

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Mercedes-Benz USA, LLC,
Petitioner,

v.

Daedalus Prime LLC,
Patent Owner.

Case [To be assigned]
Patent 9,575,895

**DECLARATION OF DR. ROBERT W. HORST
IN SUPPORT OF MERCEDES' PETITION FOR *INTER PARTES* REVIEW
OF CLAIMS 1-17 OF U.S. PATENT NO. 9,575,895**

Filed on behalf of Petitioner:
Celine Jimenez Crowson (Reg. No. 40,357)
Joseph J. Raffetto (Reg. No. 66,218)
Scott Hughes (Reg. No. 68,385)
Helen Y. Trac (Reg. No. 62,250)
Ryan Stephenson (Reg. No. 76,608)
Nicholas Rotz (Reg. No. 75,959)
HOGAN LOVELLS US LLP
555 13th Street N.W.
Washington, D.C. 20004
Telephone: 202.637.5600
Facsimile: 202.637.5710

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
A. Engagement	1
B. Background and Qualification	1
C. Information Considered.....	4
II. BACKGROUND OF THE '895 PATENT	5
A. Technology Background	5
B. The '895 Patent	9
III. CLAIM CONSTRUCTION AND POSITA DEFINITION.....	12
A. Claim Construction.....	12
B. Definition of a Person of Ordinary Skill in the Art.....	12
IV. UNDERSTANDING OF LEGAL STANDARDS.....	12
A. Anticipation	13
B. Obviousness.....	14
V. THE ELEMENTS IN CLAIMS 1-17 OF THE '895 patent are ANTICIPATED OR RENDERED OBVIOUS BY the prior art	17
A. Grounds Based on Sinharoy	18
1. <u>Ground 1</u> – Sinharoy Anticipates or Renders Obvious Claims 1, 3-7, 11-13, and 15-17.....	20
2. <u>Ground 2</u> – Sinharoy in view of Xu Renders Obvious Claims 2 and 10	54
3. <u>Ground 3</u> – Sinharoy in view of Tang Renders Obvious Claims 8-9 and 14.....	58
B. Grounds Based on Tang	70

1.	<u>Ground 4</u> – Tang Anticipates or Renders Obvious Claims 1, 5-9, 11-14, 15, 17	72
2.	<u>Ground 5</u> – Tang in view of Harikumar Renders Obvious Claims 2, 3-4, 10, and 16	102
VI.	CERTIFICATION.....	111
	<u>APPENDIX A</u>	1

I, Dr. Robert W. Horst, declare:

I. INTRODUCTION

A. Engagement

1. I have been retained on behalf of Mercedes-Benz USA, LLC (“**Mercedes**”) to offer technical opinions relating to U.S. Patent No. 9,575,895 (EX1001) (the “**895 Patent**”) and prior art references relating to its subject matter.¹

2. I have no financial interest in either party or in the outcome of this proceeding. I am being compensated for my work as an expert on an hourly basis at my standard consulting rate of \$650 per hour. My compensation is not dependent on the outcome of these proceedings or the content of my opinions.

B. Background and Qualification

3. I am currently Principal at Horst Tech LLC, where I perform research, design, and manufacturing of electronic systems and devices. I also serve as an Adjunct Research Professor at the University of Illinois at Urbana-Champaign in the Department of Electrical and Computer Engineering. All of my opinions stated in this declaration are based on my own personal knowledge and professional

¹ For brevity, when providing citations herein, I will generally refer to the name of the reference, rather than the exhibit number. The exhibit numbers associated with each reference can be found in Appendix A.

judgment. In forming my opinions, I have relied on my 47 years of experience as an engineer, technical director, researcher, chief technology officer, consultant, and research professor in the processor, CPU, cache, memory and input/output (I/O) systems, and computer hardware field.

4. I am over 18 years of age and, if I am called up on to do so, I would be competent to testify as to the matters set forth herein. I understand that a copy of my current curriculum vitae, which details my education and professional and academic experience, is being submitted by Petitioner as Exhibit 1003. The following provides an overview of some of my experience that is relevant to the matters set forth in this declaration.

5. I earned my M.S. (1978) in electrical engineering and Ph.D. (1991) in computer science from the University of Illinois at Urbana-Champaign after earning my B.S. (1975) in electrical engineering from Bradley University. During my master's program, I designed, constructed, and debugged a shared memory parallel microprocessor system. During my doctoral program, I designed and simulated a massively parallel, multi-threaded task flow computer.

6. After receiving my bachelor's degree and while pursuing my master's degree, I worked for Hewlett-Packard Company. While at Hewlett-Packard, I designed the micro-sequencer and cache of the HP3000 Series 64 Processor. From 1980 to 1999, I worked at Tandem Computers, which was acquired by Compaq

Computers in 1997. While at Tandem, I was a designer and architect of several generations of fault-tolerant computer systems and was the principal architect of the NonStop Cyclone superscalar processor.

7. I also have various publications, including publications related to CPU architecture and storage. I have also received the Distinguished Alumni Award from the University of Illinois Department of Electrical and Computer Engineering for “Pioneering Contributions to Fault-tolerant Computer Architecture.” I have served on various IEEE committees and was elected as an IEEE fellow “for contributions to the architecture and design of fault tolerant systems and networks.

8. I have my own patents directed to processor, memory, and input/output (I/O) system design and aspects of circuit chip design, such as U.S. Pat. No. 5,146,589, U.S. Pat. No. 5,287,472, U.S. Pat. No. 5,329,629, and U.S. Pat. No. 5,034,964. I have testified as an expert witness on over fifty patent matters related to computer, processor, and network systems before the Patent Trial and Appeal Board (PTAB). I have also served as an expert witness in patent and intellectual property litigation as well as on several inter partes review matters, such as IPR2018-01249, IPR2018-01315, IPR2018-01316, IPR2018-00371, IPR2018-00372, IPR2018-00374, IPR2018-00375, IPR2018-00327, IPR2018-00328, IPR2018-00329, IPR2017-01707, IPR2017-01711, IPR2017-01714,

IPR2017-01718, IPR2017-01728, IPR2017-01729, IPR2017-01732, IPR2017-01733, IPR2017-01391, IPR2017-01392, IPR2017-01393, IPR2017-01395, IPR2017-01406, IPR2017-01559, IPR2017-01705, IPR2017-01713, IPR2016-0897, IPR2016-0902, IPR2016-0903, IPR2014-01469, IPR2015-00159, IPR2015-00161, IPR2015-00163, IPR2015-00172, IPR2014-00949, IPR2014-00901, and IPR2013-00440.

9. Based on my experience and education, I am familiar with the design, operation, and functionality of processor systems described in the '895 Patent.

10. I believe that I am qualified to opine as to the knowledge and level of skill of one of ordinary skill in the art at the time of the alleged invention of the '895 Patent, as well as the state of the art at that time.

C. Information Considered

11. I have reviewed the '895 Patent and its prosecution history, as well as the other materials referenced in Appendix A. Counsel has informed me that I should consider these materials through the lens of one of ordinary skill in the art related to the '895 Patent, at the time of the earliest purported priority date of the '895 Patent, and I have done so during my review of these materials. I have been asked to assume, for purposes of this Declaration, that the '895 Patent has a priority date of December 13, 2011 (the “**Critical Date**”).

12. My analyses are based on my years of education, research, and work experience, as well as my investigation and study of relevant materials. In my analyses, I have considered the materials that I identify in this Declaration and those listed in Appendix A.

13. I may rely on these and additional materials to respond to arguments raised by the Patent Owner. I may also consider additional documents and information in further analyses—including documents that may not yet have been provided to me.

14. My review and assessment of the materials provided in this proceeding is ongoing, and I will continue to consider any new material as it is provided. I reserve the right to review, supplement, and amend my analyses based on new information and on my continuing review of the materials already provided.

II. BACKGROUND OF THE '895 PATENT

A. Technology Background

15. The '895 Patent is related to a multicore processor system with a distributed or partitioned cache memory. This was known in the art well before the '895 Patent's filing date.

16. Multicore processor systems generally have a plurality of cores, one or more memory controllers, a shared cache memory among the cores, and

integrated input/output (I/O) controller coupled with I/O devices. Each core has access to memory through one or more levels of cache memory. Typically each core has its own first level instruction and data cache, while higher level caches may be shared among multiple cores. To manage the cache coherence among the cores and caches, generally there is a single logical cache controller (or “caching agent”) to manage the coherence operations. The logical controller can be physically distributed or partitioned and have different physical portions.

17. In particular, the processor can handle transactions from an IO device coupled to the processor by directly storing the transaction data into its shared cache memory ('895 Patent, Claim 1.) It enables the processor to bypass the traditional step of storing data in a system memory before transferring it to the cache. (*See id.*, 3:34-37.) This operation is commonly referred to as “cache injection.” (*See, e.g.*, Sinharoy 1:17.) Cache injection involves directly storing data into the cache to reduce memory traffic and improve I/O processing time by eliminating the necessity of initially saving the data in a system memory. This was well-known in the art to a POSITA.

18. Furthermore, the “caching agent” or cache controller of the '895 Patent has distributed portions, which each corresponds to a given core ('895 Patent, Claim 1.) The distributed portions manage cache coherence for the processor. This division minimizes the allocation overhead associated with the

transactions and reduces the volume of snoop traffic necessary (*See id.* 3:12-15.)

This partition concept was not novel. As demonstrated in this Declaration, various prior art references already disclosed and taught this distributed or partitioned cache controller (caching agent).

19. Among caches in a multicore processor system, a processor generally relies on a cache controller (or a caching agent) to perform cache coherence protocols to ensure that multiple caches storing copies of the same data remain consistent with each other. In the processor, each core may have its own cache memory, and if multiple cores are accessing the same shared memory location, there is a possibility of data inconsistency. To avoid the data inconsistency, the cache controller (or caching agent) performs cache coherence protocols to assure that if multiple cores/processors are reading the same memory location, they will always get the same value.

20. There are various cache coherence protocols well-known in the art, for example, MOESI (with cache states “Modified, Owned, Exclusive, Shared, Invalid”).² A summary of the MOESI states is given by Yi³:

- Modified state: No other processor has a copy of the data block. The copy of the data in main memory is incorrect.
- Owned state: Only one processor can be in the owned state. All other processors can have a copy of the most recent in the shared state. The copy of the data in main memory can be incorrect.
- Exclusive state: No other processor has a copy of the data. The copy of the data in main memory is also the most recent.
- Shared state: Other processors can also have copies of the data in the shared state. The copy of data in main memory is the most recent.
- Invalid state: Either main memory or another processor cache can have valid copies of the data.

21. Each cache has a copy of the sharing status, and the cache controller uses a mechanism such as snooping to update the sharing status. When a processor stores to memory, depending the cache architecture, the data may be stored in both the cache and main memory (“write through”), or only in the cache (“write back”). With a write back cache, read data may be supplied by a cache holding a valid

² See, e.g., Yi, Kyueun & Jean-Luc Gaudiot, *Architectural implications of cache coherence protocols with network applications on chip multiprocessors* in Keqiu Li et al., NETWORK AND PARALLEL COMPUTING, 394-403 (2007).

³ See *id.*

copy, while in a write through cache, the data is typically supplied from memory.

Cache controllers are designed to ensure data consistency and correctness.

B. The '895 Patent

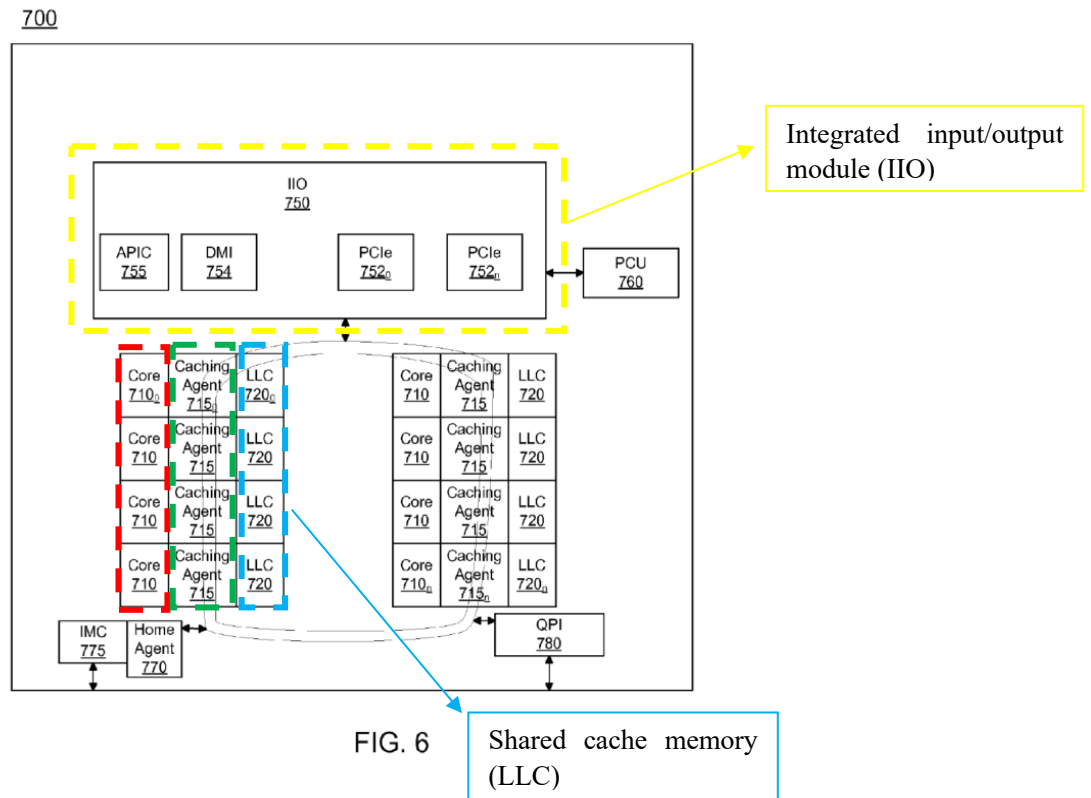
22. I understand the '895 Patent was filed on January 30, 2015, and issued on February 21, 2017. I also understand that the '895 Patent is a continuation of U.S. Pat. No. 8,984, 228, filed on December 13, 2011.

23. I have assumed, for purpose of this declaration, that the '895 Patent is entitled to this Critical Date of December 13, 2011.

24. The '895 Patent discloses a processor that includes multiple “cores,” a “shared cache memory,” an “integrated input/output module” (an IIO module, e.g., an I/O bus or I/O controller) coupled with an IO device (an input/output device), a “caching agent” (e.g., a cache controller). ('895 Patent, Claim 1.) These processor elements had been well-known in the art long before the filing date of the '895 Patent.

25. Specifically, the processor in the '895 Patent is adapted to “receive an allocation transaction from the IO device and directly store data of the allocation transaction into the shared cache memory.” (*Id.*) The processor is configured to store data from an I/O device directly into a shared cache memory without first storing the data in a system memory. (*See id.* at 3:34-37.) Further, the “caching agent” is a “single caching agent” for the processor with “distributed portions,”

each associated with a corresponding core. ('895 Patent, Claim 1.) The language in the '895 implies a differentiation between “logical” sharing/distribution and “physical” sharing/distribution. As discussed below in this Declaration, this “distributed” or partitioned setting of the caching agent and direct data storage in the cache were well-known in the art as well.



(*Id.*, Fig. 6 (annotated).)

26. Referring to annotated Figure 6 above, the '895 Patent describes how I/O data from an I/O device coupled to the IIO can be directly stored into the shared cache memory, i.e., the last level cache (LLC). (*Id.*, 3:20-3:32.) It provides a mechanism of cache injection without “first storing the data in a system memory.” (*See id.*, 3:34-37.) For example, an “allocation transaction” can be a

“request to write data directly into a cache memory,” which can be received by the caching agent from an I/O device. (*See id.*, 3:55-59.) Moreover, 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. (*See id.*, 6:33-39.) The patent notes that although “distributed caching agents are shown, understand that these distributed portions form a single caching agent.” (*See id.*, 6:35-37.)

27. The patent states that “the data can be stored coherently in accordance with a given cache coherency protocol (e.g., a modified exclusive shared invalid (MESI) protocol).” (*See id.*, 4:3-5.) Furthermore, an embodiment is described in which the cache includes ownership information, “based on state information associated with a cache line, e.g. an ownership indication as well as cache coherency information.” (*See id.*, 4:49-52.) I note that the combination of a MESI protocol and ownership information is known as a MOESI protocol in which the “O” stands for Owned.

28. The multiple cores may be coupled to each other and to a “shared cache memory formed of a plurality of units of a LLC 740₀-740_n⁴ via caching agent 710_n-715_n.” (*See id.*, 6:42-45.) The ’895 describes that the shared cache memory (LLC) can be partitioned as well. (*Id.*, 6:34-35.) Each distributed portion

⁴ The LLC 740₀-740_n should have been 720₀-720_n in FIG. 6.

of the LLC is associated with a corresponding core. (*Id.*, 6:42-46.) The cores, LLC, IIO module, and caching agent may be coupled together via a ring interconnect. (*Id.*, 6:45-49.) As discussed in greater detail below, there is nothing novel or inventive concerning these elements or a combination of these elements in the '895 Patent.

III. CLAIM CONSTRUCTION AND POSITA DEFINITION

A. Claim Construction

29. I understand that in an *inter partes* review, claims must be given their ordinary and customary meaning, as understood by one of ordinary skill in the art in light of the specification and prosecution history. I apply that standard in my analysis below to the words and phrases in the claims of the '895 Patent.

B. Definition of a Person of Ordinary Skill in the Art

30. A person of ordinary skill in the art (“**POSITA**”) in the field of the '895 Patent would have had an undergraduate degree in computer science, computer engineering, electrical engineering, or similar disciplines, along with two years of experience working with multicore processor systems, central processing unit (CPU) , cache, and memory systems. The more education one has, the less experience needed to attain an ordinary level of skill. Similarly, more experience in the field may serve as a substitute for formal education.

IV. UNDERSTANDING OF LEGAL STANDARDS

A. Anticipation

31. I understand that a patent claim is invalid when the invention that it claims is not new. To establish that a claimed invention is not new (a.k.a., “not novel”), I understand that one may establish that a single publication, or other reference in the prior art discloses (explicitly or inherently) every element required in a patent claim (*i.e.*, all features or “limitations” recited in the patent claim). I understand that a reference in the prior art “anticipates” a claimed invention if that reference discloses, either explicitly or inherently, every element of the claim.

32. I understand that “prior art” and “prior art reference” are legal terms of art referring to, for example, devices, methods, and publications that predate the earliest effective filing date of the claimed invention.

33. I understand that a prior art reference that anticipates a claim may disclose an element or limitation of a patent claim expressly or inherently. A prior art reference discloses an element or limitation of a patent claim inherently when the prior art’s disclosure necessarily requires or implies that the claimed element or limitation be present in the process, machine, manufacture, or composition disclosed. I understand there is still considered to be an inherent disclosure even if the author of the reference did not describe or understand the underlying inherent principle. I understand that one may establish that a prior art reference inherently

discloses a claimed element or limitation through the use of other reference material or through testing.

34. I understand that the description in a written reference does not have to be in the same words as the claim, but all of the requirements of the claim must be there, either stated or necessarily implied, so that a POSITA looking at that one reference would be able to make and use the claimed invention.

35. I understand that any prior art reference that can be considered for purposes of determining whether it anticipates a patent claim may also be used to determine whether the reference renders that claim obvious, as discussed below.

B. Obviousness

36. I understand that, even if a claim is not fully disclosed in a single prior art reference, the patent claim is invalid if the invention would have been obvious to a POSITA at the time of the invention. In particular, I understand that a patent claim is normally invalid as obvious if it would have been an “ordinary innovation” within the relevant field to create the claimed product or method at the time of the invention.

37. I understand that the relevant portion of pre-AIA Section 103, subsection (a) of the Patent Act states:

“A patent may not be obtained through the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented 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 said subject matter pertains.”

38. I understand that, by way of example only, a claimed invention is obvious if:

it combines prior art elements according to known methods to yield predictable results;

it simply substitutes one known element for another to obtain predictable results;

it uses a known technique to improve similar devices (methods, or products) in the same way;

it applies a known technique to a known device (method, or product) ready for improvement to yield predictable results;

it was “obvious to try” in that the inventor chose from a finite number of identified, predictable solutions, with a reasonable expectation of success;

known work in one field of endeavor prompted variations of it for use in either the same field or a different one based on design incentives or other market forces and those variations were predictable to one of ordinary skill in the art; or

some teaching, suggestion, or motivation in the prior art led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

39. When considering obviousness, I understand that I am to: (i) decide the level of ordinary skill in the field that someone would have had at the time the alleged invention was made; (ii) determine the scope and content of the prior art; (iii) determine what differences, if any, existed between the prior art and the Asserted Claims; and (iv) consider objective evidence of non-obviousness (also known as secondary considerations). Further, when considering obviousness, I understand that it is not necessary to seek out precise teachings, and it is permissible to consider the inferences, common sense, and creative steps that a POSITA (who is considered to have an ordinary level of creativity and is not an automaton) would employ.

