port, dual snoop dispatch ports (for even and odd cache lines), and *a single pool of operational resources*. Storage accesses that miss the L2 cache access the 4-MB local L3 region via the core/L2 dispatch port. Those that hit in the 4-MB local L3 region *are managed by a pool of eight read machines*. Prefetches and some L2 store misses also access the L3 region via the core/L2 dispatch port and are *managed by a pool of 24 lightweight RC machines*. When prefetched data is staged to the L3 cache, it is *managed by a pool of ten write machines*. Each write machine has an associated cast-out machine to *manage the eviction of lines displaced by writes*. Note that these write machines are also utilized by other operations such as cast-outs and cache-injections.” *Sinharoy* 19.

“The on-chip *coherence interconnect routes two sets (even and odd cache line) of coherence requests* through the horizontal trunk, *inward toward the even/odd arbitration logic located at the center of the chip*. Up to one even request and one odd request may be granted each on-chip bus cycle. Once granted, the requests are broadcast within the chip on the even/odd snoop buses outward toward the left and right edges of the chip.

Requests that will be routed to other chips are also sent to the multichip SMP interconnect (discussed in the multichip interconnect section) via a central vertical spine toward the top and bottom edges of the chip. Requests that have arrived from other chips are managed by the even/odd arbitration logic and broadcast to the snoop buses.” *Sinharoy* 23.

463. Because the pool of resources has limited entries (e.g., capable of handling eight outstanding reads, 24 outstanding store misses, and ten outstanding data writes), a POSITA would have understood the controller would beneficially use a storage element (e.g., queue or buffer) to track outstanding requests and avoid overflowing the limited pool of resources.

464. A POSITA would have understood that either a queue or a buffer would have been an obvious solution for keeping track of outstanding requests. For example, *Sinharoy* suggests using queues to keep track of requests within the core. (See *Sinharoy*, 9-10 (explaining the core includes buffers/queues to track outstanding requests, such as a 32-entry store-reorder-queue (“SRQ”) and a 32-entry load-reorder-queue (“LRQ”).)

465. A POSITA would have considered similar, known options for *Sinharoy*’s coherence fabric (“caching agent”) to track outstanding

requests, including *Blumrich*'s disclosure of a tracking queue. (First, a POSITA would have understood that a queue like the one *Blumrich* discloses would be beneficially used by *Sinharoy* for tracking requests. For example, a queue enables ordering requests, allowing the address of all incoming requests to be matched against the outstanding requests. (*Blumrich*, 0128).

466. When a write request is followed by a read request for the same address, the two requests must be executed in order; otherwise, the read will not access the correct data. A POSITA thus would have understood that a caching agent (e.g., *Sinharoy*'s coherence fabric) requires a queue to store all pending requests to perform address-checking for an incoming request. (*Blumrich*, 0128-0130).

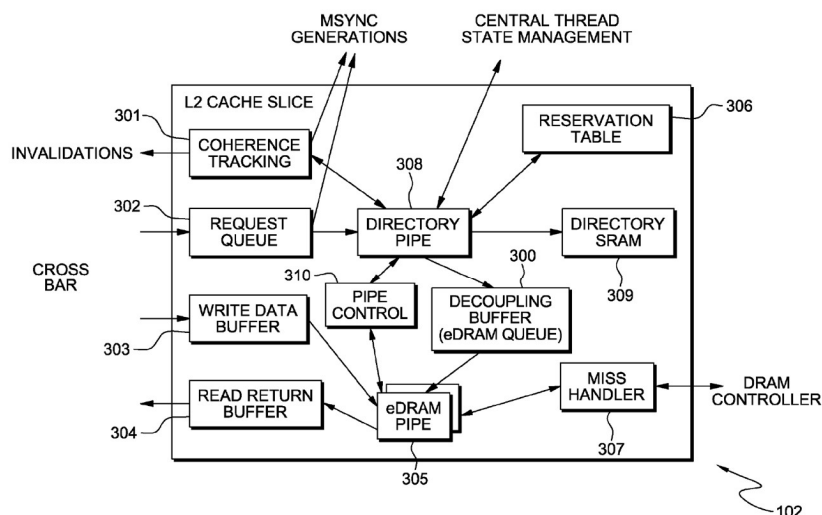


FIG. 3

Blumrich at FIG. 3.

“[0128] The *request queue 302 buffers incoming read and write requests*. In this embodiment, it is *16 entries deep*, though other request buffers might have more or less entries. The *addresses of incoming requests are matched against all pending requests to determine ordering restrictions*. The queue presents the requests to the directory pipeline 308 based on ordering requirements.

[0129] The *write data buffer 303 stores data associated with write requests*. This embodiment has a 16B wide data interface to the switch 60 and stores 16 16B wide entries. Other sizes might be devised by the skilled artisan as a matter of design choice. This buffer passes the data to the eDRAM pipeline 305 in case of a write hit or after a write miss resolution. The eDRAMs are shown at 101 in FIG. 1D.

[0130] The directory pipeline 308 accepts requests from the *request queue 302*, retrieves the corresponding directory set from the directory SRAM 309, *matches and updates the tag information*, writes the data back to the SRAM and signals the outcome of the request (hit, miss, conflict detected, etc.). Operations illustrated at FIGS. 3F, 3G, 3H, 3I-1, and 3I-2 are conducted within the directory pipeline 308.”

Blumrich, 0128-0130.

467. Additionally, a POSITA would have reasonably expected to succeed in implementing *Sinharoy*'s coherence fabric by including a request queue, as *Blumrich* teaches. The combination would have involved nothing more than combining prior art elements arranged according to known methods to yield predictable results.

7. Sinharoy Combined With Blumrich Discloses, And/ Or Renders Obvious, Claims 2, 3, and 16-18.

o) Claim 2: "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."

468. Sinharoy combined with Blumrich renders obvious, this claim.

469. Sinharoy itself discloses Claim 2. Sinharoy's caching agent (cache controllers) include RC and write machines as detailed above. Because RC machines "aggressively reorder storage operations" and manage "core fetches, read-modify-write operations, on behalf of core stores, and core prefetch operations," it would be obvious that they include temporary storage to store incoming (pending) requests to perform such

reordering, such as requests to read or to write for the cores or IIOs. (Sinharoy, 1:18). The temporary storage in the RC machines render obvious “tables of requests,” and the incoming requests to read or to write are “pending requests.”

470. Furthermore, write machines in the same “caching agent” perform write operations including “cache-injections.” (Sinharoy, 1:19.) Integrated I/O controllers (i.e., the IIO modules) also “support cache injection.” (Sinharoy 1:22.) The write operations include incoming (“pending”) requests to write for the IIO module. When performing cache injection operation, it would be obvious that the write machines include temporary storages (i.e., “table of request”) for incoming requests to write. (*Id.*) Thus, the caching agent with RC and write machines include “table[s] of requests to store pending requests for the plurality of cores and the IIO module.” (*Id.*)

471. *Sinharoy* also discloses this feature in its pool of operational resources and temporary storages discussed above. Each L3 cache controller provides a “pool of operational resources.” (*Sinharoy*, 19.) Because the pool of resources has limited entries (e.g., capable of handling eight outstanding reads, 24 outstanding store misses, and ten outstanding data

writes), the controller would beneficially use a storage element (e.g., queue or buffer) to track outstanding requests and avoid overflowing the limited pool of resources.

472. Furthermore, Sinharoy discloses the use of queues for interaction with the cache as shown in Figure 3 duplicated below, where, for example, “Load reorder queue,” “Store reorder queue,” and “Store data queue,” are used to interface to the L1 and L2 caches. Also, Sinharoy discloses “The L2 RC machines and L3 lightweight RC machines *aggressively reorder storage operations* in order to fully exploit the relaxed rules prescribed by the PowerPC storage architecture, thereby enabling the high volume of system coherence traffic needed to scale robustly to a large number of sockets.” Sinaroy 18-19.