40. I understand that objective evidence relevant to the issue of obviousness may also be considered. This type of evidence is sometimes referred to as “secondary considerations,” and may include evidence of commercial success, long-felt but unsolved needs, failure of others, and unexpected results. I understand that how much weight to afford evidence of secondary considerations is made on a case-by-case basis. I further understand that if the patent owner presents evidence of secondary considerations, that fact—the mere fact of presenting this

type of evidence—does not mean that the evidence is dispositive of the issue of obviousness. With respect to secondary considerations, I understand that Patent Owner bears the burden of proof. I have seen no evidence to date that any secondary considerations would establish non-obviousness.

V. THE ELEMENTS IN CLAIMS 1-17 OF THE '895 PATENT ARE ANTICIPATED OR RENDERED OBVIOUS BY THE PRIOR ART

41. I have been asked to provide an analysis as to whether the elements of Claims 1-17 of the '895 Patent (the “**Challenged Claims**”) are disclosed in the primary prior art references: *IBM POWER7 multicore server processor* (“**Sinharoy**”) (EX1003) and *DMA Cache: Using On-Chip Storage to Architecturally Separate I/O Data from CPU Data for Improving I/O Performance* (“**Tang**”) (EX1004). I was also asked to consider this reference in view of certain additional prior art for some of the limitations of the Challenged Claims, including *Explorations of Optimal Core and Cache Placements for Chip Multiprocessor* (“**Xu**”) (EX1005), and U.S. Patent Application Publication No. 2009/0164739 to Harikumar (“**Harikumar**”) (EX1006).

42. My analysis of these prior art references relative to the elements of the Challenged Claims—specifically, how and where the prior art references disclose the limitations of the challenged claims—is provided below. The citations that I have included are not intended to provide an exhaustive list, but rather provide examples of how the references disclose or teach the elements of these claims.

A. Grounds Based on Sinharoy

43. I have reviewed Sinharoy. Sinharoy is directing to a multicore server processor, the “POWER7,” that is the seventh-generation processor in IBM’s POWER processor line, and systems comprising POWER7 processors. Sinharoy was published and has been available since April 29, 2011. I understand Sinharoy is prior art under at least pre-AIA 35 U.S.C. § 102(a).

44. My review of the ’895 Patent file history shows that Sinharoy was not considered during prosecution of the ’895 Patent.

45. Sinharoy discloses a POWER7 server processor which has eight cores, two memory controllers, an L2 cache, and an L3 cache. (Sinharoy 1:2; 1:18-19.) Sinharoy discloses that the POWER7 processor has a “256-KB POWER7 L2 cache” and a “shared 32-MB Power L3 cache,” “composed of eight 4MB L3 regions.” (*Id.* 1:18-19; Figure 1.) The L2 cache and L3 cache each has centralized cache controllers to manage the read/write requests and cache coherence operations. (*See id.* 1:18-19). POWER7 processor supports “two integrated I/O controllers.” Each of the I/O controllers supports a “4-byte off-chip read interface and 4-byte off-chip write interface, thereby connecting the POWER7 chip to up to two I/O hub chips.” (*Id.* 1:22.) The I/O hub chips, PCI bridge, and PCI adapter are examples of input/output (I/O) devices that can be connected to the POWER7 processor via the integrated I/O controllers. (*Id.* Figure 14.)

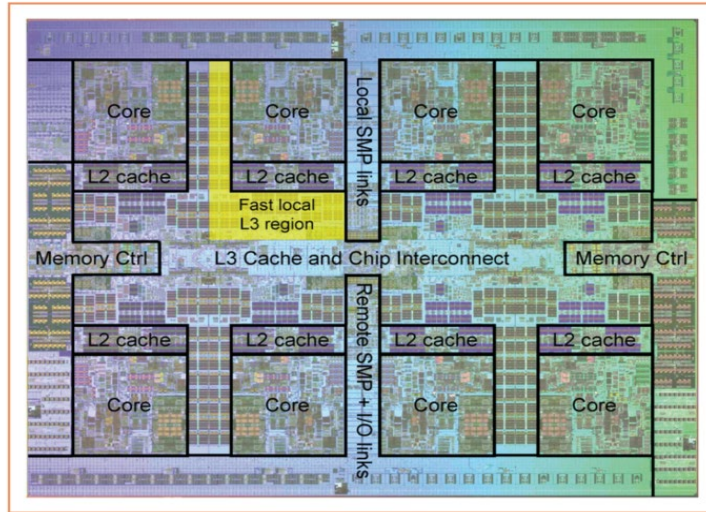


Figure 1

Die photo of the IBM POWER7 chip.

(Sinharoy, Figure 1.)

46. Sinharoy teaches that, the L2 cache and L3 cache support “cache injection,” i.e., an “I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory.” (*Id.* 1:17.) The integrated I/O controllers also support cache injection, which “enables DMA write traffic to be written directly into a cache, instead of to memory.” (*Id.* 1:22.) The cache injection operation is a cache coherence operation to directly store data from an I/O device connected to the POWER7 processor into the cache memory.

1. Ground 1 – Sinharoy Anticipates or Renders Obvious Claims 1, 3-7, 11-13, and 15-17

a. Independent Claim 1

1[p]: *A processor comprising:*

47. In my opinion, Sinharoy teaches Claim 1[p] to the extent the preamble limits the scope of the claim. As I described above, Sinharoy’s POWER7 processor system is “an eight-core processor.” (*Id.* 1:1 (“this seventh-generation POWER processor”).)

1[a]: *a plurality of cores,*

48. In my opinion, Sinharoy teaches Claim 1[a]. Sinharoy discloses that the POWER7 processor includes a plurality of cores. (Sinharoy 1:1-2.) In particular, Sinharoy discloses that the POWER7 is a processor chip, and “[o]n the chip is an eight-core processor.” (*Id.*, 1:1; *see also* 1:2 (“Figure 1 shows the POWER7 chip, which has eight processor cores, each with 12 execution units capable of simultaneously running four threads.”); *see also* 1:17 (“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.”).)

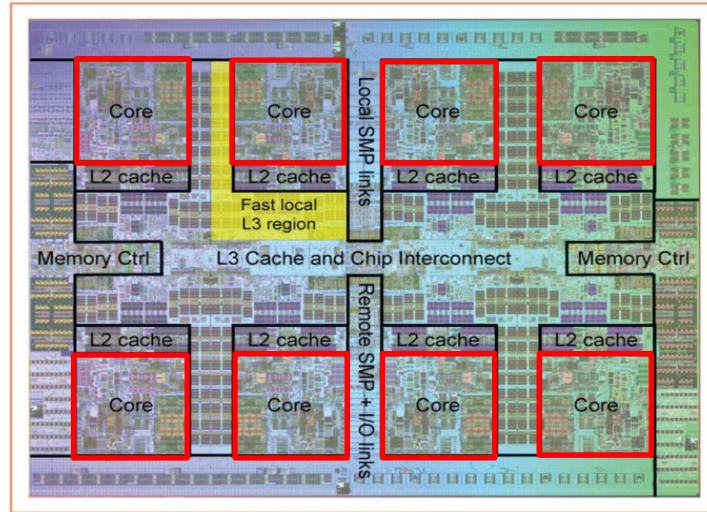


Figure 1

Die photo of the IBM POWER7 chip.

(*Id.*, Figure 1 (annotated).)

1[b]: a shared cache memory,

49. In my opinion, Sinharoy teaches Claim 1[b]. Sinharoy discloses a “shared 32-MB L3 cache” comprising “eight 4-MB L3 regions.” (Sinharoy 1:18; *see also* 1:2 (“the large 32-MB-shared-L3 cache to be on-chip”) (emphasis added).) Each local L3 cache region is “tightly coupled to the L2 cache associated with a given core.” (*Id.*, 1:18.) The “POWER7 cache hierarchy includes a shared 32-MB L3 cache comprised of the 4-MB local L3 regions from the eight cores.” (*Id.*, 1:17.) 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.” (Sinharoy 1:19.) Thus, the L3 cache in Sinharoy is the “shared cache memory” and is shared among the cores.

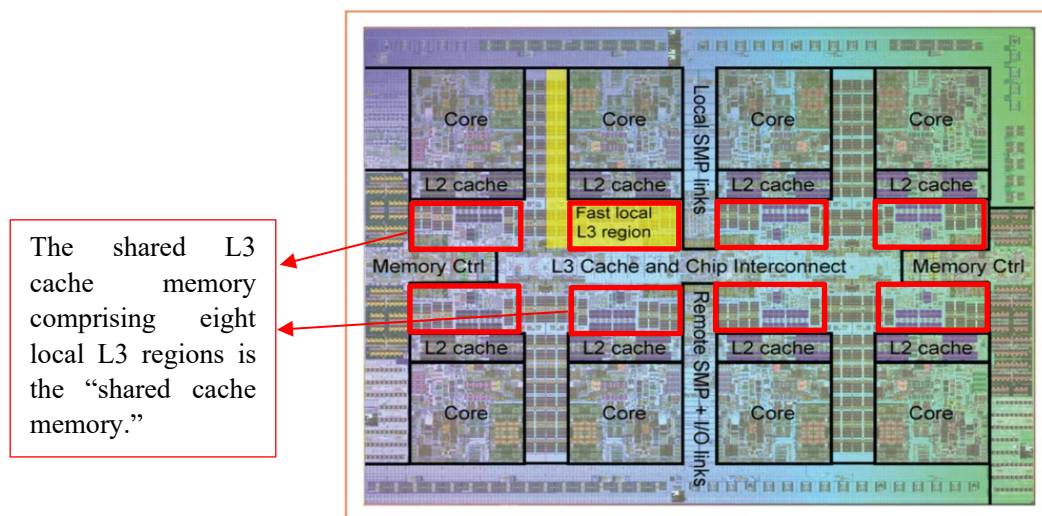


Figure 1
Die photo of the IBM POWER7 chip.

(*Id.*, Figure 1 (annotated).)

1[c]: a memory controller to interface with a memory coupled to the processor,

50. In my opinion, Sinharoy teaches Claim 1[c]. Sinharoy discloses that the POWER7 processor has “two memory controllers—one on each side of the chip.” (Sinharoy 1:2.) Each memory controller supports “four channels of double-data-rate-three (DDR3) memory.” (*Id.*) These “eight channels together provide 100 GB/s of sustained memory bandwidth.” (*Id.*) Sinharoy further discloses a “memory subsystem.” (*See id.*, 1:21-22.) A given memory channel “can be connected through an off-chip interface to a memory buffer chip.” (*Id.*, 1:21.)

51. First, the memory chips used in the memory subsystem of Sinharoy are the “memory coupled to the processor.” The memory subsystem in Sinharoy provides “a dual port interface between the buffer chip and the memory chips” and

may use “800-, 1066-, 1,333, and 1,600-MHz DDR3 memory chips.” (*Id.*, 1:21.)

The POWER7 memory subsystem “will provide in excess of 100 GB/s sustained memory bandwidth per chip.” (*Id.*, 1:22). By providing the memory bandwidth to the POWER7 processor chip and connecting to the processor chip through the off-chip interface and the memory buffer chip, the DDR memory chips are off-chip and the “memory coupled to the processor.”

52. Second, the eight memory channels, supported by the two memory controllers in Sinharoy, involve “interfacing” between the “memory controllers” and the memory chips in Sinharoy, because the channels together “provide 100 GB/s of sustained memory bandwidth.” (Sinharoy 1:2.)

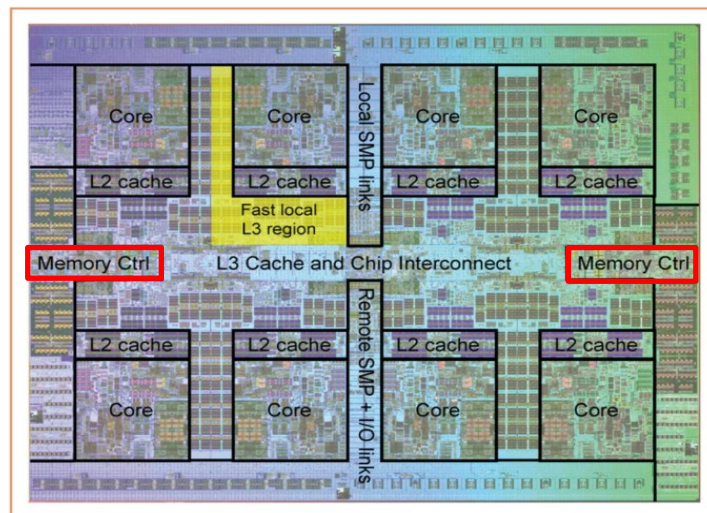


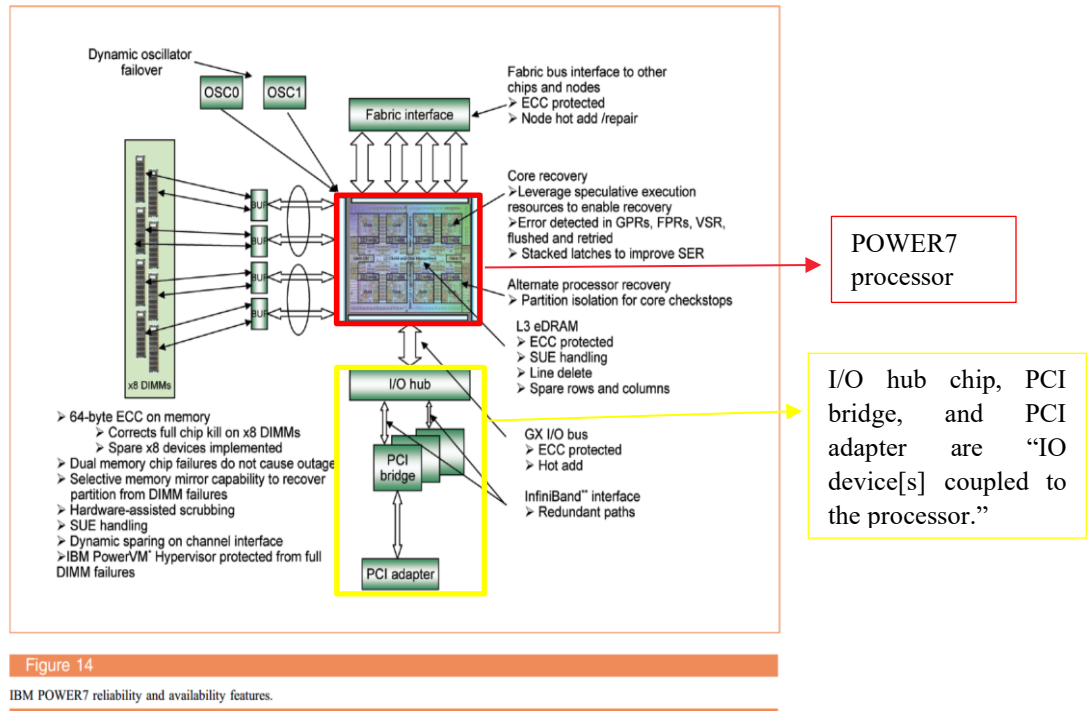
Figure 1

Die photo of the IBM POWER7 chip.

(Sinharoy, Figure 1 (annotated).)

1[d]: an integrated input/output (IIO) module to interface between the processor and an IO device coupled to the processor and

53. In my opinion, Sinharoy teaches Claim 1[d]. Sinharoy discloses that the POWER7 processor includes “two integrated I/O controllers” (i.e., IIO modules). (Sinharoy 1:22.) Each of these two integrated I/O 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.” (*Id.*; see Sinharoy Figure 14.) According to Figure 14 of Sinharoy, an I/O hub chip is further connected to a PCI bridge and PCI adapter. The two I/O hub chips along with the PCI bridge and PCI adapter are the “IO device coupled to the processor,” because they are (1) capable of providing input/output data and (2) connected to the POWER7 processor chip via off-chip read and write interfaces for read/write transactions.



(Sinharoy, Figure 14 (annotated).)

54. Further, the two integrated I/O controllers interact between the POWER7 processor and the I/O hub chips, PCI bridge, and PCI adapter (i.e., IO devices) through the off-chip read and write interfaces. For example, the POWER7 integrated I/O controllers support “cache injection, which enables DMA write traffic to be written directly into a cache, instead of to memory.” (*Id.* 1:22.) The cache injection operation involves write traffic (“interfacing”) between the cache of the processor and the I/O hub chips (i.e., IO devices).

1[e]: a caching agent to perform cache coherency operations for the plurality of cores and the I/O module,

55. In my opinion, Sinharoy teaches Claim 1[e]. Sinharoy discloses the “caching agent,” including the centralized cache controllers for the L2 and L3

caches including 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. (*See* Sinharoy 1:18-19.) In particular, Sinharoy discloses that the L2 cache includes a “single centralized controller, with two address-hashed cache data arrays” and each L3 cache region includes a “single centralized controller.” (*Id.*, 1:18-19.) Both L2 and L3 cache controllers have read/claim (RC) machines incorporated in them, which manage “core fetches, read-modify-write operations on behalf core stores, and core prefetch operations.” (*Id.*, 1:18.) A set of eight fully functional RC machines in the L2 cache controller manage “core fetches and stores that hit the L2 cache.” (*Id.*) A set of 24 more energy-efficient “lightweight” RC machines in the L3 cache controller manage “prefetches and stores that miss the L2 cache.” (*Id.*) The L2 RC machines and L3 lightweight RC machines “aggressively reorder storage operations . . . thereby enabling the high volume of system coherence traffic needed to scale robustly to a large number of sockets.” (Sinharoy 1:18-19 (emphasis added).) In addition, when “prefetched data is staged to the L3 cache, it is managed by a pool of ten write machines.” (Sinharoy 1:19.)

56. These “core fetches, read-modify-write operations on behalf core stores, and core prefetch operations” in the cache controllers are “cache coherency operations for the plurality of cores” in Claim 1[e], because they involve system

coherence traffic among the caches and the cores.⁵ As such, the cache controllers for the L2 and L3 caches (including the pool of RC and write machines) are “caching agent to perform cache coherency operations for the plurality of cores.”

57. The same “caching agent” comprising the L2 and L3 cache controllers with the pool of RC and write machines also performs cache coherency operations for the integrated I/O controllers (i.e., IIO modules). For example, Sinharoy discloses that the POWER7’s integrated I/O controllers (i.e., IIO modules) “support cache injection, which enables DMA write traffic to be written directly into a cache, instead of to memory.” (Sinharoy, 1:22.) Sinharoy discloses that L2 and L3 caches support “new coherent operations”, such as “cache injection. (*Id.*)

⁵ This is consistent with the disclosure in the ’895 Patent. It discloses that “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.” (’895 Patent, 2:44-47.) “[C]aching 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.” (*Id.* 3:1-5.) Examples of “caching agent” include cache controllers to perform cache coherency operations.

Thus, this “cache injection” operation is a “cache coherency operation for the IIO module.”

58. Furthermore, the write machines, which as described above are part of the same “caching agent,” also handle “cast-outs” and these “cache-injection” operations. (Sinharoy 1:19.) Sinharoy discloses that such “cache injection” operations are supported by the L2 and L3 caches and the L2 and L3 cache controllers. (See Sinharoy, 1:17.) Therefore, the cache controllers for the L2 and L3 caches including the pool of RC machines and write machines are also “caching agent . . . to perform cache coherency operations for the IIO module.”

1[f]: 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,

59. In my opinion, Sinharoy teaches Claim 1[f]. As explained above, the L2 and L3 caches in Sinharoy support cache injection operation, where an “I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, instead of to memory.” (Sinharoy 1:17.; *see also supra* Section V.A.1.a (Claim 1[e]).)

60. Details of I/O subsystem that includes such I/O devices are provided in Sinharoy 1:22. In particular, as discussed in Claim 1[d], I/O hub chips, PCI bridge, and PCI adapter connected to the POWER7 processor are “IO devices.” POWER7 I/O integrated controllers (i.e., IIO modules) “support cache injection,

which enables DMA write traffic to be written directly into a cache, instead of to memory.” (*Id.* 1:22.)

61. As would be understood by a POSITA, an example of an I/O transaction is, for example, when an application requests information to be read from a file on a disk drive. The I/O driver software running in one of the cores initiates the read from the disk, but before the data starts to stream from the drive, it is allocated a region of memory for storing the incoming data. In Sinharoy, that region of memory may be cached and the writes made directly to the cache via a direct memory access (DMA). The cache controller cannot allow the writes until the I/O device (e.g., disk controller) has been given write access to that region of memory and any copies in other caches have been invalidated. The cache states in Sinharoy that allow writes are those in Table 2 with “Update” Authority – T, TE, M, ME, MU, TN and TEN. “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.” (Sinharoy 1:17)

62. A POSITA would also understand that when an application, for instance, wants to write information to a disk drive, the core first writes the data to a buffer in memory. The Sinharoy cache is write-back, not write through, and the

write data is first sent from the processor to cache before it is forwarded to the disk controller. The cache controller again assures that the write operation is to a region of cache that allows Update (T, TE, M, ME, MU, TN and TEN). Then the I/O device (e.g., disk controller) reads the information from memory and writes it to the disk drive. The memory read by the I/O device can be from any location with a valid copy of the data – an L3 cache or main memory.