473. Therefore Sinharoy renders obvious “the caching agent” (e.g. the RC machines) “includes a table of requests to store pending requests for the plurality of cores and the IIO module” (e.g. aggressively reorders storage operations via, for example, a “Store reorder queue” and a “Store data queue”). The ‘228 patent explains that a table of requests includes buffers, temporary storage or queues (“As seen, **caching agent 200** can include an ingress port 210 that includes **multiple ingress queues 211-**

213. As seen, ingress port 210 may be coupled to an arbiter 215 which can arbitrate amongst incoming requests and provide them to *a table of requests (TOR) 220*. As seen, *TOR 220 may be a buffer or other temporary storage* for holding incoming requests. In the embodiment shown, *TOR 220 may include at least one dedicated resource 225 for posted transactions*. In one embodiment, *dedicated resource 225 may be a single entry of the queue*, although the scope of the present invention is not limited in this regard.” ‘228 Patent 2:43-53)

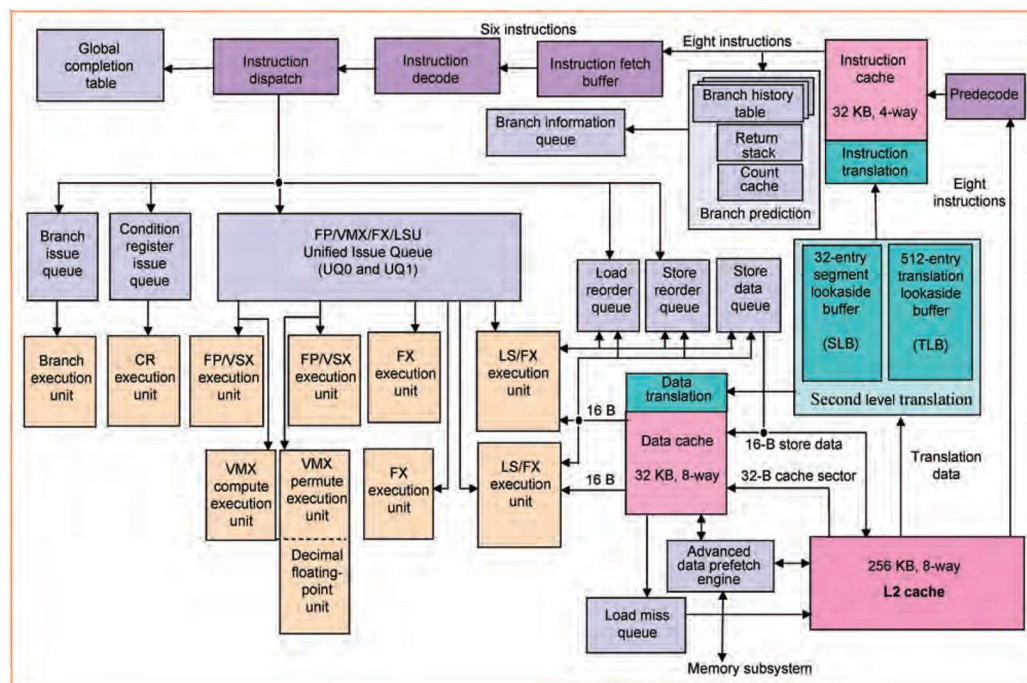


Figure 3

IBM POWER7 processor core pipeline flow.

Sinharoy at Figure 3.

474. If *Sinharoy* alone does not render claim 2 obvious, the *Sinharoy-Blumrich* combination renders claim 2 obvious. As discussed above *Sinharoy* discloses the use of “pools” and reordering in the write machines, a POSITA would have understood that either a queue or a buffer would have been an obvious solution for keeping track of outstanding requests. For example, *Blumrich* teaches to use a “request queue 302” to “buffer[] incoming read and write requests.” (*Blumrich*, ¶[0128]; see also *id.* at Fig. 3.) *Blumrich*’s request queue allows “[t]he addresses of incoming requests [to be] matched against all pending requests to determine ordering restrictions.” (*Id.*)

475. Thus, a POSITA at least would have found a queue like the one *Blumrich* discloses to be an obvious implementation of *Sinharoy* and would have reasonably expected to succeed in implementing such a solution.

p) Claim 3: “3. The apparatus of claim 2, wherein the table of requests includes at least one dedicated entry to store a posted transaction.”