63. Because the integrated I/O controllers support and connect the I/O hub chips via off-chip read and write interfaces, they support read and write transactions from the I/O hub chips. (*Id.* 1:22.) Writing data into a cache memory from an I/O chip (or PCI bridge, PCI adapter, i.e., IO device) involves a request to write data from the IO device into the cache memory, which is an “allocation transaction from the IO device.”⁶

64. In addition, the pool of the write machines in the L3 cache also handles “cache-injections.” (Sinharoy 1:19.) Thus, when they perform a cache

⁶ This is consistent with the disclosure in the '895 Patent. It discloses that the “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 . . .” ('895 Patent 3:55-58.) Examples of “allocation transactions” include requests to read or write data into a cache memory.

injection operation to write data from the I/O hub chip into the L3 cache (i.e., the “shared cache memory” as discussed in Claim 1[b]), the processor “directly store[s] data of the allocation transaction into the shared cache memory.”

1[g]: wherein the caching agent is 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,

65. In my opinion, Sinharoy teaches Claim 1[g]. The cache controllers for the L2 and L3 caches in Sinharoy including the pool of RC and write machines in Sinharoy are a “caching agent” and form a logical “single caching agent” for the processor. (*See supra* Section V.A.1.a (Claim 1[e]) (the cache controllers for the L2 and L3 caches including the pool of RC and write machines performing cache coherence operations for the cores and the IIO module of the processor)). The disclosures in the ’895 Patent show that the caching agent is a single logical unit for the whole processor, which has distributed physical partitions.⁷ In a multicore

⁷ The ’895 Patent discloses 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.” (’895 Patent, 3:6-10.) The ’895 Patent further discloses, with respect to FIG. 6 thereof, that processor 700 includes a distributed configuration having “partitions or slices each including a core 710 and a partition of a caching

processor system, caches need to maintain coherence with each other. A cache cannot maintain its own coherence without responding to transactions to other caches. If one cache ignored transactions to other caches, the system would not operate properly because multiple caches could have copies of the same cache line, but those lines could hold different data. As such, a “single caching agent” is a necessary single logical component formed by physically distributed portions.

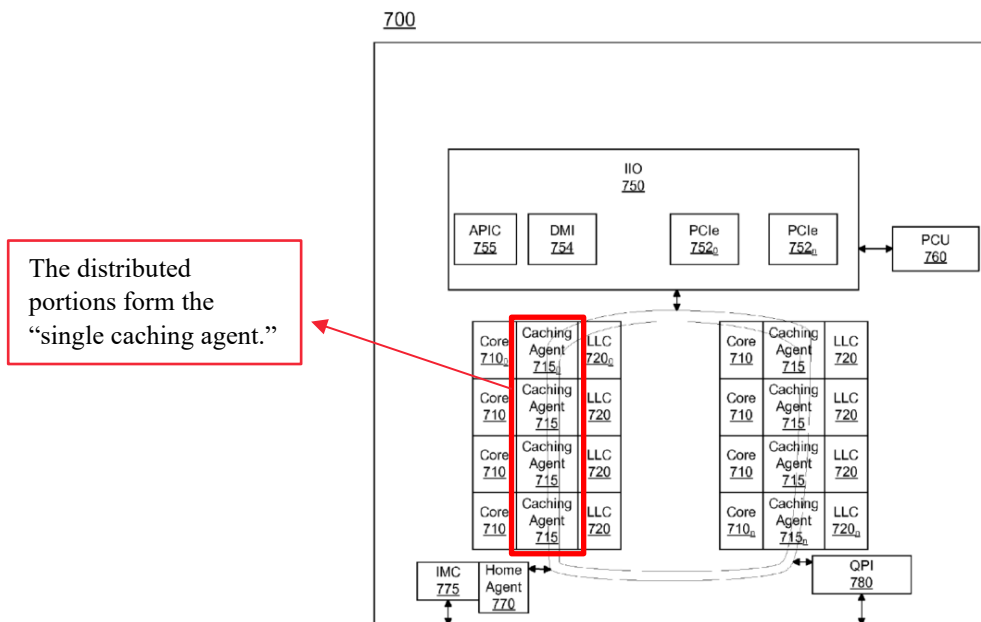
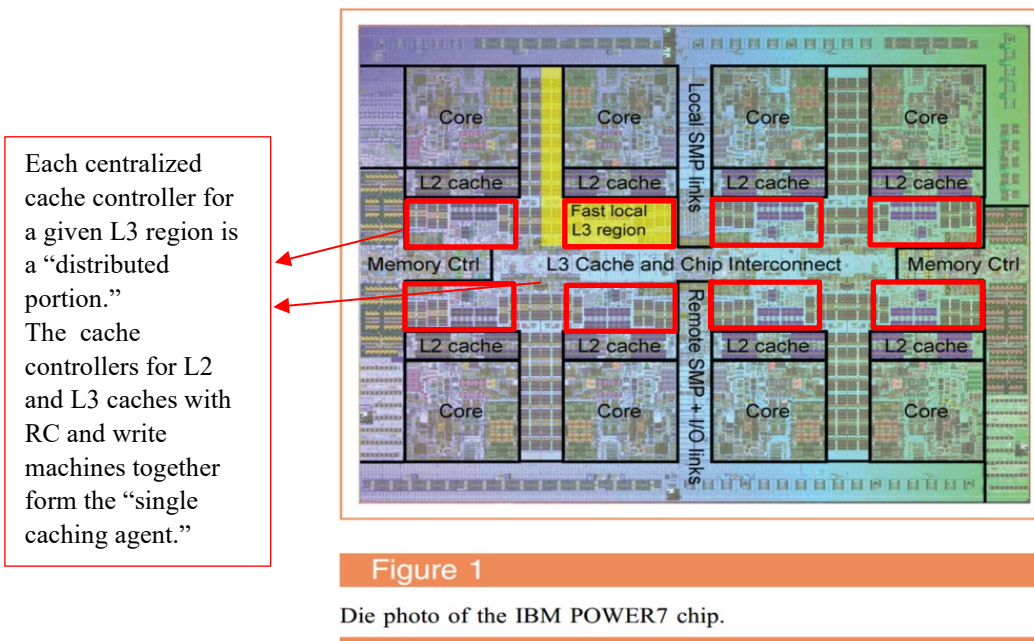


FIG. 6

(’895 Patent, FIG. 6 (annotated).)

agent 715 and a LLC 720.” (*Id.* 6:33-35.) In particular, it discloses that “while distributed caching agents are shown, understand that these distributed portions **form** a single cache agent.” (*Id.* 6:35-37 (emphasis added).)

66. Furthermore, each centralized cache controller for a given L3 region is a “distributed portion” of the “caching agent.” Each “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.” (Sinharoy 1:19 (emphasis added) (indicating each L3 region cache controller corresponds to a given core).) Because the local L3 region is tightly “coupled to the L2 cache associated with a given core,” each centralized controller for its corresponding L3 region is associated with a “corresponding one of the plurality of cores.” (*Id.*)



(Sinharoy, Figure 1 (annotated))

b. Independent Claim 11

11[p]: *A non-transitory machine-readable medium having stored thereon instructions, which if performed by a machine cause the machine to perform a method comprising:*

67. In my opinion, Sinharoy teaches Claim 11[p] to the extent the preamble limits the scope of the claim. Sinharoy discloses a multicore processor that can perform certain instruction sets. (Sinharoy 1:1; *see also* 1:4 (“how instructions flow to the various issue queues and are then sent to the functional units for execution”).) A POSITA would understand that general purpose computers have nonvolatile disks or flash drives and those drives include the object code of programs which are read from the drive and brought into memory in order to run the programs. Thus general purpose computer systems include “machine-readable medium having stored thereon instructions.” A POSITA would understand the instructions in the Instruction Cache (Fig 3, 1:4) in Sinharoy have to come from somewhere and a POSITA would understand the instructions to be read via the I/O system from some type of off-chip nonvolatile memory such as a disk drive or flash drive. The off-chip instruction storage is required in order for the POWER7 to run any code and to perform any methods. Thus a POWER7 system includes a “non-transitory machine-readable medium having stored thereon instructions” to perform a method.

11[a]: receiving an allocation transaction in an integrated input/output (IIO) module of a multicore processor from an IO device coupled to the multicore processor;

68. In my opinion, Sinharoy teaches Claim 11[a]. As stated in Claim 1[a], POWER7 processor in Sinharoy is a multicore processor. (*See supra* Section V.A.1.a (Claim 1[a]).) As explained in Claim 1[f], “allocation transactions” include requests to read or write data into a cache memory. As discussed in Claim 1[d], the integrated I/O controllers in Sinharoy are IIO modules, and the connecting I/O hub chips, PCI bridge, and PCI adapter are IO devices “coupled to the multicore processor.” (*See supra* Section V.A.1.a (Claim 1[d]).)

69. A POSITA would understand that an I/O hub is a chip that interfaces between a processor chip and a standard external bus. Sinharoy Figure 14 shows the I/O hub interfacing the Power7 to InfiniBand buses which are high speed busses used for connecting processors and I/O devices. The PCI bridges in Figure 14 then interface between standard PCI busses and InfiniBand. The PCI bridges and I/O hubs are designed to implement DMA operations to allow PCI devices (i.e., IO devices), such as disk and communication controllers, to transfer directly into and out of memory. I/O integrated controllers (i.e., IIO modules) “support cache injection, which enables DMA write traffic to be written directly into a cache, instead of to memory.” (Sinharoy 1:22.) As stated in Claim 1[f], integrated I/O controllers support read and write operations from IO devices through the

connecting read and write interfaces. (*Id.* 1:22.) As such, writing data into a cache memory from an I/O hub chip, connected with PCI bridge and adapter (i.e., the IO device) involves receiving a request to write (i.e., allocation transaction) in the integrated I/O controller from the IO device.

11[b]: forwarding the allocation transaction to a caching agent of the multicore processor to cause the data to be stored into a shared cache memory of the multicore processor, the caching agent shared by a plurality of cores of the multicore processor and the IIO module,

70. In my opinion, Sinharoy teaches Claim 11[b]. As stated in Claim 1[e], cache controllers for L2 and L3 caches in POWER7 chip comprising the pools of RC and write machines in Sinharoy are a “caching agent.” (*See supra* Section V.A.1.a (1[e]).) The same caching agent performs cache coherency operations for the plurality of cores and the IIO module and is shared by the cores and the IIO module. (*See supra* Section V.A.1.a (1[e]).) For example, the same caching agent performs “core fetches, read-modify-write operations on behalf core stores, and core prefetch operations,” which are “cache coherency operations for the plurality of cores.” (*See id.*) It also performs “cache-injection” operations for the IIO module. (*See id.*) As such, the caching agent is “shared by a plurality of cores of the multicore processor and the IIO module.”

71. Furthermore, as explained in Claim 1[b], the shared L3 cache in Sinharoy is the “shared cache memory.” (*See supra* Section V.A.1.a (Claim 1[b]).) As stated in Claim 1[e], writing data into a cache memory from an I/O hub chip

(e.g., the I/O device) involves a request to write (i.e., an allocation transaction) during the cache injection operation. A POSITA would understand that Sinharoy's write machines include storage and logic capable of performing the up to ten memory writes on behalf of the I/O device. "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 1:19) during the cache injection operation, the data from the IO device can be stored directly into the shared L3 cache, instead of the memory. (*See supra* Section V.A.1.a (Claim 1[f]).) In addition, the pool of the write machines, as part of the same "caching agent," also handles "cache-injections." (Sinharoy 1:19) Thus, when the processor performs the cache injection operation to directly store data into the shared L3 cache, the operation involves forwarding the allocation transaction to the caching agent "to cause the data to be stored into a shared cache memory."

11[c]: wherein the caching agent is a single caching agent for the multicore processor and includes a plurality of distributed portions each associated with a corresponding one of the plurality of cores; and

72. For the reasons stated in claim 1[g] *supra*, in my opinion, Sinharoy teaches Claim 11[c].

11[d]: directly storing the data of the allocation transaction into the shared cache memory of the multicore processor without sending the data to a system memory coupled to the multicore processor.

73. In my opinion, Sinharoy teaches Claim 11[d]. As discussed in Claim 1[c], by providing the memory bandwidth to the POWER7 processor chip and connecting to the processor chip through the off-chip interface and the memory buffer chip, the memory chips are “system memory coupled to the processor.” (*See supra* Section V.A.1.a (Claim 1[c]).) Furthermore, the L2 and L3 caches in Sinharoy support cache injection operation, where an “I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, **instead of** to memory.” (Sinharoy 1:17 (emphasis added); *see supra* Section V.A.1.a (Claim 1[f]).) Targeting the operation to the cache indicates the operation will be allocated for storage in the shared cache memory. And, because it is not targeted to the memory, it will not be sent to the memory. Thus, when POWER7 processor performs the cache injection operation to directly store data into the shared L3 cache (i.e., the shared cache memory) instead of to memory, it directly stores “the data of the allocation transaction into the shared cache memory of the multicore processor without sending the data to a system memory.”

c. Independent Claim 15

15[p]: A system comprising:

74. In my opinion, Sinharoy teaches Claim 15[p] to the extent the preamble limits the scope of the claim. Sinharoy discloses the structure of a single POWER7 processor chip and a system with multiple POWER7 processor chips. (Sinharoy 1:23 (“up to 32 POWER7 chips may be connected to form a single 256-way SMP system”).)

15[a]: 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,

75. In my opinion, Sinharoy teaches Claim 15[a] for the reasons stated in *supra* Section V.A.1.a (Claims 1[a]-1[e]).

15[b]: wherein the first caching agent is to receive an allocation transaction from the first IIO module and directly store data of the allocation transaction into the first shared cache memory,

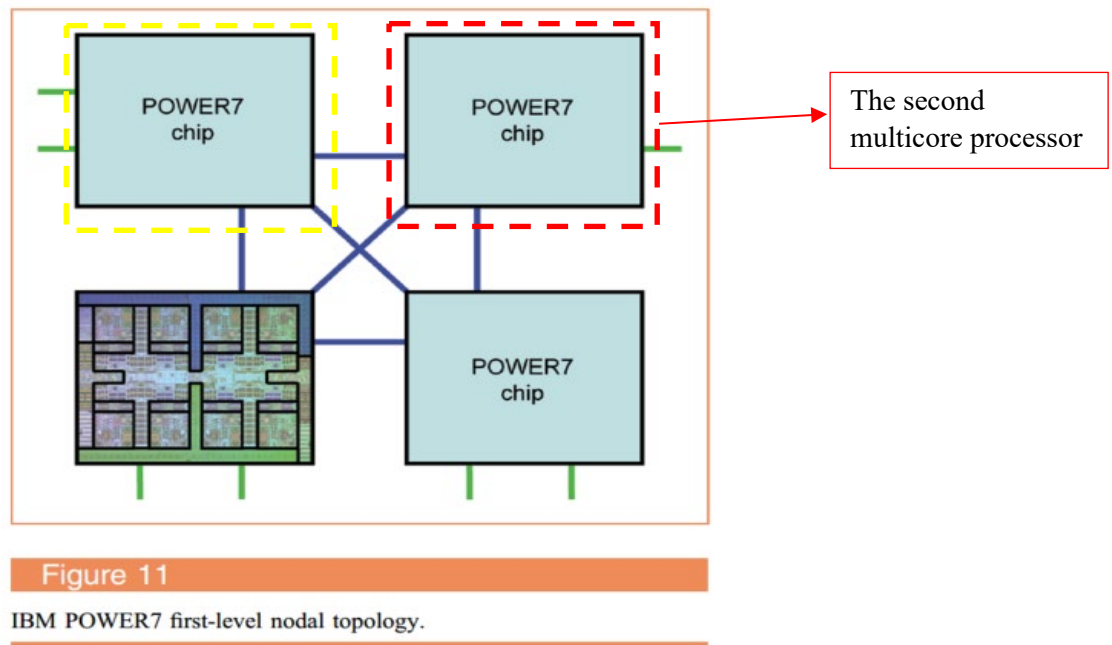
76. In my opinion, Sinharoy teaches Claim 15[b] for the reasons stated in *supra* Section V.A.1.a (Claim 1[f]).

15[c]: 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,

77. In my opinion, Sinharoy teaches Claim 15[c] for the reasons stated in *supra* Section V.A.1.a (Claim 1[g]).

15[d]: 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

78. In my opinion, Sinharoy teaches Claim 15[d] for similar reasons stated in *supra* Section V.A.1.a (Claims 1[a]-1[e]). For example, Sinharoy discloses an exemplary system which combines up to four POWER7 chips. (Sinharoy 1:23.) The second POWER7 chip in the system annotated below is the “second multicore processor” with the same structure as the “first multicore processor.”



(Sinharoy, Figure 11 (annotated).)

15[e]: a peripheral controller coupled to at least one of the first and second multicore processors.

79. In my opinion, Sinharoy teaches Claim 15[e]. Sinharoy discloses in one exemplary system, the four POWER7 processor chips is connected to cluster interconnect chip. (Sinharoy 1:25.) Each cluster interconnect chip incorporates up to three $16 \times$ PCI Express** Gen2 I/O subsystem interfaces. (*Id.*) The cluster interconnect chip with PCI (peripheral component interconnect) Express interfaces is the “peripheral controller coupled to at least one of the first and second multicore processors.”

80. A POSITA would understand that peripheral controllers are well known in the art. Such controllers include devices to interface to storage devices (e.g. disk and flash memory), communications controllers (e.g. for Ethernet networks) and user interface devices (e.g. display devices, keyboards and mice). Controllers are often developed and sold by third parties and typically include hardware and firmware to operate the device and to send data between memory and the peripheral device over an industry standard bus such as PCI or PCI Express. The PCI Express Gen2 I/O subsystem interface described at Sinharoy 1:25 would interface to these peripheral controllers.

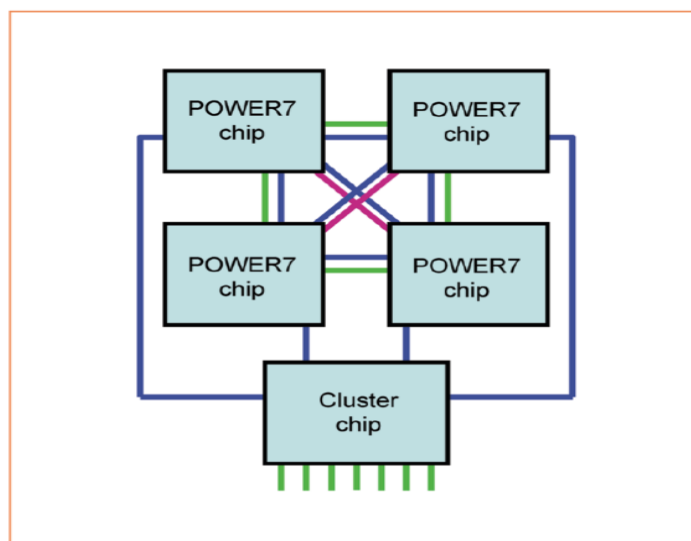


Figure 13

Cluster system first-level coherent nodal topology.

(Sinharoy, Figure 13.)

d. Dependent Claim 3

3: The processor of claim 1, wherein the caching agent includes a table of requests to store pending requests for the plurality of cores and the I/O module.

81. In my opinion, Sinharoy teaches Claim 3. As stated in Claim 1[e], the centralized cache controllers for L2 and L3 caches including the pool of RC and write machines are a “caching agent.” (*See supra* Section V.A.1.a (Claim 1[e]).) The RC machines manage “core fetches, read-modify-write operations, on behalf of core stores, and core prefetch operations.” (Sinharoy 1:18.) These operations involve incoming or pending requests (1) to read, (2) to write, or (3) to store data for the cores. In particular, the eight RC machines in the L2 cache controller manage “core fetches and stores that hit the L2 cache.” (Sinharoy 1:18.) The 24

“lightweight” RC machines in the L3 cache controller manage “prefetches and stores that miss the L2 cache.” (*Id.*)

82. Furthermore, because RC machines “aggressively reorder **storage operations** in order to fully exploit the relaxed rules prescribed by the PowerPC storage architecture,” (Sinharoy 1:18 (emphasis added)), they include temporary storages for those incoming requests and thus, are “tables of requests to store pending requests for the plurality of cores.” A POSITA would have understood that they are TORs, because they include buffers or temporary storages for storing incoming or pending requests.⁸

83. In addition, the write machines in the same “caching agent” can perform certain write operations, such as “cache-injections.” (Sinharoy 1:19; *see supra* Section V.A.1.a (Claim 1[e]).) In the I/O subsystem section, Sinharoy discloses that integrated I/O controllers (i.e., the IIO modules) also “support cache injection, which enable DMA write traffic to be written directly into a cache, instead of a memory.” (Sinharoy 1:22.) These write operations include incoming or pending write requests for the IIO module. When performing cache injection

⁸ This is consistent with the disclosure of the ’895 Patent. It discloses that a “table of requests (TOR)” may be a buffer or other temporary storage for holding incoming requests.” (’895 Patent, 2:51-53.)

operation into the L3 cache, the pool of ten write machines for the L3 cache includes temporary storages for those incoming write requests for the IIO module and thus, are “tables of requests to store pending requests for [] the IIO module.” Therefore, the RC machines and write machines within the “caching agent” in Sinharoy are “table[s] of requests to store pending requests for the plurality of cores and the IIO module.”

e. Dependent Claim 4

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

84. In my opinion, Sinharoy teaches Claim 4. Sinharoy discloses the L2 and L3 caches support a cache injection operation. (Sinharoy 1:17.) For example, if 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.” (*Id.*) The write transaction (i.e., installing the data into the local L3 region) is a “posted transaction.”⁹ As such, the

⁹ This is consistent with the disclosure in the ’895 Patent. It discloses the table of requests may include at least one “dedicated resource for post transactions.” (’895 Patent, 2:53-55.) “In one embodiment, dedicated resource 225 may be a single entry of the queue.”(*Id.*, 2:55-57.) “Generally, a posted transaction is a transaction which when sent by a source is considered complete by the source and the source does not

“dedicated entry” includes a single entry of the cache queue within the temporary storage or buffer (i.e., “TOR”).

85. Furthermore, as stated in Claim 3, when performing cache injection operation into the L3 cache, the pool of ten write machines for the L3 cache includes temporary storage for those write transactions. (*See supra* Section V.A.1.d (Claim 3).) Because write machines within the caching agent of Sinharoy have temporary storage, they include dedicated queues or resources (i.e., dedicated entry) to store such a write transaction (i.e., posted transaction) when performing the cache injection.

86. A POSITA would understand that I/O devices benefit from buffering because a device may wish to start or continue an inbound transfer before the cache controller has completed the task of updating the cache storage locations for to allow updates to those cache lines. For this reason, the write state machines include some amount of storage, such as a first-in-first-out (FIFO) queue to absorb the incoming data before the cache is ready to begin the transfer. A queue or buffer is required to support prefetching because the idea of prefetching is that information is retrieved before it is ready to be sent to memory. Sinharoy states

receive a completion or other confirmation message regarding the transaction. One such example of a posted transaction may be a write transaction.” (*Id.*, 5:12-17.)

that “[w]hen **prefetched data** is staged to the L3 cache, it is managed by a pool of ten write machines.” (Sinharoy 1:19, emphasis added) Because the temporary storages include at least one entry for a write transaction (i.e., a posted transaction), the pool of write machines are “tables of requests includ[ing] at least one dedicated entry to store a posted transaction.”

f. Dependent Claim 5

5: The processor 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.

87. In my opinion, Sinharoy teaches Claim 5. As stated in Claim 1[b], the shared L3 cache is the “shared cache memory.” (*See supra* Section V.A.1.a (Claim 1[b]).) The L3 cache in POWER7 processor is the last level cache (LLC). (*See* Sinharoy, Abstract, FIG. 1.)

88. The '895 Patent discloses in FIG. 6 that 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.” ('895 Patent 6:33-35.) In particular, it discloses that “the various cores may be coupled to each other a

shared cache memory **formed** of a plurality of units of a LLC 740₀-740_n via caching agent 715₀-715_n.’’¹⁰ (*Id.* 6:42-45 (emphasis added).)

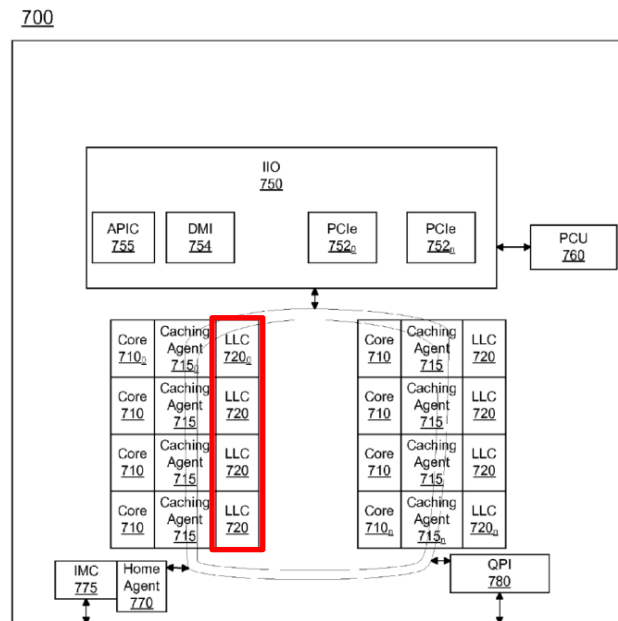


FIG. 6

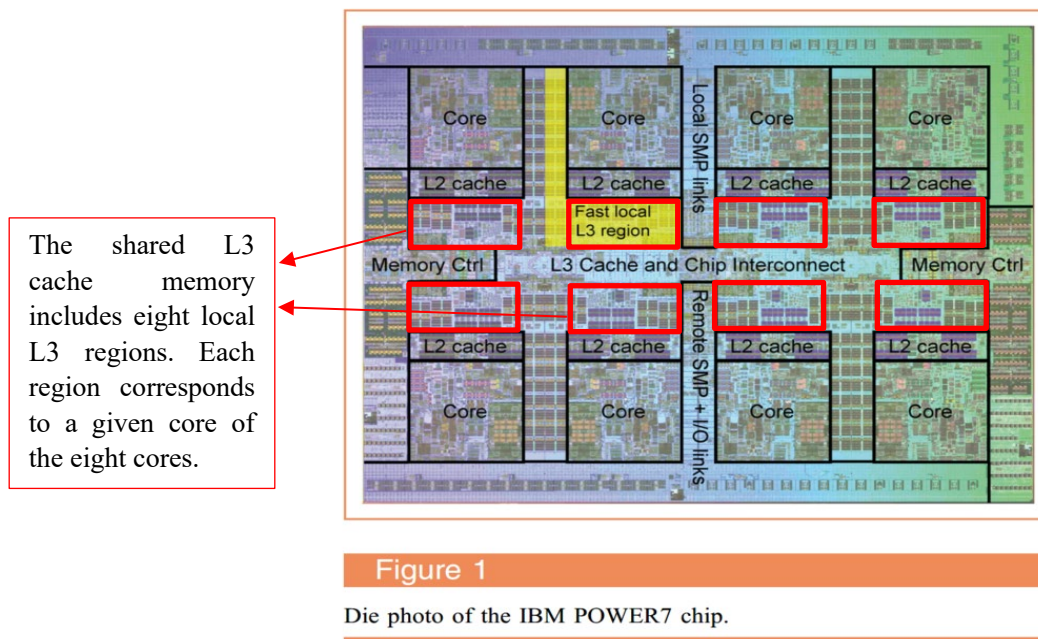
(’895 Patent, FIG. 6 (annotated))

89. Sinharoy discloses a three-level cache where L3 is the last level cache (LLC). “The memory subsystem contains three levels of on-chip cache with SOI imbedded dynamic random access memory (DRAM) devices used as the **last level of cache** [LLC].” (Sinharoy 1:1, emphasis added.) The shared L3 cache in Sinharoy (i.e., the shared cache memory) includes a plurality of distributed portions, as it is composed of “eight 4-MB L3 regions.” (Sinharoy 1:19.) Each L3

¹⁰ Note that the LLC is labeled 720 in Fig. 6 and at 6:35, and 740 at 6:44 and 6:45.

My analysis assumes that these were meant to refer to the same elements.

region is tightly coupled to the L2 cache “associated with a given core.” (*Id.*) 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 1:17.) As such, each local L3 region of the L3 cache is a “distributed portion” of the shared cache memory, which corresponds to one of the eight cores of the POWER7 processor. (See Sinharoy Figure 1.)



(Sinharoy, Figure 1 (annotated).)

g. Dependent Claim 6

6: *The processor of claim 1, wherein the processor is to directly store the data of the allocation transaction into the shared cache memory without initiation of a memory transaction to the memory.*

90. In my opinion, Sinharoy teaches Claim 6. As discussed in Claim 1[f], the L2 and L3 caches in Sinharoy support the “cache injection” operation. (See

supra Section V.A.1.a (Claim 1[f]).) In particular, an “I/O device performing a direct memory access (DMA) write operation may target the operation to the cache, **instead of to memory.**” (Sinharoy 1:17 (emphasis added).) In the I/O subsystem section, Sinharoy further discloses that integrated I/O controllers (i.e., IIO modules) support such an operation, “which enables DMA write traffic to be written directly into a cache, instead of to memory.” (Sinharoy 1:22.) The write traffic from the IO device directly into a cache, instead of into memory, does not initiate “a memory transaction to the memory,” because it does not write into the memory. The integrated I/O controllers support read and write operations from the I/O hub chips through the connecting read and write interfaces. (*Id.* 1:22.) When the POWER7 processor performs the cache injection operation and directly writes data into the shared L3 cache (i.e., shared cache memory), the processor is “is to directly store the data of the allocation transaction into the shared cache memory without initiation of a memory transaction to the memory.”

h. Dependent Claim 7

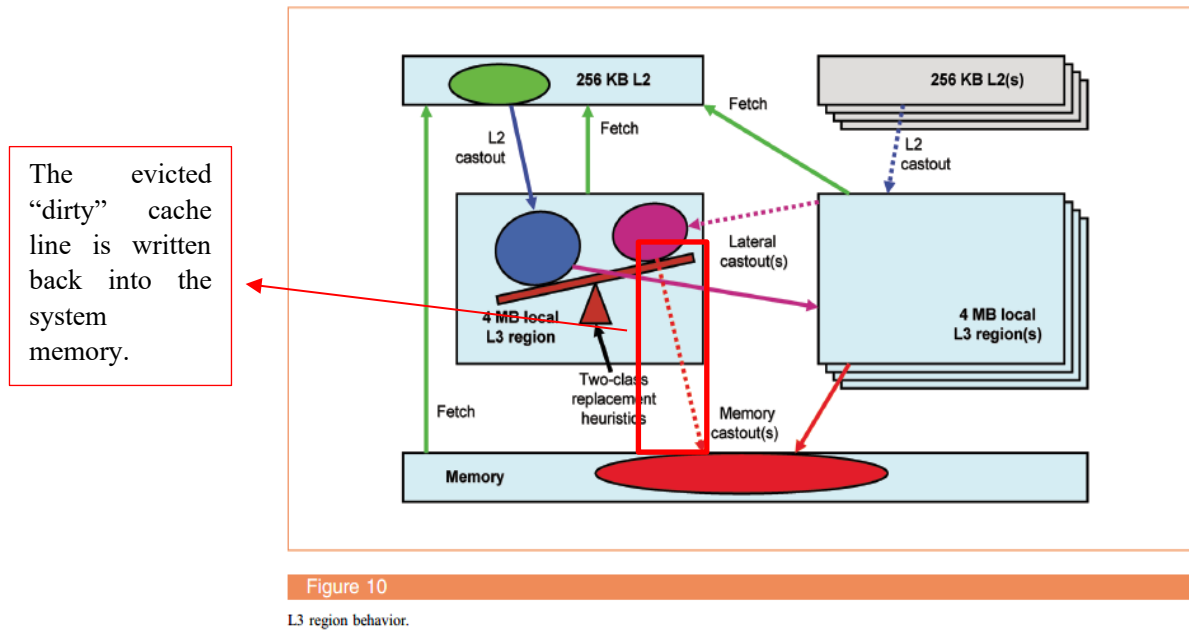
7: The processor of claim 6, wherein the processor is to write the data of the allocation transaction to the memory responsive to eviction of a cache line including the data.

91. In my opinion, Sinharoy teaches Claim 7. A cache line is the smallest portion of data that can be mapped into a cache. For example, the POWER7 processor can move a given cache line among the L2 and L3 caches. (*See* Sinharoy

1:18.) For example, when “prefetched data is staged to the L3 cache,” it is managed by a pool of “ten write machines,” which is part of the caching agent. (Sinharoy 1:19.) “Each write machines has an associated cast-out machine to manage the “evictions of lines” displaced by writes.” (*Id.*) The “evictions of lines” are evictions of cache lines including corresponding data of those evicted cache lines. Managing the eviction of lines involves writing the data of the allocation transaction to memory. It would be obvious to a POSITA that the cast-out machine is the part of the processor that writes data to memory in response to the allocation transaction (eviction).

92. In Figure 10, Sinharoy discloses the behaviors among L3 regions, the L2 cache, and the system memory. (Sinharoy 1:20-21.) For example, 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.” (*Id.* 1:21.) A “dirty” line is a cache line that has been modified and needs to be written back to the memory. A POSITA would understand that a dirty cache line is a cache line which contains data that has been modified (updated), but those updates have not yet been sent to main memory. Dirty cache lines must at some point be written back to main memory (unless it is known that the dirty lines contain information that will never be reused). Dirty lines are written to memory to prevent the updates

from being lost when that cache line is later used to hold other information. When the POWER7 processor performs the cache injection operation and a cache line is evicted from the L3 region due to its dirty state, POWER7 processor is to write the data of the cache line into the system memory, as depicted in Figure 10 below.



(Sinharoy, Figure 10 (annotated).)

i. Dependent Claim 12

12: *The non-transitory machine-readable medium of claim 11, wherein the method further comprises 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 the system memory.*

93. In my opinion, Sinharoy teaches Claim 12. Sinharoy discloses 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

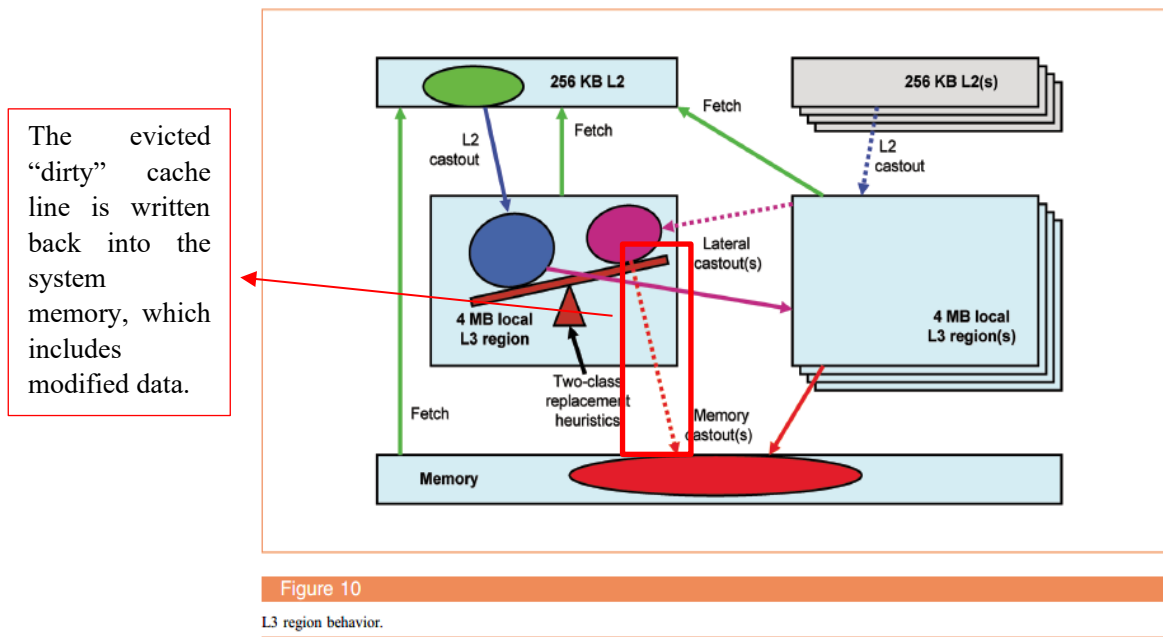
actively sharing them but also destroying copies as they fall into disuse in order to preserve capacity.” (Sinharoy, 1:17.) When a core actively utilizes a region of L2 with reduced latency, that means that at least one core is accessing the data a number of times without sending the transaction to system memory. A POSITA would understand that such a mechanism might be used by processors accessing seldomly updated information such as a look-up-table. It may be beneficial for several processors to have fast access to the table by bringing it into their caches, but because the table is unchanging, the memory copy remains valid and there is no need to write the cached copies back to memory. Similarly, when POWER7 processor performs the cache injection operation and the data is stored in the shared L3 cache, the data can be accessed a number of times by at least one of the cores without initiating a memory transaction. “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 1:17.) A POSITA would have understood that the purpose of a cache is to access data a number of times without sending a transaction to memory.¹¹

¹¹ This is consistent with the disclosure in the ’895 Patent. It discloses that “[w]ith the data now present in the LLC, it can be accessed a number of times for read and

j. Dependent Claim 13

13: *The non-transitory machine-readable medium of claim 11, wherein the method further comprises sending the data to the system memory from the shared cache memory if a cache line including the data is evicted from the shared cache memory, when the cache line includes modified data.*

94. In my opinion, Sinharoy teaches Claim 13. For the same reasons stated in Claim 7, when the POWER7 processor performs the cache injection operation and a cache line is evicted from the L3 region due to its dirty state, the data is evicted from the L3 cache and includes “modified data.” A POSITA would have understood that a “dirty” line is a cache line that has been modified and needs to be written back to the memory.



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.” (’895 Patent, 4:6-9.)

(Sinharoy, Figure 10 (annotated)))

k. Dependent Claim 16

16: The system of claim 15, 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.

95. In my opinion, Sinharoy teaches Claim 16 for the same reasons stated in *supra* Section V.A.1.d-e (claims 3-4).

l. Dependent Claim 17

17: The system of claim 15, 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.

96. In my opinion, Sinharoy teaches Claim 17 for the reasons stated in *supra* Section V.A.1.i (claim 12).

2. Ground 2 – Sinharoy in view of Xu Renders Obvious Claims 2 and 10

a. Overview of Xu

97. I have reviewed Xu. Xu, titled “*Explorations of Optimal Core and Cache Placements for Chip Multiprocessor*,” was published on November 14, 2011. I understand Xu is prior art to the ’895 Patent at least under at least § 102(a). In the Introduction Section of Xu, it discloses that using on-chip ring interconnect had been well known in the art. (Xu at 1.) My review of the ’895 Patent file history shows that Xu was not considered during prosecution of the ’895 Patent.

b. Dependent Claim 2

2: The processor of claim 1, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.

98. In my opinion, Sinharoy in view of Xu teaches Claim 2. Xu discloses that IBM POWER7, Intel Sandy Bridge, Nehalem-EX, and Westmere-EX processors rely on a ring interconnect as the “on-chip interconnect.” (*Id.*) In Table 1, Xu explicitly discloses that the on-chip interconnect for the **IBM POWER7 processor chip** [i.e., processor in Sinharoy] is a ring interconnect. (*Id.* at 2 (emphasis added).) Because Xu and Sinharoy are in the same field of study and in fact describes the same POWER7 processor, a POSITA who seeks to design a processor system like POWER7 would be motivated to rely on information from both references. A POSITA would have readily understood that the same POWER7 processor chip uses a ring interconnect as the on-chip interconnect, because Xu discloses the same POWER 7 processor chip in that section.

TABLE I: Comparison of different Chip Multiprocessors.

CMP	Cores	Last Level Cache	Number of Transistors	Process	Die Size	On-chip Interconnect
AMD MagnyCours [3]	12 (2×6)	12MB (2×6MB)	1,808M (2×904M)	45nm	692mm ² (2×346mm ²)	Crossbar
IBM Power 7 [6]	8	32MB	1,200M	45nm	567mm²	Ring
Intel Nehalem [1]	4	8MB	731M	45nm	263mm ²	Crossbar
Intel Westmere [1]	6	12MB	1,170M	32nm	239mm ²	Crossbar
Intel SandyBridge [4]	4	8MB	995M	32nm	216mm ²	Ring
Intel Nehalem-EX [1]	8	24MB	2,300M	45nm	684mm ²	Ring
Intel Westmere-EX [4]	10	30MB	2,600M	32nm	513mm ²	Ring
Sun SPARC T3 [5]	16	6MB	1,000M	40nm	377mm ²	Hierarchical Crossbar

(*Id.*, Table 1 (annotated).)

99. In the “On-Chip Interconnect” section of Sinharoy, Sinharoy discloses that 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.” (Sinharoy 1:23.) The on-chip ring interconnect of POWER7 processor connect the eight cores with their associated L2 caches and local L3 regions, two memory controllers, and two I/O controllers. (See Sinharoy 1:22.) The on-chip interconnect is 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.” (*Id.* 1:22-23.) 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.” (*Id.* 1:23.) Thus, the ring interconnect couples the centralized cache controllers for the L2 and L3 caches.