476. *Sinharoy-Blumrich* discloses Claim 3. As stated above for claim 3, the RC and write machines within *Sinharoy*'s caching agent include temporary storages (i.e., tables of request) to store incoming requests (e.g., requests to write or read). (*See supra* claim 3.)

477. When write machines perform cache injection operations, they involve writing a cache line into a local L3 region. (*Sinharoy*, 1:17, 1:19.) Such a write transaction is a “posted transaction” because (as identified in the ‘228 patent) *Sinharoy* does not disclose any “completion or confirmation message regarding the” injection being provided to the I/O device that is the injection source. (Ex-1001, 5:3-7)

478. Because write machines within *Sinharoy*'s caching agent have temporary storage, it would be obvious to include dedicated queues or resources (i.e., dedicated entry) to store such a write transaction (i.e., posted transaction) when performing the cache injection.

479. Thus, if *Sinharoy* alone does not render claim 3 obvious, the *Sinharoy-Blumrich* combination renders claim 3 obvious.

a. Claim 16:

i. 16[pre] “A system comprising:”

480. I have been advised that preamble can be limiting in certain circumstance but I do not need to make any opinions herein whether this preamble is limiting because Sinharoy discloses the claimed system as detailed in the elements below.

481. For example, *Sinharoy*'s "up to 32 POWER7 chips may be connected to form a single 256-way SMP system." (*Sinharoy*, 23; *see also id.*, 23-25, Figs. 11-13.) Figure 13 shows an exemplary system with four multicore processors:

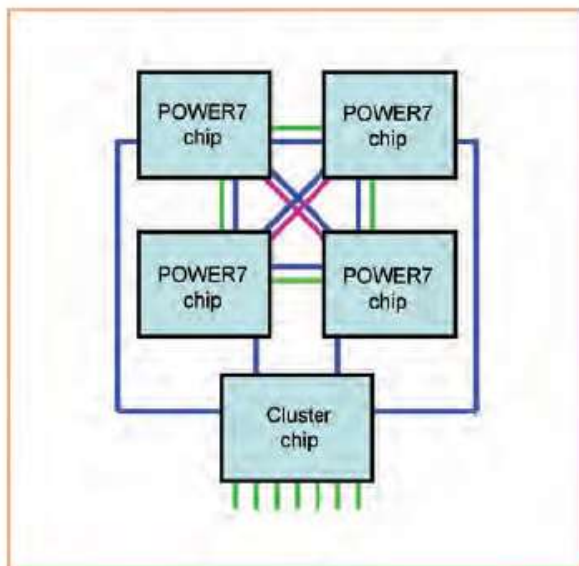


Figure 13

Cluster system first-level coherent nodal topology.

482. *Sinharoy*, Fig. 13.

ii. 16[a] “a first multicore processor
including a first plurality of cores,”

iii. 16[b] “a first shared cache memory;”

483. *Sinharoy* discloses, or renders obvious, these elements.

484. *Sinharoy* discloses 16[a] and 16[b] for the reasons cited for [1a], [1b].
(*Supra* [1a], [1b]).

iv. 16[c] “a first integrated input/output (IIO)
module to interface between the first
multicore processor and a first plurality of
IO devices coupled to the first multicore
processor, and;”

485. *Sinharoy* discloses, or renders obvious, this element for the reasons
discussed in [1c].

486. *Sinharoy*’s server system includes up to 32 multicore server
processors. (*Supra* [16pre].) Each server processor includes an integrated
I/O controller (i.e., “an integrated input/output (IIO) module”) to
interface between the multicore processor and a plurality of IO devices

coupled to the multicore processor. (*Supra* [1c].) The evidence provided for [1c] thus demonstrates that *Sinharoy* discloses the features of 16[c].

- v. 16[d] “a first caching agent to perform
cache coherency operations for the first
plurality of cores and the first IIO
module;.”

487. *Sinharoy* discloses, or renders obvious, this element.