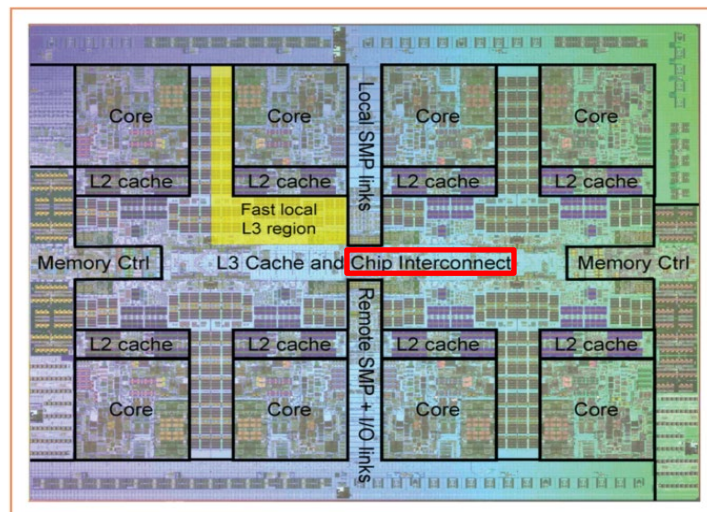


Figure 1

Die photo of the IBM POWER7 chip.

(Sinharoy, Figure 1 (annotated).)

100. As discussed above, POWER7 processor uses a ring interconnect as the on-chip interconnect. As stated above, the cache controllers for each local L3 cache regions are the “plurality of distributed portions of the caching agent.” (*See supra* Section V.A.1.a (Claim 1[g]).) The on-chip ring interconnect is 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.” (*Id.* 1:22-23.) 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.” (*Id.* 1:23.) Thus, the ring interconnect in POWER 7 connects the centralized cache controllers for each local L3 cache regions.

c. Dependent Claim 10

10: The processor of claim 1, further comprising a ring interconnect to couple the plurality of cores and the shared cache memory via a caching agent.

101. In my opinion, Sinharoy in view of Xu teaches Claim 10. The on-chip interconnect is 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.” (*Id.* 1:22-23.) 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.” (*Id.* 1:23; *see also* Sinharoy Figure 1) Because the on-chip ring interconnect runs across the center of the POWER7 chip, a POSITA would have readily understood that the ring on-chip interconnect connects the eight cores and the shared L3 cache, as well as the cache controllers for the L2 and L3 caches including the pool of RC and write machines (i.e., the caching agent).

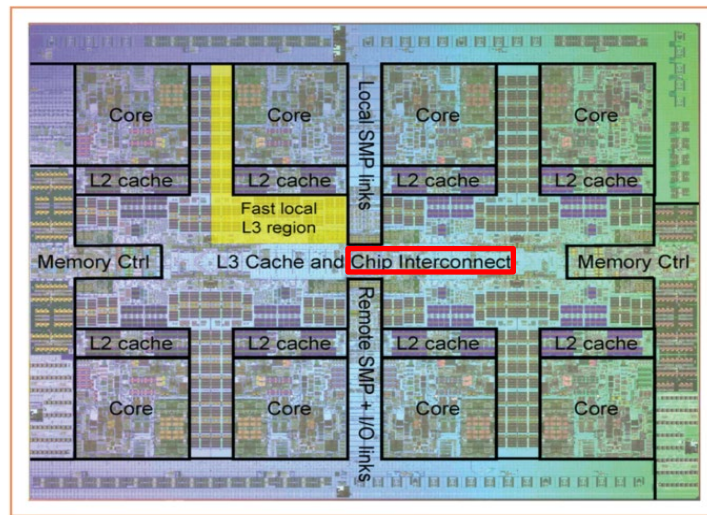


Figure 1

Die photo of the IBM POWER7 chip.

(Sinharoy, Figure 1 (annotated).)

3. **Ground 3 – Sinharoy in view of Tang Renders Obvious Claims 8-9 and 14**

a. **Overview of Tang**

102. I have reviewed Tang. Tang, titled “*DMA Cache: Using On-Chip Storage to Architecturally Separate I/O Data from CPU Data for Improving I/O Performance*,” was published and has been available since January 14, 2010. I understand Tang is prior art to the ’895 Patent at least under at least pre-AIA §

102(a) and § 102(b). My review of the '895 Patent file history shows that Tang was not considered during prosecution of the '895 Patent. A POSITA would have been motivated to combine Sinharoy and Tang because they are both directed to multicore processor systems, in particular, both discussing cache coherence protocol. Tang shows additional details on how the cache coherence protocol in Sinharoy could be implemented.

b. Dependent Claims 8 and 14

8: The processor 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 the IO device responsive to receipt of a snoop request in the caching agent for the cache line from another agent.

14: The non-transitory machine-readable medium of claim 11, wherein the method further comprises 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.

103. In my opinion, Sinharoy in view of Tang teaches Claims 8 and 14.

104. Sinharoy discloses that the caching agent, comprising the centralized cache controllers for the L2 and L3 caches including RC and write machines, receives snooping requests for cache lines from other agents. (Sinharoy, 1:22-23, 1:19-1:20.) Sinharoy discloses that the caching agent handles “storage accesses,” such as requests to write data from an I/O controller (and thereby owned by the I/O controller) into a cache line during supported cache injection operations. (*Id.*, 1:17-19, 1:22.) Sinharoy further discloses 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.” (*Id.*, 1:19, 1:22-23.) A POSITA would have understood where a cache line owned by an I/O controller (i.e., IIO module) during a cache injection operation “miss” the L2 and L3 caches associated with a particular core, another core (i.e., another agent) and its associated remote L3 region can send a “snoop request” to the caching agent for that cache line.

105. Furthermore, Sinharoy discloses 13-cache-state protocol used for cache “coherent operations,” in the POWER7 multicore processor. (Sinharoy 1:17, Table 2.) In Table 2, Sinharoy discloses 13-cache-state protocol used for cache “coherent operations,” such as “TE” (formerly exclusive, now shared) and “T” (formerly modified bias toward sharing, now shared)¹². (Sinharoy 1:17.) As

¹² As discussed in Technology Background, “Modified state” means “no other processor has a copy of the data block. The copy of the data in main memory is incorrect.” “Owned state” means “only one processor can be in the owned state. All other processors can have a copy of the most recent in the shared state. The copy of the data in main memory can be incorrect.” “Invalid state” means “either main memory or another processor cache can have valid copies of the data.”

described in the Technology Background *supra* about cache coherence, a POSITA would have readily understood that the cache states in Sinharoy are a minor variation of the well-known standard MOESI protocol in the art. Sinharoy includes the “T” and “TE” states in which a cache has update authority and the cache line is potentially shared. These state have substantially the same function as the “Owned” state in a MOESI protocol. Furthermore, Sinharoy include three invalid states (I “Invalid”, IG “Invalid, cached scope-state” and IN “Invalid, scope predictor”) that serve substantially the same function as the “Invalid” state in the MOESI protocol. (*See Id.*) A POSITA would have understood that a cache state change in Sinharoy from T or TE states to one of the Invalid states is a “drop ownership indication” because in the T or TE states, the cache is allowed to update the cache line, indicating that the cache line is owned by that cache. If the I/O controller wishes to use that line for other information, such as a prefetch, ownership must first be dropped by causing that cache to transition to one of the invalid states.¹³

¹³ This is consistent with the disclosure in the ’895 Patent. It discloses that “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

106. Tang, in the same field of art as Sinharoy, discloses a multicore-processor system with shared cache with distributed portions. (Tang § 3.2.2.) Because Tang and Sinharoy are in the same field of art, a POSITA would have been motivated to look to the cache-state transitions during cache coherence operations in Tang to understand the cache states listed in Sinharoy. A POSITA would have understood that Tang provides details on similar cache states discussed in Sinharoy, which would be common across multi-core processors implementing the cache coherence protocols described in Sinharoy and Tang. Furthermore, a POSITA would recognized that Tang and Sinharoy are just using slightly different terminology for the same well-known cache-state transitions, and a POSITA would have readily understood how the details on cache state transitions described in Tang apply to the corresponding cache state changes in Sinharoy. Moreover, Tang

indicator as well as cache coherency state information, in some embodiments. When such a 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). Note that this indication may be because it is likely that the ownership of the cache line by the IIO module is responsive to a prefetch by a coupled IO device and thus the data with regard to the IO device is of a speculative nature and is not yet needed.” (’895 Patent 4:46-59.)

discloses cache-state transitions during cache coherence operations. (*Id.* § 3.1.3.) Tang discloses when cache B tries to read the cache line (i.e., data) from cache A, cache B¹⁴ issues a “Probe-Write hit” to cache A, and when cache A receives the “Probe-Write hit”, it changes the data block with “Owned State” to “Invalid State”. (See Tang § 3.1.3; Figure 8.) A POSITA would have understood that the cache controllers (i.e., caching agent) for cache A and B manage the “Probe-Write hit” (i.e., a snoop request) and cache-state transition. A POSITA would have understood that in other words, the caching agent, in receipt of a snoop request,

¹⁴ Cache B in Tang is a DMA cache. Tang discloses “we have refined the MOESI protocol [3] and the ESI protocol [16] for WB and WT policies respectively, i.e., the IO-MOESI (Figure 8) and the IO-ESI (Figure 9) protocols. Essentially, the refined cache coherence protocols for the DMA Cache are the same as the original MOESI/ESI protocols.” (Tang § 3.1.3.) Tang further discloses the “only difference between the IO-MOESI/IO-ESI and the original protocols is exchanging the local sources and the probe sources of the state transitions (see Table 3) . . . Therefore, the DMA Cache can be viewed as a remote cache.” (*Id.*) The “DMA cache” cache-state transitions disclosed in Tang would be understood by a POSITA to apply equally to cache state transitions in other remote cache systems, such as those in Sinharoy.

changes the cache state from “owned” to “invalid,” which involves “send[ing] a drop ownership indication.”

Table 2 IBM POWER7 cache states.

<i>State</i>	<i>Description</i>	<i>Authority</i>	<i>Sharers and scope</i>	<i>Source data</i>	<i>Data cast-out</i>	<i>Scope cast-out</i>
I	Invalid	None	N/A	N/A	N/A	None
ID	Deleted, do not allocate	None	N/A	N/A	N/A	None
S	Shared	Read	Yes, scope unknown	No	No	None
SL	Shared, local data source	Read	Yes, scope unknown	At request	No	None
T	Formerly MU, now shared	Update	Yes, probably global	If notified	Yes	Required, global
TE	Formerly ME, now shared	Update	Yes, probably global	If notified	No	Required, global
M	Modified, avoid sharing	Update	No	At request	Yes	Optional, local
ME	Exclusive	Update	No	At request	No	None
MU	Modified, bias toward sharing	Update	No	At request	Yes	Optional, local
IG	Invalid, cached scope-state	None	N/A, probably global copies	N/A	N/A	Required, global
IN	Invalid, scope predictor	None	N/A, probably local copies	N/A	N/A	None
TN	Formerly MU, now shared	Update	Yes, local	If notified	Yes	Optional, local
TEN	Formerly ME, now shared	Update	Yes, local	If notified	No	None

(Sinharoy, 1:17, Table 2.)

107. Tang discloses cache coherence protocol used in multicore processors. (See Tang, § 3.1.3 (“we have refined the MOESI protocol [3] and the ESI protocol [16] for WB and WT policies respectively, i.e., the IO-MOESI (Figure 8) and the IO-ESI¹⁵ (Figure 9) protocols. Essentially, the refined cache coherence protocols

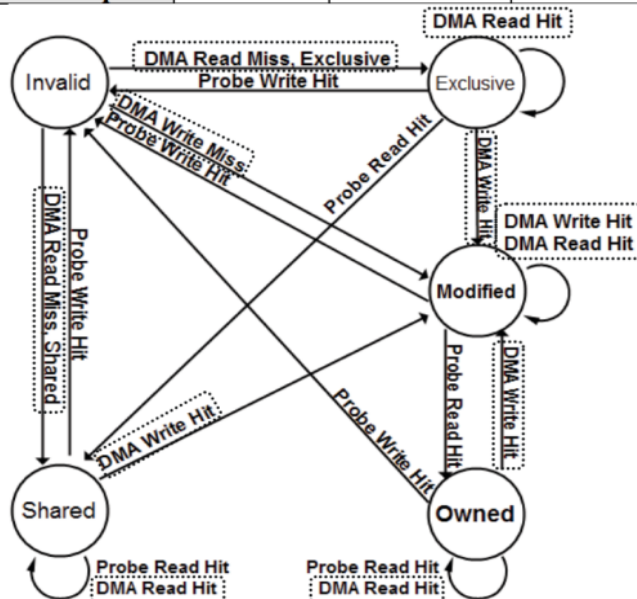
¹⁵ The ESI protocol (Exclusive/Shared/Invalid), and its extension by Tang to I/O is a simple 3-state protocol used in write-through caches. Neither the ESI or Tang’s extension to I/O (IO-ESI) are part of this analysis. Instead, the MOESI protocol

for the DMA Cache are the same as the original MOESI/ESI protocols.”.)

Specifically, Tang discloses how the cache states change during read or write transactions. Tang further discloses that the “only difference between the IO-MOESI/IO-ESI and the original protocols is exchanging the local sources and the probe sources of the state transitions (see Table 3). For example, if a CPU generates a write request to its cache (i.e., local cache), the other CPU’s caches (i.e., remote cache) and the DMA Cache might receive the same ‘Probe-Write’ request. Therefore, **the DMA Cache can be viewed as a remote cache.**” (*Id.* (emphasis added).) As noted in Table 3 of Tang, the cache-state transitions for a remote cache are the same as the cache-state transitions in a DMA cache.

(Modified/Owned/ Exclusive/Shared/Invalid) and its I/O extension disclosed by Tang are similar to the cache states in the Sinharoy write-back caches. (See Technology Overview IIA for YI’s description of the meaning of the MOESI states.) A POSITA would be motivated to combine Sinharoy with Tang’s IO-MOESI protocol because the state transitions of Sinharoy’s caches closely follow the details of the MOESI protocol shown in Tang’s Figure 8.

	Local Cache	Remote Cache	DMA Cache
	MOESI/ESI	MOESI/ESI	IO-MOESI/IO-ESI
CPU Read Request	CPU-Read	Probe-Read	Probe-Read
CPU Write Request	CPU-Write	Probe-Write	Probe-Write
DMA Read Request	Probe-Read	Probe-Read	DMA-Read
DMA Write Request	Probe-Write	Probe-Write	DMA-Write



(Tang, § 3.1.3 Table 3 & Figure 8 (disclosing cache state changes from “Owned” to “Invalid”))

66

have a reasonable expectation of success, because the cache states are known in the art.

109. As disclosed in Tang, if cache A has a block (line) that has been modified, it holds that block in the M (Modified) state. Later, when that modified block is read by cache B, it issues a Probe Read Hit to tell cache A that it is now sharing the data and that cache B should become the exclusive owner of that block by changing its state from M (Modified) to O (Owned). Later, if cache B (or another cache) wants to write to that block, cache A will receive a “Probe-Write hit” to tell it to change the data block from the O (Owned) state” to the I (Invalid) state.” That transition from Owned to Invalid would normally require the block to be written to memory, but Tang includes additional information that is used to avoid the write back to memory when it is known that the block holds I/O data that will never be reused and can be discarded. “The replacement policy of the L1/L2 caches requires a slight modification in which the block with the “IO_Data” tag is evicted directly without being swapped into the victim buffer or the LLC. To propagate the tag, the data paths also need to add one bit. When forwarding blocks to other caches, the DMA Cache and LLC tag the data copy as “IO_Data” and “CPU_Data”, respectively. In other words, the DMA Cache is the source of “IO_Data” tag and the LLC is the source of “CPU_Data” tag.” (See Tang § 3.1.5; Figure 8.) In other words, the cache controller (i.e., caching agent) will “send a

drop ownership indication” and the Probe-Write hit from another cache is “a snoop request in the caching agent for the cache line from another agent.”

Figure 8 IO-MOESI State Transitions for DMA Cache with Write-Back Policy¹

110. As applied in Sinharoy, where a cache line owned by an I/O controller during a cache injection operation “miss” the L2 and L3 caches associated with a particular core, another core (i.e., another agent) and its associated remote L3 region can send a “snoop request” to the caching agent for that cache line in Sinharoy. When the caching agent receives the snoop request, in view of the cache-state transition in Tang, it would change the cache state for that cache line from T or TE to one of the “Invalid” states, which involves “send[ing] a drop ownership indication” to the IIO module. (*See also infra* Section V.B.1.g (Claim 8).)

c. Dependent Claim 9

9: The processor of claim 8, wherein the IO device prefetched the cache line, and another agent is one of the plurality of cores.

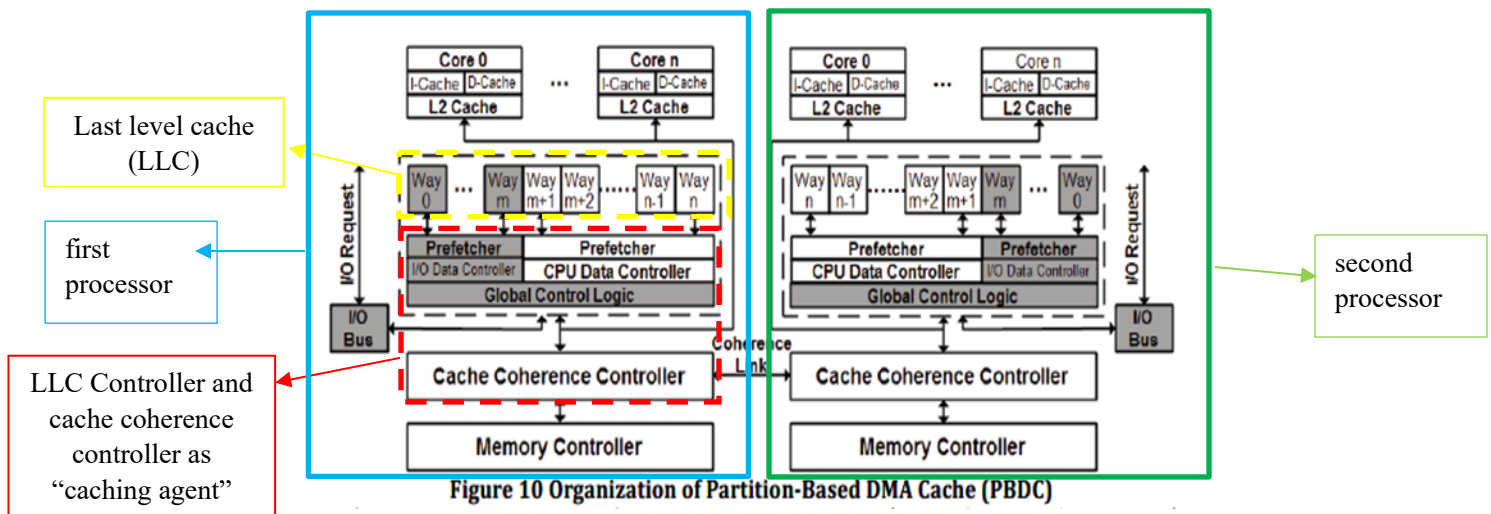
111. In my opinion, Sinharoy in view of Tang teaches Claim 9. As discussed in Claim 8, where a cache line owned by an I/O controller during a cache injection operation “miss” the L2 and L3 caches associated with a particular core, another core (i.e., another agent) and its associated remote L3 region can send a “snoop request” to the caching agent for that cache line. (*See supra* Section V.A.3.b (Claim 8).) Sinharoy discloses that the R/C machines within its cache controllers manage “core fetches, read-modify-write operations on behalf of core stores, and core prefetch operations.” (Sinharoy 1:18.) “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.” (Sinharoy 1:19.) As discussed in Claim 8 *supra*, the cache line in the IIO module will be invalidated due to the state change from T or TE state in IIO module to one of the “Invalid” states. (*See supra* Section V.A.3.b (Claim 8).) A POSITA would have understood that the cache line may have been in the T or TE state because IO device connected to the IIO module had prefetched the cache line.

B. Grounds Based on Tang

112. I have reviewed Tang. Tang is directing to a multicore processor to store I/O data into a shared cache memory. Tang was published and has been available since January 14, 2010. I understand Tang is prior art to the '895 Patent at least under at least pre-AIA § 102(a) and § 102(b).

113. My review of the '895 Patent file history shows that Tang was not considered during prosecution of the '895 Patent.

114. Tang discloses systems and methods to directly inject or store I/O data into a cache memory, because “moving I/O data in/out memory” exhausts portion of memory bandwidth as well as accounts for a considerable part of I/O operation latency. (See Tang, § 1.)



(Tang, § 3.2.1 Figure 10 (annotated).)