488. As discussed for [1d] and [1e], each of *Sinharoy*’s server processors includes a coherence fabric (i.e., “caching agent”) to perform cache coherency operations for the plurality of cores and the IIO module thus disclosing this element. (*Supra* [1d], [1e]).

- vi. 16[e] “wherein the first caching agent
includes a table of requests having a
plurality of entries each to store a pending
request for one of the first plurality of
cores or the first IIO module, including at
least one dedicated entry to store a pending

request corresponding to a posted
transaction:”

489. Sinharoy combined with Blumrich discloses, or renders obvious, this
element.

490. As discussed for [16pre], Sinharoy discloses a server system with up
to 32 multicore server processors. (*Supra* [16pre].) As discussed for
claim 2, for each of these multicore processors, the coherence fabric (i.e.,
“caching agent”) includes a table of requests to store pending requests for
the plurality of cores and the IIO module. (*Supra* claim 2.)

491. As discussed for claim 3, the table of requests includes at least one
dedicated entry to store write requests (corresponding to a posted
transaction) for a cache injection. (*Supra* claim 3.) The evidence provided
for claims 2 and 3 thus demonstrates that Sinharoy discloses the features
of 16[e].

vii. 16[f] “a second multicore processor
including a second plurality of cores,”

492. Sinharoy discloses, or renders obvious, this element.

493. As discussed for [16pre], *Sinharoy* discloses a server system with up to 32 multicore server processors. (*Supra* [16pre].) This would include at least a first and a second multicore processor. As discussed for [1a], each of those server processors includes a plurality of cores. (*Supra* [1a].)

viii. 16[g] “a second shared cache memory,”

494. *Sinharoy* discloses, or renders obvious, this element.

495. As discussed for [16pre], *Sinharoy* discloses a server system with up to 32 multicore server processors. (*Supra* [16pre].) As discussed for [1b], each of those server processors, including a second multicore server processor, includes a “32-MB-shared-L3 cache” (i.e., “shared cache memory”) within processor. (*Supra* [1b].) The evidence provided for [1b] thus demonstrates that *Sinharoy* discloses the features of 16[g].

ix. 16[h] “a second IIO module to interface
between the second multicore processor
and a second plurality of IO devices
coupled to the second multicore processor,
and”

496. *Sinharoy* discloses, or renders obvious, this element.

497. As discussed for [16pre], Sinharoy discloses a server system with up to 32 multicore server processors. (Supra [16pre].) As discussed for [1c], each of those server processors, including a second multicore server processor, includes an integrated I/O controller (i.e., “an integrated input/output (IIO) module”) to interface between the second multicore processor and a plurality of IO devices coupled to the second multicore processor. (Supra [1c].) The evidence provided for [1c] thus demonstrates that Sinharoy discloses the features of 16[h].

- x. 16[i] “a second caching agent to perform
cache coherency operations for the second
plurality of cores and the second IIO
module; and”

498. Sinharoy discloses this element.

499. As discussed for [16pre], Sinharoy discloses a server system with up to 32 multicore server processors. (Supra [16pre].) As discussed for [1d], each of those server processors, including a second multicore server processor, includes a coherence fabric (i.e., “caching agent”) to perform cache coherency operations for the plurality of cores and the IIO module.

(Supra [1d].) The evidence provided for [1d] thus demonstrates that
Sinharoy discloses the features of 16[i].

500.

- xi. 16[j] “a peripheral controller coupled to at
least one of the first and second multicore
processors.”

501. Sinharoy discloses, or renders obvious, this element.

502. As discussed for [16pre], *Sinharoy* discloses a server system with up
to 32 multicore server processors. (*Supra* [16pre].) In one example, as
part of that server system, *Sinharoy* further discloses a “cluster
interconnect chip,” which incorporates up to three PCI express interfaces
and provides a connection between the least two multicore server
processors and any PCI devices that connect to the PCI express
interfaces:

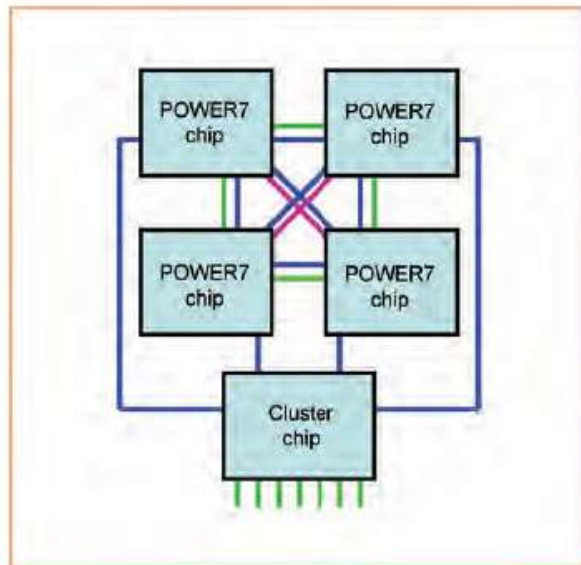


Figure 13

Cluster system first-level coherent nodal topology.

503.

504. (*Sinharoy*, 25, Fig. 13)

505. This cluster interconnect chip is a “peripheral controller,” as claimed, because it provides a connection between at least one multicore processor and peripheral components.

506. As shown above, the cluster interconnect chip is “coupled to at least one of the first and second multicore processors.”

b. Claim 17: “17. The system of claim 16, wherein the first caching agent is to receive an allocation transaction from the first IIO module, the allocation transaction comprising a request to

directly store data of the allocation transaction
into the first shared cache memory, and directly
store the data of the allocation transaction into
the first shared cache memory.”

507. Sinharoy discloses, or renders obvious, this claim for the reasons
discussed in claims 6 and 11 above.

508. Sinharoy discloses “wherein the [first] caching agent [of the first
multicore processor] is to receive an allocation transaction from the IIO
module.” (*Supra* claim 6.) Sinharoy discloses that “the allocation
transaction compris[es] a request to directly store data of the allocation
transaction into a [first] shared cache memory.” (*Supra* [11a].) Sinharoy
discloses “directly storing the data of the allocation transaction into the
[first] shared cache memory of the [first] multicore processor.” (*Supra*
[11c].).

c. Claim 18: “18. The system of claim 17, wherein
the data in the shared cache memory is to be
accessed via at least one of the first plurality of
cores a number of times, without sending a

transaction to a system memory coupled to the
first multicore processor.”

509. Sinharoy discloses, or renders obvious, this claim for the reasons
discussed for claims 7 and 12 above.

**E. Ground 2C: Claim 5 Is Obvious Over Sinharoy Combined With
Sistra.**

1. Overview of Sistra

510. I have included an overview of Sistra in Section IX above.

**2. Motivation to Combine Sinharoy and Sistra, including with
Blumrich.**

511. One of ordinary skill would have been motivated to modify
Sinharoy’s interconnect topology in view of *Sistla*’s disclosure of a ring
interconnect topology. For purposes of Ground 2D below, the presence
of Blumrich does not impact this motivation to combine or the reasonable
expectation of success because Blumrich is cited for a component (table
of requests/request queue) that is not impacted by Sistra’s ring
interconnect technology.

512. *Sinharoy* explains its L3 local regions are connected through an on-
chip “interconnect that physically occupies a horizontal equatorial trunk
that runs across the center of the POWER7 chip.” (*Sinharoy*, Fig. 1, 22-

23.) *Sinharoy*'s on-chip interconnect uses an "arbitration logic located at the center of the chip." (*Sinharoy*, 23.) Each L3 cache controller (part of *Sinharoy*'s distributed "coherence fabric"), for example, receives snoops through two sets (odd, even) of coherent interconnect routes from the cores or other instances of the distributed L3 cache controllers. (*Sinharoy*, 19, 22-23, Fig. 1)