115. Tang describes two designs of a DMA cache, the decoupled DMA cache (DDC) and the Partition-based DMA Cache (PBDC). A POSITA would

understand that DMA stands for Direct Memory Access, and in this context refers to the ability of an I/O device to directly write data into the cache (also known as cache injection). The DDC design has a separate DMA cache while the PBDC allows DMA from an IO device to a portion of the last level cache (LLC). In the Partition-based DMA cache design (PBDC), Tang discloses a multiprocessor system using “several ways¹⁶” of the processor’s last level cache (LLC) as the “dedicated DMA cache.” (Tang, § 3.2.1.). As illustrated in Figure 10, two multicore processors are connected via a coherence link. Each processor includes multiple cores (Core 0 to Core n), an I/O bus, and a memory controller. Each processor also includes an LLC Controller (last level cache controller), which includes “Global Control Logic, IO Data Controller, CPU Data Controller, two Prefetchers, and Configuration Module” and cache coherence controller. (Tang, § 3.2.2.) The LLC Controller components

¹⁶ A POSITA would understand that caches are organized into one or more “ways” which may also be called “banks.” When a memory line is stored into a cache, the cache controller decides where to put it, often using a “least recently used” algorithm to evict the oldest line and use that storage for the new line. Each cache line can only be stored in one place in each way (bank). A 1-way cache is also called direct mapped because there is only one place each cache line can be stored. Often caches are 2, 4, or 8-way set associative.

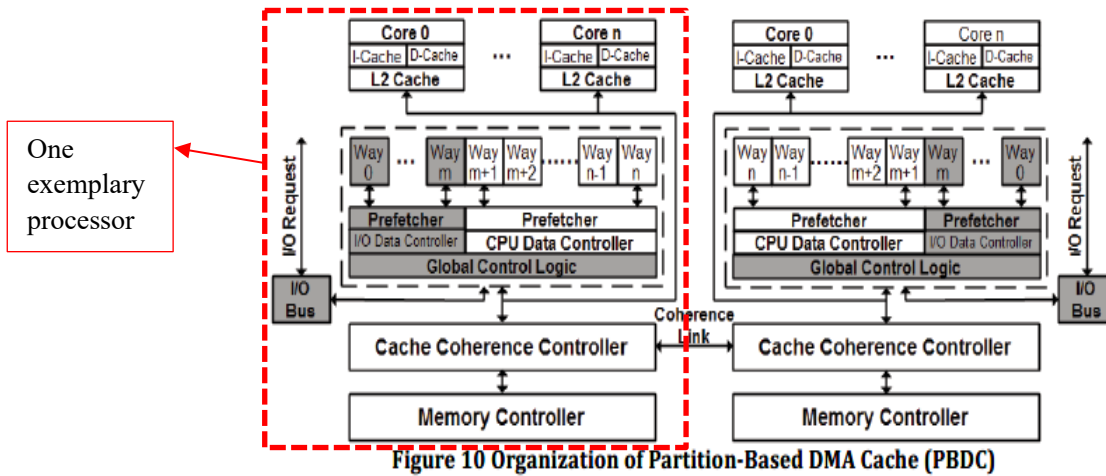
manage “write policy, cache coherence protocol, replacement policy” for the LLC and the processor. (See Tang § 3.2.2.) A POSITA would understand that “write policy” refers to the decision on where to write the data (which cache “way” or main memory) (See Tang 3.1.2), cache coherence policy refers to the over-all cache protocol (modified MOESI) (See Tang 3.1.2), and “replacement policy” refers to the decision on whether or not to write a line back to memory before invalidating the cache line. (See Tang 3.1.4) In particular, Tang discloses that I/O data from an I/O device can be written directly into the DMA cache. (See, e.g., Tang, § 2; § 3.2.2.) A POSITA would understand that in a system without cache injection, data from I/O must first be written to memory, then later retrieved from memory and copied into one or more levels of a processor cache before the incoming I/O data may be accessed by a processor core.

1. Ground 4 – Tang Anticipates or Renders Obvious Claims 1, 5-9, 11-14, 15, 17

a. Independent Claim 1

1[p]: A processor comprising:

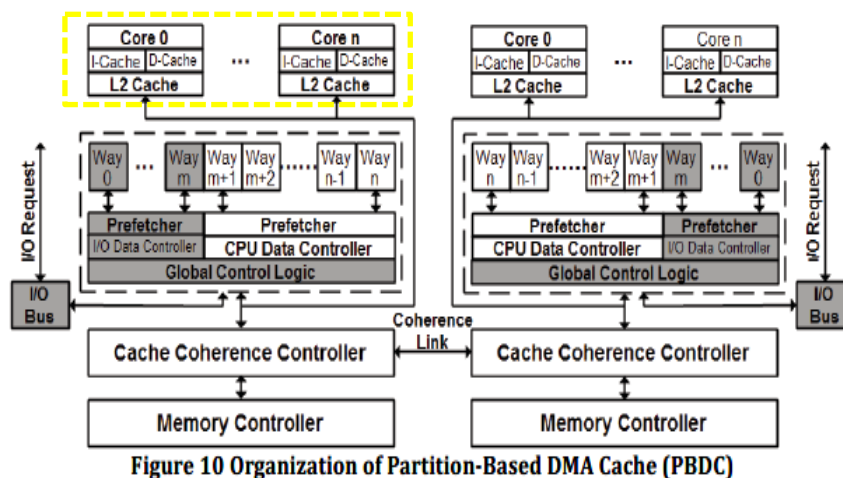
116. In my opinion, Tang teaches Claim 1[p] to the extent the preamble limits the scope of the claim. Tang discloses two DMA cache designs for “processors.” (Tang, § 3.) In the Partition-Based DMA Cache (PBDC), Figure 10 discloses a “multiprocessor” system with two processors.



(Tang, Figure 10 (annotated)).

1[a]: a plurality of cores,

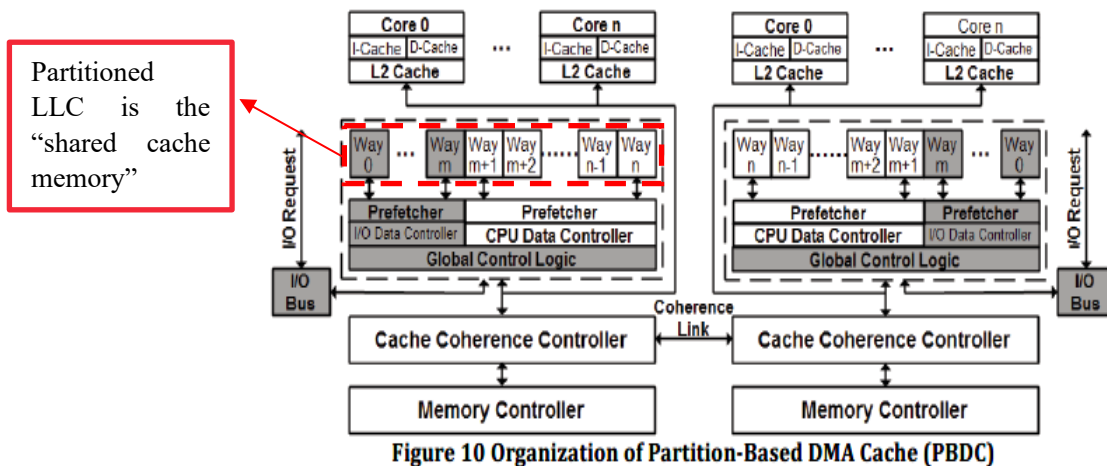
117. In my opinion, Tang teaches Claim 1[a]. Figure 10 of Tang discloses a Partition-Based DMA Cache (PBDC) system in a multiprocessor system, including a plurality of cores—namely cores Core 0 through Core n. (Tang, § 3.2.1.) As shown in Figure 10, the plurality of cores in Tang are identified as “Core 0” through “Core n.” (*Id.*, Figure 10).



(Tang, Figure 10 (annotated)).

1[b]: a shared cache memory,

118. In my opinion, Tang teaches Claim 1[b]. Figure 10 of Tang discloses a Partition-Based DMA Cache (PBDC) system including using the “processor’s LLC” as a “dedicated DMA cache.” (Tang, § 3.2.1). The LLC can be partitioned into several ways to store I/O data and CPU data. (Tang, § 3.2.1 (“I/O data ways (Way₀~Way_m) and the CPU data ways (Way_{m+1}~Way_n)”). The partitioned LLC (Way₀~Way_n) use the same “cache coherence protocol” as the same design of Decoupled DMA cache (DDC). (Tang, § 3.2.1, § 3.1.2.) Furthermore, the partitioned LLC (Way₀~Way_n) is shared among the Core 0 to Core n. Thus, the partitioned LLC is the “shared cache memory.”



(*Id.*, Figure 10 (annotated).)

1[c]: a memory controller to interface with a memory coupled to the processor,

119. In my opinion, Tang teaches Claim 1[c]. Figure 10 of Tang discloses a PBDC system including a memory controller.” (*Id.*, 1:21.)

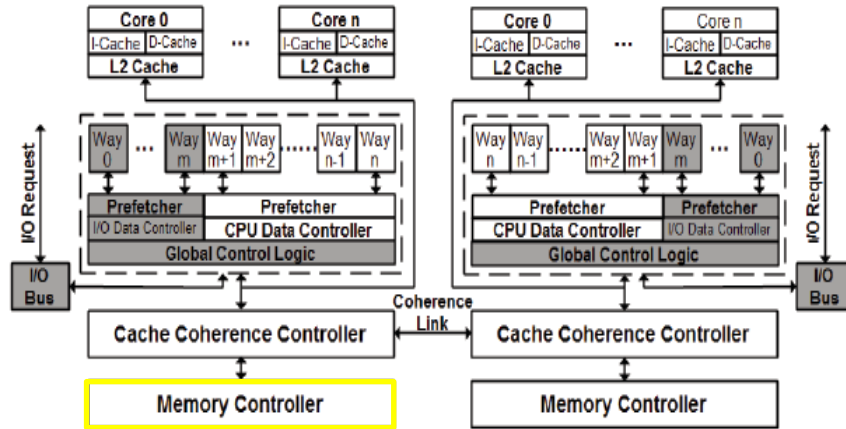


Figure 10 Organization of Partition-Based DMA Cache (PBDC)

(Tang, Figure 10 (annotated).)

120. Tang discloses that the components for the DMA operation includes a system memory coupled to the processor. (FIG. 1.)

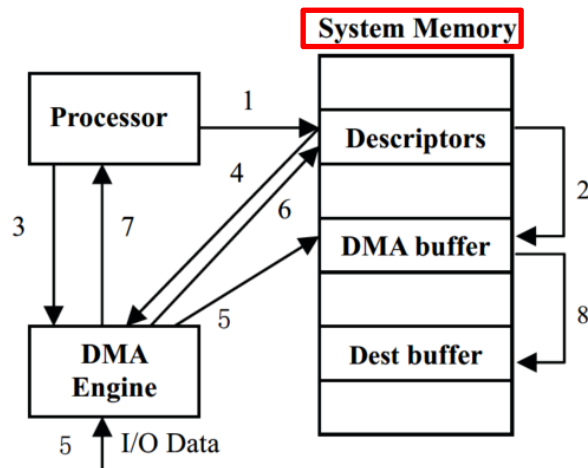


Figure 1 The Interactions of processor, memory and DMA engine in DMA operation

(Tang, FIG. 1 (annotated).)

121. Furthermore, Tang discloses the memory controller interacts with the DMA cache and the system memory. (Tang § 2.1.) For example, Tang discloses the DMA data transfer flow in the “DMA-Produce-CPU-Consume (DPCC)”

direction. (*Id.*) For example, when the CPU consumes the I/O data, the compulsory misses will take place and trigger the “memory read requests” to the memory controller. (*Id.*) The “memory read requests” and “memory write requests” are interfacing with the system memory. (Tang FIG. 2.)

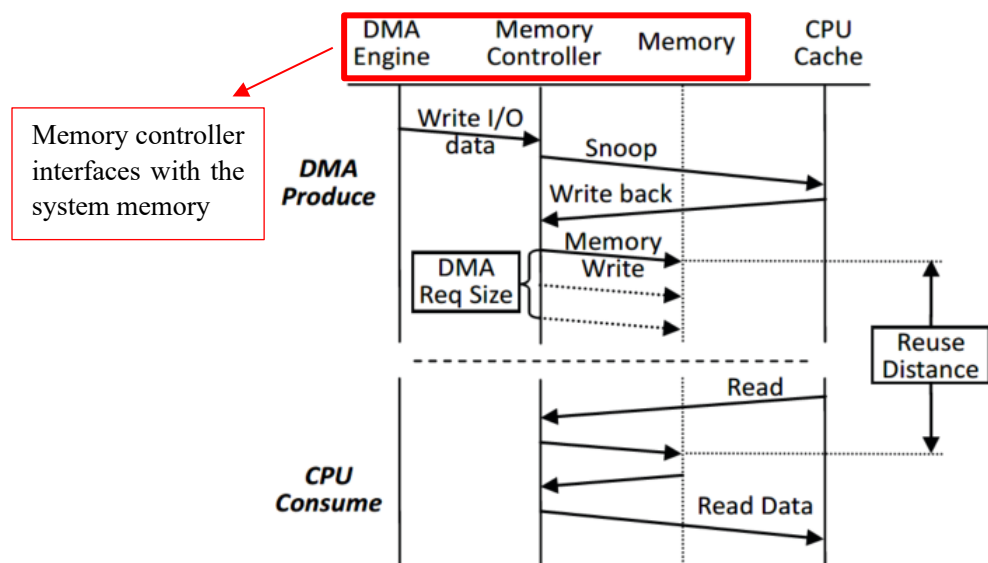


Figure 2 DMA data transfer flow in the “DMA-Producer-CPU-Consumer (DPCC)” direction

(Tang, FIG. 2 (annotated).)

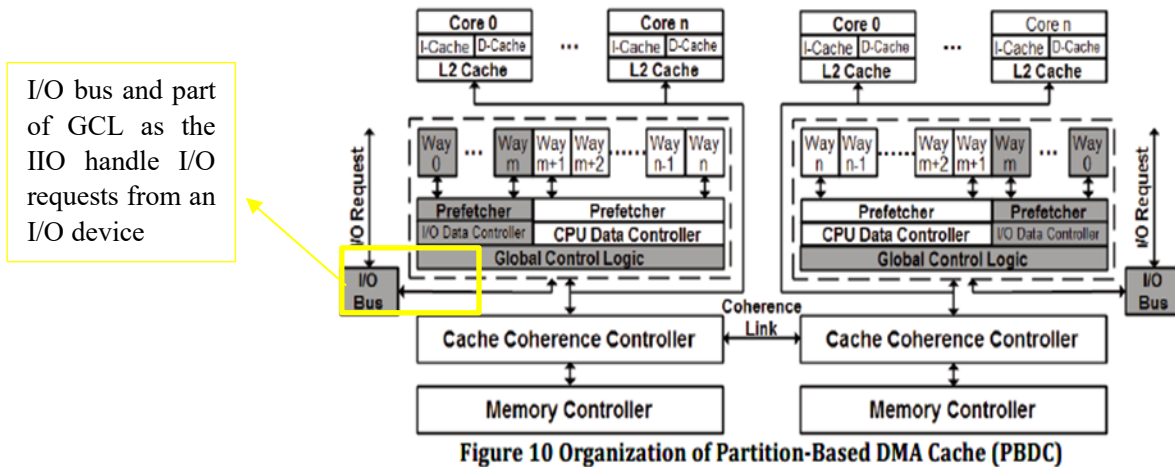
1[d]: an integrated input/output (IIO) module to interface between the processor and an IO device coupled to the processor and

122. In my opinion, Tang teaches Claim 1[d]. Tang discloses an integrated I/O module comprising the I/O bus interface and the part of the global control logic that maintains cache coherence for I/O transactions. A POSITA would understand that “integrated” in the claims is referring to the logical integration of the I/O with the cache controller, not necessarily a physical integration of the IIO module with

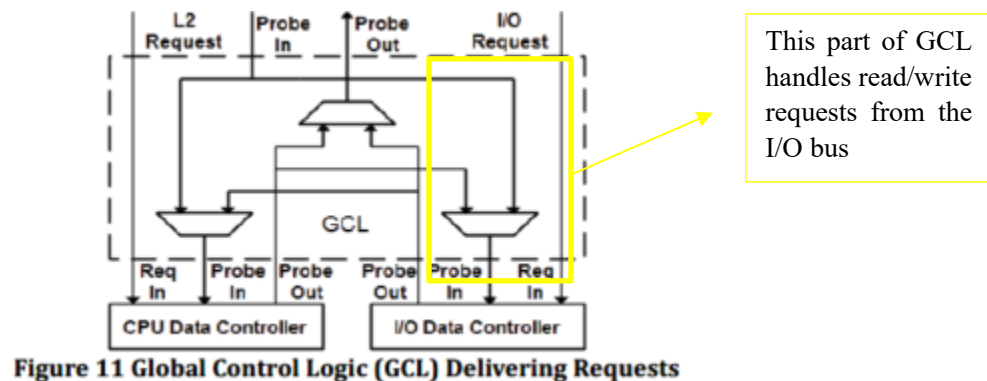
the processor. Tang also talks about the close integration of the I/O interface and caching functions. To the extent a physical integration is required to meet the claim, a POSITA would understand that partitioning into different chips is a standard design choice made in computer design, and different amounts of integration are common based on the available technology. Tang discloses that “[n]ew *I/O bus* technologies, such as PCI-Express 2.0, AMD’s HyperTransport 3.0 and Intel’s QPI, can provide bus bandwidths of 10~20GB/s that are close to the bandwidth of a DDR2/3 DRAM memory system.” (Tang, § 1 (emphasis added).) Tang discloses a DMA cache technique to use “dedicated on-chip storage to store I/O data.” (*Id.*, § 1.) Tang defines I/O data as “a piece of data produced by I/O devices directly.” (*Id.*, § 2.) Figure 10 of Tang discloses a Partition-Based DMA Cache (PBDC) system that includes the I/O bus to transfer I/O data from I/O devices.¹⁷ (*See id.*, § 3.2.1.) Further, Tang discloses that part of Global Control Logic (GCL) “receives the read/write requests from the I/O bus and forward them

¹⁷ This is consistent with the disclosure in the ’895 Patent. It discloses that an “I/O module may act as 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.” ’895 Patent, 2:28-32.

to IOD-Ctrler.” (Tang, § 3.2.2) As such, the I/O bus interface and part of the GCL are an “integrated input/output (IIO) module,” as they their functions are integrated and could be implemented in the same chip to interface between I/O devices connected to the processor chip and the processor.



(Tang, § 3.2.1 Figure 10 (annotated).)



(Tang, § 3.2.1 Figure 11 (annotated).)

1[e]: a caching agent to perform cache coherency operations for the plurality of cores and the IIO module,

123. In my opinion, Tang teaches Claim 1[e]. Tang discloses that in order to “address the I/O data coherence challenge, we have refined both the MOESI [3]

and ESI [16] cache coherence protocols for the DMA cache.” (Tang, § 1.) Tang discloses “the refined cache coherence protocols for the DMA Cache are the same as the original MOESI/ESI protocols.” (*Id.*, § 3.1.3) Tang discloses that both the Decoupled DMA cache (DDC) and Partitioned-Based DMA Cache (PBDC) use the same “write policy, cache coherence protocol, replacement policy.” (*Id.*, § 3.2.1 (“Because some important design issues of **PBDC, such as write policy, cache coherence protocol, replacement policy, can adopt the same design of DDC . . .**”) (emphasis added).) Figure 10 discloses a PBDC system includes a cache coherence controller and an LLC controller. For example, the LLC controller can receive the Probe-Read/Probe-Write requests from cache coherence controller. (*See id.*) The Probe-Read/Probe-Write requests are part of “cache coherency operations.” The LLC-Ctrler includes IO Data Controller and CPU Data Controller, which are responsible for “maintaining cache coherence” and “tagging data for the I/O data ways (Way₀~Way_m) and the CPU data ways (Way_{m+1}~Way_n).”¹⁸ (*See Id.*, § 3.2.2.)) These LLC data ways (Way₀ to Way_n) are corresponding to

¹⁸ This is consistent with the disclosure in the '895 Patent. It discloses that “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.” ('895 Patent, 2:44-47.)

the core 0 to core n. Thus, the cache coherence controller and the LLC controller are the “caching agent to perform cache coherency operations” for the cores.

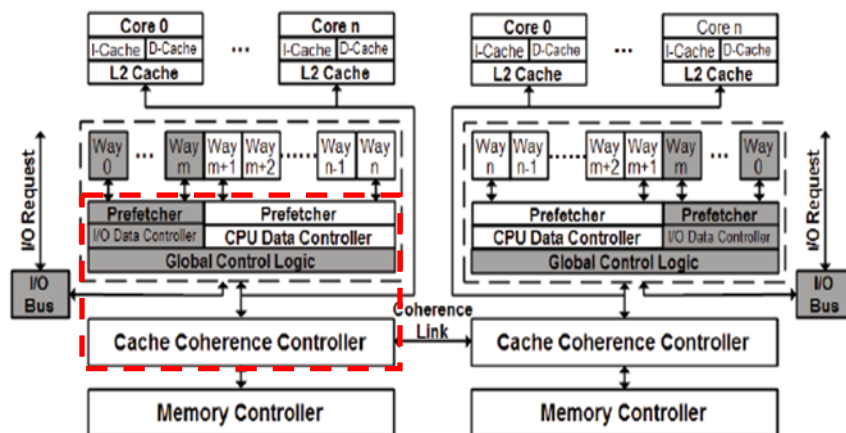


Figure 10 Organization of Partition-Based DMA Cache (PBDC)

(Tang, § 3.2.1 Figure 10 (annotated).)