513. A POSITA would have found it obvious to use a ring interconnect in a multicore processor architecture with all the relevant features of *Sinharoy*'s architecture. *Sistla*, for example, teaches a multicore processor with a distributed cache much like *Sinharoy*'s architecture. (*Sistla* ¶[0001], Figs. 3-4.) *Sistla*'s multicore processor comprises "a plurality of core-cache clusters 310₁-310_N in communication with each other over an on-die interconnect 320." (*Sistla*, ¶¶[0030]-[0031].) Each core-cache cluster includes a scalability agent (a distributed portion of a caching agent) that maintains memory coherency. (*Sistla* ¶[0018].) *Sistla* teaches that "on-die interconnect 320 is configured as a ring interconnect." (*Sistla* ¶[0030].) *Sistla* thus demonstrates that the relevant features in *Sinharoy*'s architecture are compatible with a ring interconnect, and a POSITA would have had a reasonable expectation of

success in modifying *Sinharoy*'s interconnect topology to include a ring interconnect.

514. A POSITA would have found it obvious to use a ring interconnect to link the cores, caches, and caching agent in an architecture like *Sinharoy*'s, because a ring interconnect would have provided a high-performance, scalable interconnect topology that would have been expected improve *Sinharoy*'s bus-style interconnect architecture with a centralized arbitration logic. A POSITA would have understood, for example, that a ring interconnect provides a simple, fast, and efficient interconnection between cores and caches in a multicore processor. (*See Marty*, 24, 45; *Chrysos* ¶[0018].) Specifically, a ring interconnect provides higher bandwidth compared to a bus and supports a larger number of processor cores. (*See Blake*, Tbl. 2).

515. A POSITA also would have understood that a bus interconnect, like *Sinharoy*'s interconnect design, requires a pipelined design with centralized arbitration logic to maintain cache coherency resulting in significant performance loss. (*Marty*, 46-47.) In contrast, “[r]ings offer simple, distributed arbitration and access methods.” (*Marty*, 48.) A ring interconnect “can also aid in reliability” by retaining sent messages and

resending that message if the data that becomes corrupted, which is important in multicore processors. (*Marty*, 48).

516. A ring also provides other advantages over a bus system interconnect like the type employed by *Sinharoy*³. For example, a Sinharoy bus interconnect suffers from low bandwidth between adjacent cores compared to a ring interconnect that provides high bandwidth between adjacent cores. Also, a Sinharoy bus interconnect is suitable for smaller number of cores and more complex scalability compared to ring interconnect that is suitable for a large number of nodes and leading to much simpler scalability. (*Blake*, Tbl. 2.) Furthermore, an architecture like the Sinaroy interconnect face scalability limitations due to their need

³ “The on-chip data interconnect consists of eight 16-byte buses that span the horizontal trunk. Four flow from left to right, and the other four flow from right to left. These buses are bracketed by memory controllers found at the left and right edges of the chip. They are divided into multiple segments, such that multiple 16-byte data packets may be pipelined within the multiple segments of the same bus at any given time. The buses operate at the on-chip bus frequency.” *Sinharoy* 23.

for centralized arbitration logic, whereas rings do not suffer from the same problem. (*Marty*, 46.)

517. *Sistla*'s teachings demonstrate that a POSITA would have understood that replacing *Sinharoy*'s interconnect topology with a ring topology, as *Sistla* teaches, would have been a simple substitution of one element for another, such that doing so would have been compatible with the basic structure and design of *Sinharoy*'s architecture. And because *Sistla* employs an architecture analogous to *Sinharoy*'s architecture (i.e., a multicore processor having a distributed shared cache and caching agent configuration), a POSITA would have reasonably expected to succeed in making such a substitution. (*Compare Sinharoy*, Fig. 1, with *Sistla*, Figs. 3-4.) The combination is nothing more than a combination of prior art elements arranged according to known methods to yield predictable results. A POSITA thus would have found it obvious to implement *Sinharoy*'s basic architecture using a ring interconnect topology as an alternative to *Sinharoy*'s bus-style interconnect.

3. Sinharoy Combined With Sistra Discloses, And/ Or Renders Obvious, Claim 5.

q) Claim 5: “5. The apparatus of claim 4, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.”

518. The *Sinharoy-Sistla* combination renders claim 5 obvious.

519. The *Sinharoy-Sistla* result would have been that *Sinharoy*’s 4-MB L3 regions including their controllers (“plurality of distributed portions of the caching agent”) would be coupled via a ring interconnect as discussed above in the motivation to combine

F. Ground 2D: Claim 19 Is Obvious Over Sinharoy Combined With Sistra and Blumrich.

r) Claim 19: “19. 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.”

520. The *Sinharoy-Blumrich-Sistla* combination renders claim 19 obvious.

Claim 19 recites a combination of claim 16 (upon which it depends) in
conjunction with the elements recited in element [1e] and claims 4 and 5.

521. *Sinharoy* discloses [1e] and claim 4 as per Ground 2A. *Sinharoy-Blumrich* discloses claim 16 as per Ground 2B. *Sinharoy-Sistla* renders obvious claim 5 as per ground 2C. This combination merely adds the well-known ring interconnect from *Sistla* to the *Sinharoy-Blumrich* combination. The motivations to combine *Sistla* with *Sinharoy* are detailed in Ground 2C. *Blumrich*’s disclosure is separate and does not impact either these motivations to combine or reasonable expectation of success because, for example, *Blumrich* is cited for the use of a table of requests within a caching agent and *Sistra* is cited for the interconnect technology between different caching agents.

XI. SECONDARY CONSIDERATIONS DO NOT REBUT THE PRIMA FACIE CASE OF OBVIOUSNESS

522. In my opinion, secondary considerations do not rebut the prima facie case of obviousness for at least two reasons. *First*, I have not seen any evidence of any secondary consideration to date. Indeed, the patent applicant never proffered such evidence during prosecution of the '228 patent. To the extent patent owner seeks to proffer secondary considerations evidence in this proceeding, I reserve the right to comment on such evidence.

523. *Second*, I do not believe that the patent will be able to establish a nexus between any secondary consideration and the challenged claims. I have been advised that a nexus does not exist where the alleged secondary consideration results from something other than what is both claimed and novel in the claims. As I explained above, there is nothing novel in the claims. Indeed, all claim elements are disclosed in the prior art. Thus, I do not see a basis for the patent owner to assert that there is a nexus between any secondary consideration and the claims.

XII. CONCLUSION

524. I may utilize the documents cited and/or listed herein, or portions of those documents, as exhibits at any hearing or trial in this proceeding. I

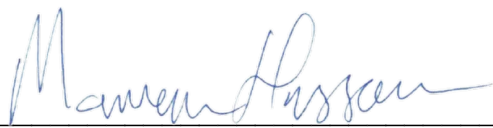
Declaration of Marwan Hassoun, Ph.D. Under 37 C.F.R. § 1.68 in Support of
Petition for *Inter Partes* Review of U.S. Patent No. 8,984,228

may further prepare and use exhibits that summarize portions of my testimony or key terms or concepts presented therein, or other demonstrative exhibits, at any hearing or trial in this proceeding.

525. I reserve the right to supplement my testimony and this report in response to any judicial determinations, in response to the opinions expressed by the patent owner's experts in this proceeding, and/or in light of additional evidence or testimony brought forth at trial or otherwise brought to my attention after the date of my signature below.

526. I hereby declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct, and that all statements made of my own knowledge are true and that all statements made on information and belief are believed to be true. I understand that willful false statements are punishable by fine or imprisonment or both.
See 18 U.S.C. § 1004.

December 10, 2024

A handwritten signature in blue ink, reading "Marwan Hassoun", is written over a horizontal line.

Marwan Hassoun

IPR2025-00207

Declaration of Marwan Hassoun, Ph.D. Under 37 C.F.R. § 1.68 in Support of
Petition for *Inter Partes* Review of U.S. Patent No. 8,984,228

IPR2025-00207

Declaration of Marwan Hassoun, Ph.D. Under 37 C.F.R. § 1.68 in Support of
Petition for *Inter Partes* Review of U.S. Patent No. 8,984,228

EXHIBIT A

EXHIBIT B (Materials Considered)

- All documents cited and discussed in this report