124. In addition, Tang discloses the LLC-Ctrlr includes two prefetchers, which are responsible “for fetching I/O data into DMA Cache in the CPDC direction.” (*Id.*, § 3.1.1 (for reasons stated above, the DDC and PBDC designs have the same write policy, cache coherence protocol, replacement policy.)) When the prefetchers fetch the I/O data from the I/O bus (as the “IIO module”), the fetch/prefetch operation is also part of the “cache coherency operation.” Thus, the

“[C]aching 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.” (*Id.* 3:1-5.)

cache coherence controller and LLC-Ctrler as the caching agent also perform cache coherency operations for the IIO module.

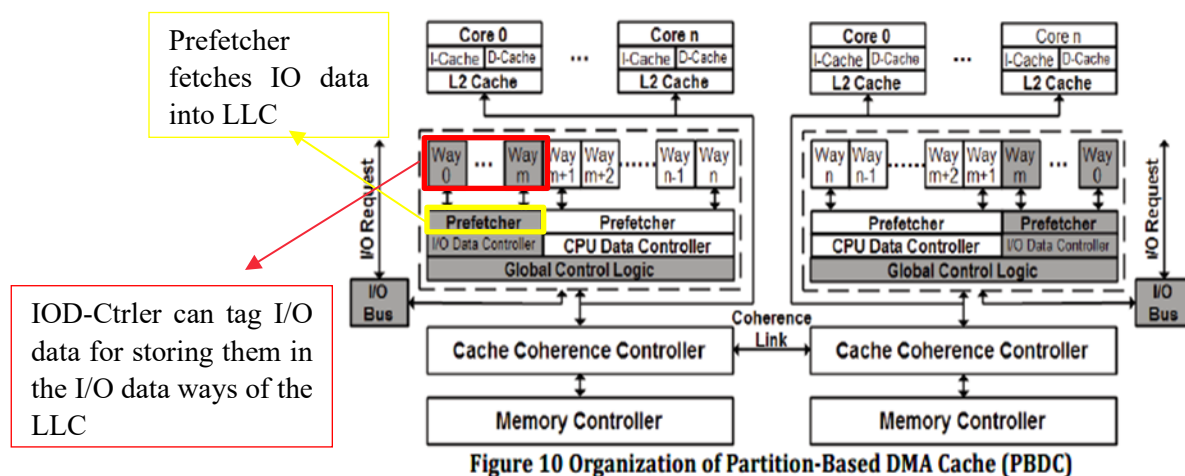
1[f]: wherein the processor is to receive an allocation transaction from the IO device and directly store data of the allocation transaction into the shared cache memory,

125. In my opinion, Tang teaches Claim 1[f]. Tang discloses that the cache coherence protocol of the PBDC design “can adopt the same design of DDC,” the prefetcher in Figure 10 of the PBDC design can also fetch I/O data into DMA cache. (See Tang, § 3.2.1.) Fetching I/O data to be stored into the cache includes requests to write the I/O data into the DMA cache, which are allocation transactions.¹⁹ Tang discloses that the processor’s LLC with several ways (Way₀-Way_n) as the “DMA cache” in the PBDC design. (See Tang, § 3.2.1 (same write policy, cache coherence protocol, and replacement policy as the DDC design disclosed in § 3.1).) As stated in Claim 1[b], the LLC with partitioned Way₀ to Way_n is the “shared cache memory.” (See *supra* Section V.B.1.a (Claim 1[b]).) Thus, when I/O data is stored into the LLC, Tang discloses “directly stor[ing] data

¹⁹ This is consistent with the disclosure in the ’895 Patent. It discloses that the “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 . . .” (’895 Patent 3:55-58.)

of the allocation transaction into the shared cache memory,” where the allocation transaction can be requests to write from the I/O device. Fetching I/O data to be stored into the cache includes requests to write the I/O data into the DMA cache, which are allocation transactions.

126. In addition, Tang discloses the LLC-Ctrler (last level cache controller) includes an IO Data Controller (IOD-Ctrler). (Tang, § 3.2.2.) The IOD-Ctrler are responsible for “write policy, maintaining cache coherence, replacement policy, and tagging data for the I/O data ways (Way₀-Way_m).” (*Id.*) Thus, Tang discloses that I/O data from the I/O device can be directly stored into the I/O data ways in the shared LLC. Storing I/O data into the I/O data ways of the LLC includes requests to write I/O data from the I/O device. As such, Tang discloses “directly store data of the allocation transaction into the shared cache memory.”



(Tang, § 3.2.1 Figure 10 (annotated).)

1[g]: wherein the caching agent is 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,

127. In my opinion, Tang teaches Claim 1[g]. As stated in Claim 1[e], the cache coherence controller and the LLC Controller (LLC-Ctrler) are the “caching agent” and form a logical “single caching agent for the processor.” The disclosures in the ’895 Patent show that the caching agent is a single logical unit for the whole processor, which has distributed physical partitions. (*See supra* Section V.B.1.a (1[e]).) In a multicore processor system, caches need to maintain coherence with each other. If each cache ignored read and write transactions to blocks shared with other caches, the system would not operate properly because the same cache block could be simultaneously modified by multiple processors, resulting in inconsistent (incoherent) data. Instead, cache coherence is maintained by a “single caching agent” which is a single logical component formed by physically distributed portions.²⁰

²⁰ This is consistent with the disclosures in the ’895 Patent. It discloses 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.” (’895 Patent, 3:6-10.) The ’895 Patent further discloses, with respect to FIG. 6

128. In the PBDC design, Tang discloses that the LLC is partitioned into “several ways” as the “dedicated DMA Cache. (Tang § 3.2.1; *see also* § 6 (“The partitioned cache approaches are proposed to reduce cache interferences among the threads in multicore platform.”).)

129. Furthermore, within the caching agent, Tang discloses the LLC-Ctrler includes Global Control Logic (GCL), IO Data Controller (IOD-Ctrler), CPU Data Controller (CPUD-Ctrler), two Prefetchers and Configuration Module (CM). (Tang, § 3.2.2.) The LLC data ways Way₀ to Way_n are partitioned LLC portions. (*See* Tang, Figure 10.) IOD-Ctrler and CPUD-Ctrler, as part of the caching agent are responsible for “write policy, maintaining cache coherence, replacement policy and tagging data for the I/O data ways (Way₀~Way_m) and the CPU data ways (Way_{m+1}~Way_n).” (Tang, § 3.2.2.) Because the LLC-Controller and the cache coherence controller (i.e., the caching agent) manage cache coherence protocol for the plurality of cores (core

thereof, that 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.” (*Id.* 6:33-35.) In particular, it discloses that “while distributed caching agents are shown, understand that these distributed portions **form** a single cache agent.” (*Id.* 6:35-37 (emphasis added).)

0 to core n) and the n-way LLC (Way₀ to Way_n), they necessarily have partitioned regions or units (i.e., distributed portions), each associated with a given core.

LLC-Ctrler and Cache Coherence Controller, as the single caching agent, have distributed portions, each associated with a given core among Cores 0 to n

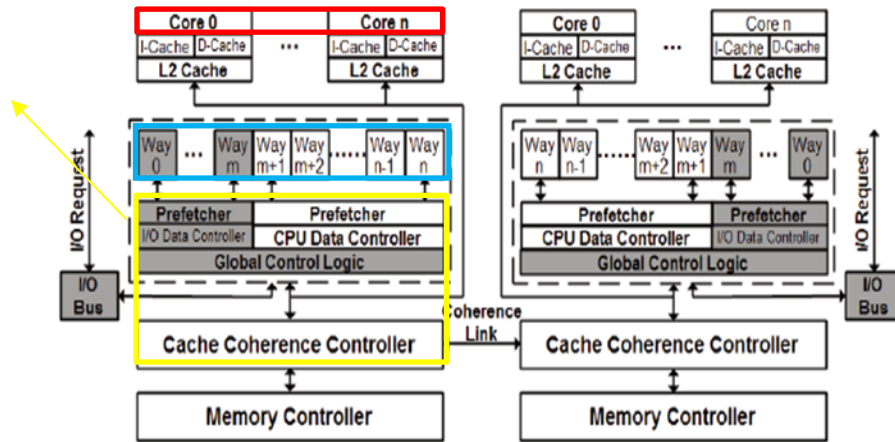


Figure 10 Organization of Partition-Based DMA Cache (PBDC)

(Tang, § 3.2.1 Figure 10 (annotated).)

b. Independent Claim 11

11[p]: *A non-transitory machine-readable medium having stored thereon instructions, which if performed by a machine cause the machine to perform a method comprising:*

130. In my opinion, Tang teaches Claim 11[p] to the extent the preamble limits the scope of the claim. A POSITA would understand that general purpose computers have nonvolatile disks or flash drives and those drives include the object code of programs which are read from the drive and brought into memory in order to run the programs. Thus, general purpose computer systems include “machine-readable medium having stored thereon instructions.” As discussed above in the PBDC design, Tang discloses a multiprocessor system that can handle write policy, cache coherence protocol, replacement policy of the DMA cache.

(See Tang, § 3.2.1.) The multiprocessor system discloses the non-transitory machine-readable medium having stored thereon instructions” to perform a method.

11[a]: receiving an allocation transaction in an integrated input/output (IIO) module of a multicore processor from an IO device coupled to the multicore processor;

131. In my opinion, Tang teaches Claim 11[a]. As discussed in Claim 1[f], Tang defines I/O data as “a piece of data produced by I/O devices directly.” (Tang, § 2). As stated in Claim 1[d], Figure 10 of Tang discloses a PBDC system includes the I/O bus to handle I/O data from I/O devices. (*Id.*, § 3.2.1). The I/O bus and part of the GCU are the IIO module, which handles I/O requests from an IO device coupled to the processor. (See *supra* Section V.B.1.a (Claim 1[d]).)

132. As discussed in Claim 1[f] *supra*, the processor receives a request to write I/O data from the IO device (i.e., “allocation transaction”) in the I/O bus (i.e., the IIO module). Thus, In my opinion, Tang teaches Claim 11 (a).

11[b]: forwarding the allocation transaction to a caching agent of the multicore processor to cause the data to be stored into a shared cache memory of the multicore processor, the caching agent shared by a plurality of cores of the multicore processor and the IIO module,

133. In my opinion, Tang teaches Claim 11[b]. As discussed in Claim 1[e] *supra*, the cache coherence controller and LLC-Ctrler in Tang are the “caching agent,” which are shared by the cores and the I/O bus (i.e., IIO module). (See Claim 1[e] *supra*.) In the PBDC design, Tang discloses a “Last Level Cache

Controller (LLC-Ctrler)”, which includes Global Control Logic (GCL), IO Data Controller (IOD-Ctrler), CPU Data Controller (CPUD-Ctrler), two Prefetchers and Configuration Module (CM). (Tang, § 3.2.2.) Fetching I/O data to be stored into the DMA cache includes requests to write the I/O data into the DMA cache. These requests to write are allocation transactions. (*See supra* Section V.B.1.a (Claim 1[f]).)

134. Further, as stated in Claim 1[b], the LLC with partitioned Way₀ to Way_n is the “shared cache memory.” (*See supra* Section V.B.1.a (Claim 1[b]).) As discussed in Claim 1[f], the IOD-Ctrler of the LLC controller is responsible for “write policy, maintaining cache coherence, replacement policy, and tagging data for the I/O data ways (Way₀-Way_m).” (*See supra* Section V.B.1.a (Claim 1[f]).) Thus, Tang discloses that I/O data from the I/O device can be directly stored into the I/O data ways in the shared LLC. Storing I/O data into the I/O data ways of the LLC includes sending requests to write I/O data from the I/O device to the IOD-Ctrler (i.e., part of the caching agent). Thus, Tang discloses “forwarding the allocation transaction to a caching agent of the multicore processor to cause the data to be stored into a shared cache memory,” where the allocation transaction can be requests from the I/O device via the prefetcher.

11[c]: wherein the caching agent is a single caching agent for the multicore processor and includes a plurality of distributed portions each associated with a corresponding one of the plurality of cores; and

135. For the reasons stated in *supra* Section V.B.1.a (claim 1[g]), In my opinion, Tang teaches Claim 11[c].

11[d]: directly storing the data of the allocation transaction into the shared cache memory of the multicore processor without sending the data to a system memory coupled to the multicore processor.

136. In my opinion, Tang teaches Claim 11[d]. For the reasons stated in Claim 1[f] *supra*, Tang discloses directly storing the data of the allocation transaction into the shared cache memory. (*See supra* Section V.B.1.a (Claim 1[f]).) For the reasons stated in Claim 1[c], Tang discloses a system memory coupled to the processor. (*See supra* Section V.B.1.a (Claim 1[c]).) Furthermore, Tang discloses a write policy for the DMA cache, including write-back (WB) and write-through (WT) policies. (*See* Tang, § 3.1.2.) Tang discloses that when adopting WB policy, the I/O data written to the DMA Cache is initiated as “Modified-state” and “deferred to be updated into the memory until it is replaced.” (*Id.*) When adopting WT policy, the I/O data is “written to the DMA Cache as well as the memory.” (*Id.*) Thus, when the DMA Cache (i.e., the shared cache memory) performs WB policy to write I/O data into the DMA cache and before such data is replaced, the data of the allocation transaction is stored directly into the shared

cache. The WB policy does not send the data to “a system memory coupled to the multicore processor” until the data is replaced.

c. Independent Claim 15

15[p]: *A system comprising:*

137. In my opinion, Tang teaches Claim 15[p] to the extent the preamble limits the scope of the claim. In the PBDC design, Tang discloses a multiprocessor **system** that can handle write policy, cache coherence protocol, replacement policy of the DMA cache. (See Tang, § 3.2.1 (emphasis added).)

15[a]: *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,*

138. In my opinion, Tang teaches Claim 15[a] for the reasons stated in *supra* Section V.B.1.a (Claims 1[a]-1[e]).

15[b]: *wherein the first caching agent is to receive an allocation transaction from the first IIO module and directly store data of the allocation transaction into the first shared cache memory,*

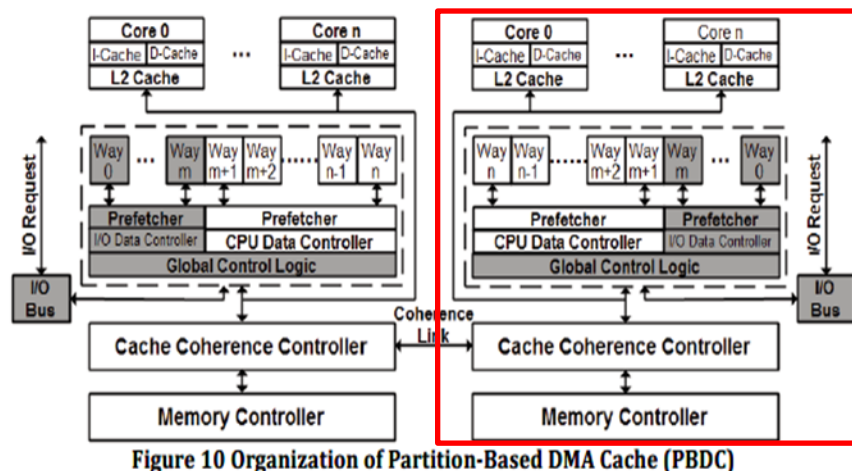
139. In my opinion, Tang teaches Claim 15[b] for the reasons stated in *supra* Section V.B.1.a (Claim 1[f]).

15[c]: 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,

140. In my opinion, Tang teaches Claim 15[c] for the reasons stated in *supra* Section V.B.1.a (Claim 1[g]).

15[d]: 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

141. In my opinion, Tang teaches Claim 15[d] for similar reasons stated in *supra* Section V.B.1.a (Claims 1[a]-1[e]). Tang discloses a system which has two processors, connected with each other via a coherence link. (Tang Figure 10.) The second processor in the system annotated below is the “second multicore processor” with the same structure as the “first multicore processor.”



The second multicore processor with a second IIO, a second plurality of IO devices, and a second caching agent

(Tang, § 3.2.1 Figure 10 (annotated).)

15[e]: a peripheral controller coupled to at least one of the first and second multicore processors.

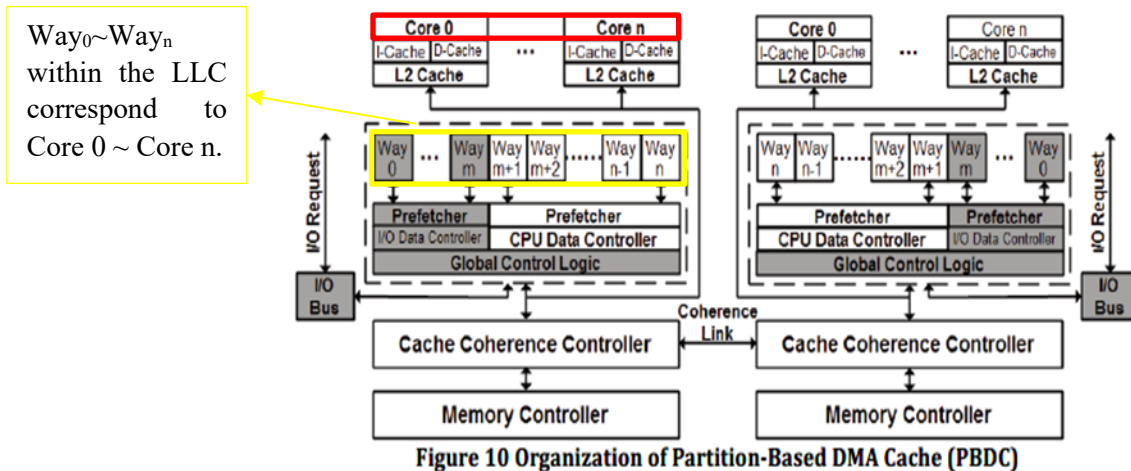
142. In my opinion, Tang teaches Claim 15[e]. Tang discloses each of the first and second multicore processors include a “peripheral controller” coupled to the process, as each have I/O bus technologies such as “PCI-Express 2.0, AMD’s HyperTransport 3.0, and Intel’s QPI, that connect to peripheral devices having peripheral controllers. (Tang §1.) For example, many peripheral devices with peripheral controllers are connected to processors via Tang’s disclosed PCI-Express 2.0 bus technology.

d. Dependent Claim 5

5: The processor 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.

143. In my opinion, Tang teaches Claim 5. As stated in Claim 1[b], the partitioned last level cache (LLC) in Tang is the “shared cache memory.” (See *supra* Section V.B.1.a (Claim 1[b]).) IOD-Ctrler (IO Data Controller) and CPUD-Ctrler (CPU Data Controller), as part of the caching agent, are responsible for “write policy, maintaining cache coherence, replacement policy and tagging data for the I/O data ways (Way₀~Way_m) and the CPU data ways (Way_{m+1}~Way_n).” (Tang, § 3.2.2.) The data ways Way₀ to Way_n in the LLC are each associated with a given core among Core 0 to Core n. (See Tang, Figure 10; see also *supra* Section

V.B.1.a (Claim 1[g]).²¹ Each way of the LLC is a “distributed portion” corresponding to a given core.



(Tang, § 3.2.1 Figure 10 (annotated).)

²¹ This is consistent with the disclosure in the '895 Patent. It discloses in FIG. 6 that 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.” ('895 Patent 6:33-35.) In particular, it discloses that “the various cores may be coupled to each other a shared cache memory **formed** of a plurality of units of a LLC 740₀-740_n via caching agent 715₀-715_n.” (*Id.* 6:42-45 (emphasis added).)

e. **Dependent Claim 6**

6: The processor of claim 1, wherein the processor is to directly store the data of the allocation transaction into the shared cache memory without initiation of a memory transaction to the memory.

144. In my opinion, Tang teaches Claim 6. Tang discloses write policies for the DMA cache, including write-back (WB) and write-through (WT) policies. (See Tang, § 3.1.2.) Tang discloses that when adopting WB policy, the I/O data written to the DMA Cache is initiated as “Modified-state and deferred to be updated into the memory until it is replaced.” (*Id.*) When adopting WT policy, the I/O data is “written to the DMA cache as well as the memory. (*Id.*)” Thus, when the partitioned LLC (i.e., the DMA cache) adopts WB policy and before the I/O data is replaced, as discussed in Claim 11[d], the I/O data is directly written in the shared cache without touching the memory. (See *supra* Section V.B.1.b (Claim 11[d]).)

145. In addition, Tang discloses that for the DMA cache, when using WB policy, it can use the IO-MOESI protocol. (See Tang, § 3.1.3.) “When the DMA cache receives a DMA write request, it first broadcasts a Probe-Read request to other caches. If the Probe-Read request hits in a CPU’s LLC, the LLC changes the hit block’s state to either Shared-state or Owned-state, and sends data to the DMA Cache. After receiving the most recent data, the DMA Cache broadcasts a Probe-Write. The LLCs, in which the Probe-Write hit, change the hit block with Shared-

state or Owned-state to Invalid-state. Then, the DMA cache changes the block to Modified-state (using IO-MOESI with WB policy) or Exclusive-state (using IO-ESI with WT policy **where the data is updated into memory as well**)”. (Tang, § 3.1.3, emphasis added) Thus, the DMA write request process for the shared LLC using IO-MOESI with WB policy does not initiate a memory transaction.

f. Dependent Claim 7

7: The processor of claim 6, wherein the processor is to write the data of the allocation transaction to the memory responsive to eviction of a cache line including the data.

146. In my opinion, Tang teaches Claim 7. Claim 7 is dependent on Claim 6 in which the “allocation transaction” refers to storing I/O data into cache without storing it into memory. Claim 7 requires the cached I/O data to be later written to memory when the line is evicted. A POSITA would understand that when the cache line was written, it was “dirty”, meaning the cached value was not the same as the memory value. Later, when that cache line is evicted from the cache, it needs to be written back to memory unless it is known that the line will never be reused. Tang teaches a standard MOESI cache protocol that includes writing I/O cache lines back to memory (as well as the IO-MOESI protocol in which the

modified line is not written back)²². Tang discloses a write policy for the DMA cache, including write-back (WB) and write-through (WT) policies. (Tang, § 3.1.2.) Tang discloses when adopting a standard WT policy, the “I/O data is written to the DMA Cache as well as the memory.” (*Id.*)

147. A POSITA would understand that the dirty cache line would be written back to the memory in the process of being evicted from the cache. Thus, when the DMA cache adopts a standard WB policy, the I/O data of the allocation transaction is evicted from the DMA cache and written back into the system memory.

g. Dependent Claim 8

8: The processor 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 the IO device responsive to receipt of a snoop request in the caching agent for the cache line from another agent.

148. In my opinion, Tang teaches Claim 8. For the reasons stated in Section V.A.3.b (Claim 8), when L2 cache (cache B) associated with a given core tries to read the cache line (i.e., data) from the shared LLC, the L2 cache for the

²² The write-back policy in Tang 3.1.2 is describing a standard policy, such as MOESI, not the modified IO-MOESI policy described later in the paper. A POSITA would understand that Tang has disclosed both policies for write-back caches including one where evictions always cause writes back to memory.

core issues a “Probe-Write hit” to the LLC, where the “Probe-Write hit” is a “snoop request.” (Section V.B.3.b (Claim 8) *supra*.) Tang discloses that the DMA cache, when using WB policy, can use the IO-MOESI protocol. (See Tang, § 3.1.3.) “When the DMA cache receives a DMA write request, it first broadcasts a **Probe-Read request** to other caches. If the Probe-Read request hits in a CPU’s LLC, the LLC changes the hit block’s state to either Shared-state or Owned-state, and sends data to the DMA Cache. After receiving the most recent data, the DMA Cache **broadcasts a Probe-Write. The LLCs, in which the Probe-Write hit, change the hit block with Shared-state or Owned-state to Invalid-state.** Then, the DMA cache changes the block to Modified-state (using IO-MOESI with WB policy)” (Tang, § 3.1.3, emphasis added) In response to the snoop request, the *caching agent* (portion associated with the IIO module) changes the cache state of the cache line from “Owned” to “Invalid,” (“*send[ing] a drop ownership indication to the IIO module.*”) (Tang, §3.1.3) A POSITA would have understood that the Probe-Write hit is a *snoop request* that is managed and received by the cache controllers (*caching agent*) which causes the state to be changed to Invalid (a *drop ownership indication*).

h. Dependent Claim 9

9: The processor 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 the IO device responsive to receipt of a snoop request in the caching agent for the cache line from another agent.

149. In my opinion, Tang teaches Claim 9. For the reasons stated in Claim 8, when L2 cache (cache B) associated with a given core tries to read the cache line (i.e., data) from the shared LLC (the DMA cache), the core associated with the L2 cache is “another agent.” (*See supra* Section V.B.1.g (Claim 8).) Furthermore, Tang discloses “a DMA read request on the DMA cache can trigger a prefetching operation,” as “I/O data has very regular reference patterns.” (Tang, § 3.1.5.) Where the cache line is owned by the IIO module, an IO device connected to the IIO already prefetched the cache line during the prefetching operation, because the cache line will be invalidated. A POSITA would have understood that the cache line may have been in the “Owned” state because IO device connected to the IIO module had prefetched the cache line

i. Dependent Claim 12

12: The non-transitory machine-readable medium of claim 11, wherein the method further comprises 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 the system memory.

150. In my opinion, Tang teaches Claim 12. A POSITA would understand that the purpose of a cache is to access data a number of times without sending a

transaction to memory.²³ Similarly, in the PBDC design, Tang discloses a “Last Level Cache Controller (LLC-Ctrler)”, which includes Global Control Logic (GCL), IO Data Controller (IOD-Ctrler), CPU Data Controller (CPUD-Ctrler), two Prefetchers and Configuration Module (CM). (Tang, § 3.2.2.) I/O Data Controller (IOD-Ctrler) is responsible for write policy, maintaining cache coherence, replace policy and tagging data for the I/O data ways (Way₀ ~ Way_m). (*Id.*) As stated in Claim 1[f], storing the I/O data into the I/O data ways of the LLC is storing data into the “shared cache memory.”

151. Furthermore, when GCL receives the read request from the L2 cache, it forwards the request to CPUD-Ctrler. (*Id.*) An L2 cache is corresponding to a specific core, thus, read requests from the L2 cache are “accessing the data in the shared cache memory via at least one of the plurality of cores.” If such requests for the core hit the I/O data ways, an internal probe-read request is generated to IOD-Ctrler, sending the requested I/O data to the L2 cache, which corresponds to a

²³ This is consistent with the disclosure in the '895 Patent. It discloses that “[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 as well as the initiating IO device or other IO devices coupled to the processor.” ('895 Patent, 4:6-9.)

specific core. The process between the L2 cache for the core and the I/O data ways in the LLC does not involve a transaction to the system memory.

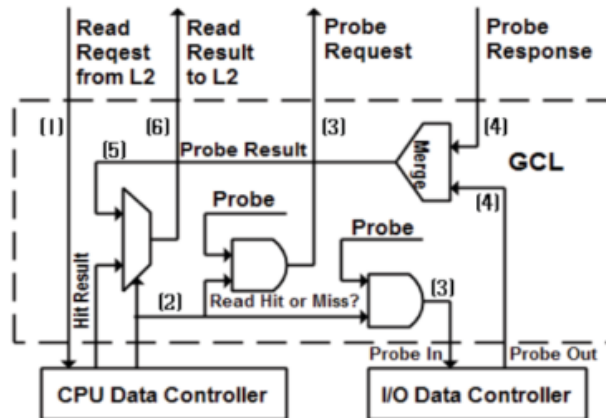


Figure 12 The Process Flow of L2 Cache Read Request

(Tang, § 3.2.1 Figure 12.)

152. Furthermore, as stated in Claim 6 *supra*, when the DMA Cache adopts WB policy and before the I/O data is replaced, the data of the allocation transaction is stored directly into the shared cache without sending the data to a system memory. It would have been readily understood by a POSITA that such data can be accessed by a core via requests from its L2 cache, as described in Figure 12.

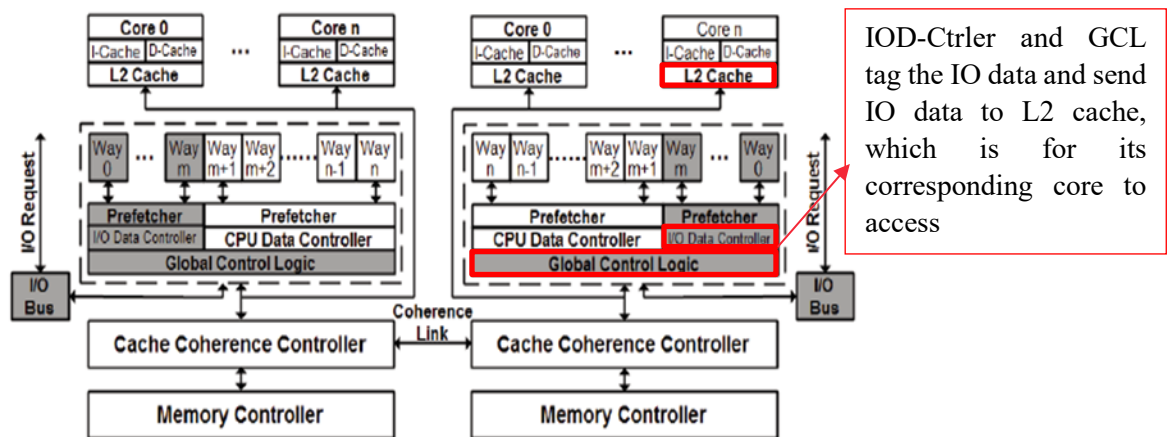


Figure 10 Organization of Partition-Based DMA Cache (PBDC)

IOD-Ctrler and GCL tag the IO data and send IO data to L2 cache, which is for its corresponding core to access

(Tang, § 3.2.1 Figure 10 (annotated).)

j. Dependent Claim 13

13: The non-transitory machine-readable medium of claim 11, wherein the method further comprises sending the data to the system memory from the shared cache memory if a cache line including the data is evicted from the shared cache memory, when the cache line includes modified data.

153. In my opinion, Tang teaches Claim 13. A cache line is the smallest portion of data that can be mapped into a cache. Tang discloses a write policy for the DMA cache, including write-back (WB) and write-through (WT) policies. (Tang, § 3.1.2.) Tang discloses that when adopting WB policy, the I/O data written to the DMA Cache is initiated as “Modified-state and deferred to be updated into the memory until it is replaced.” (*Id.*) Furthermore, because “usually I/O data is used only once, it first evicts the used I/O data.” (Tang, § 3.1.4.) Thus, when the DMA updates the system memory with such data, the data is evicted from the shared cache memory. Further, in Figure 8, Tang discloses that when the DMA Cache receives a DMA write request, it first broadcasts a Probe-Read request to other caches, such as the CPU’s LLC, and the LLC can send data to the DMA cache. (*See id.*, § 3.1.3.) After receiving the most recent data from the LLC, the DMA Cache changes the block to Modified-state (using IO-MOESI with WB policy). Data in the Modified state may be evicted as shown in Figure 8 with the arrow from the Modified state to the Invalid state. If that data is in the CPU portion of the shared cache (not the I/O portion), the eviction involves writing the data

back to memory. That step comprises “sending the data to the system memory from the shared cache memory if a cache line including the data is evicted from the shared cache memory, when the cache line includes modified data.”

Table 3 Sources of State Transitions

	Local Cache	Remote Cache	DMA Cache
	MOESI/ESI	MOESI/ESI	IO-MOESI/IO-ESI
CPU Read Request	CPU-Read	Probe-Read	Probe-Read
CPU Write Request	CPU-Write	Probe-Write	Probe-Write
DMA Read Request	Probe-Read	Probe-Read	DMA-Read
DMA Write Request	Probe-Write	Probe-Write	DMA-Write

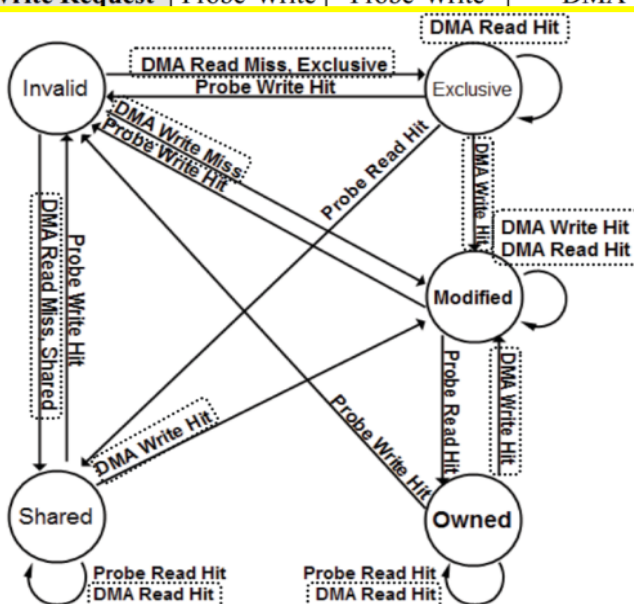


Figure 8 IO-MOESI State Transitions for DMA Cache with Write-Back Policy¹

(Tang, § 3.1.3 Table 3 & Figure 8 (annotated).)

k. Dependent Claim 14

14: *The non-transitory machine-readable medium of claim 11, wherein the method further comprises 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.*

154. In my opinion, Tang teaches Claim 14 for the reasons stated in *supra* Section VIII.B.1.e(Claim 8).

I. Dependent Claim 17

17: The system of claim 15, 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.

155. In my opinion, Tang teaches Claim 17 for the reasons stated in *supra* Section V.B.1.i (claim 12).

2. Ground 5 – Tang in view of Harikumar Renders Obvious Claims 2, 3-4, 10, and 16

156. I have reviewed Harikumar. U.S. Patent Application Publication No. 2009/0164739 to Harikumar, titled “*Method, System and Apparatus for Handling Events for Partitions in a Socket with Sub-Socket Partitioning*,” was filed November 7, 2008 and published on June 25, 2009. I understand Harikumar is prior art to the ’895 under at least pre-AIA 35 U.S.C. § 102(a) and § 102(b). Harikumar, in the same field of art as Tang, discloses a multicore processor with a distributed LLC. (Harikumar ¶ [0031]). Harikumar explicitly discloses the cores and the distributed LLC (Last Level Cache banks) 408 are connected to each other within the socket by a first level interconnect 403. (*Id.*) Harikumar discloses the first level interconnect 403 is an on-die ring interconnect. (*Id.*)

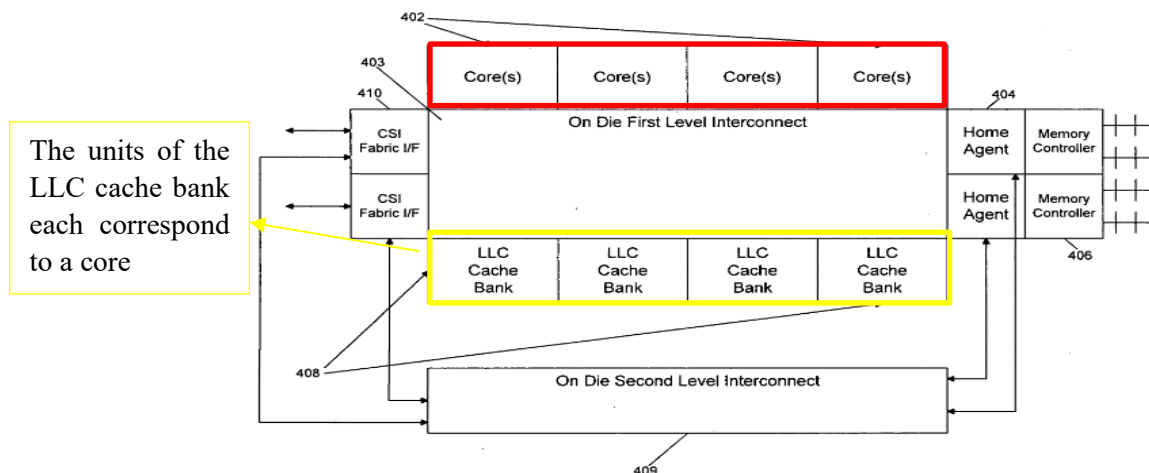


FIG. 4

(Harikumar at FIG. 4.)

157. It would have been obvious to a POSITA to use the ring interconnect disclosed in Harikumar in the processor of Tang. Harikumar and Tang are analogous art; both disclose multicore processors with distributed LLCs, each distributed portion associated with a corresponding core. (*Compare* Tang, Figure 10 (annotated), *with* Harikumar Figure 4 (annotated); *see also* Tang, § 3.2.2; Harikumar, ¶ [0031].)

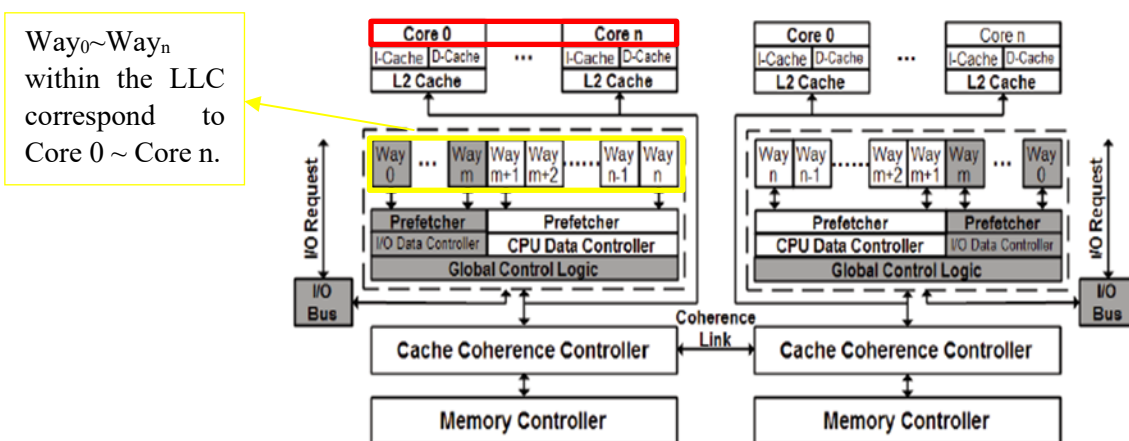


Figure 10 Organization of Partition-Based DMA Cache (PBDC)

(Tang, § 3.2.1 Figure 10 (annotated).)

158. A POSITA would have recognized that using the ring interconnect disclosed in Harikumar in the processor of Tang would be the expected use of a well-known interconnect structure for a processor. Multiprocessor systems need to communicate snoop traffic to all other processor caches to maintain coherence. Some systems use a shared bus, while other use a network such as a ring or crossbar for communicating (coupling) the coherence traffic to the caches. A POSITA would have been motivated to use the ring interconnect disclosed in Harikumar in the processor of Tang to connect the cores, the shared cache memory, and the caching agents and to manage the coherence operations among the components. Using a ring interconnect on a processor chip to couple and connect its components was well known in the art.

159. A POSITA would have a reasonable expectation of success, and such would have been a simple substitution, as the ring interconnect of Harikumar, when used in Tang's multicore processor, would continue to perform their same function: connecting different components of the processor and managing coherence operations and requests. Selecting a ring instead of a bus or crossbar involve tradeoffs in chip area, speed and scalability and is typical of the types of tradeoffs normally made in computer design. Applying the teachings of Harikumar to Tang's multicore processor would have merely involved adding one known

technique of connecting the components of the processor (the cores, caching agent, the shared cache memory) to yield a predictable result (to provide interconnect protocol among the different components of the processor).

a. Dependent Claims 2, 10

160. In my opinion, Tang in view of Harikumar teaches Claims 2 and 10.

2: The processor of claim 1, wherein the plurality of distributed portions of the caching agent are coupled via a ring interconnect.

10: The processor of claim 1, further comprising a ring interconnect to couple the plurality of cores and the shared cache memory via a caching agent.

161. As discussed above, it would have been obvious to a POSITA to use the ring interconnect disclosed in Harikumar in the PBDC design of Tang.

Harikumar explicitly discloses the cores and the distributed LLC (Last Level Cache banks) 408 are connected to each other within the socket by a first level interconnect 403. (Harikumar ¶ [0031].) Harikumar discloses the first level interconnect 403 is an on-die ring interconnect. (*Id.*) As such, using the ring interconnect from Harikumar in Tang would couple the distributed portions of the LLC controller (i.e., “distributed portions” of the caching agent), the cores, and the shared LLC (i.e., the shared cache memory).

b. Dependent Claim 3

3: *The processor of claim 1, wherein the caching agent includes a table of requests to store pending requests for the plurality of cores and the I/O module.*

162. In the Partition-Based DMA Cache (PBDC) design, Tang discloses that the processor's last level cache (LLC) can be divided into "several ways [n-way]" to store I/O data. (*Id.*, § 3.2.2.) As stated in Claim 1[e] *supra*, the cache coherence controller and LLC-Controller (LLC-Ctrler) are the "caching agent." The LLC-Ctrler includes Global Control Logic (GCL), which is responsible for routing the read/write requests and the Probe-Read/Probe-Write requests to the right destinations. (*Id.*).

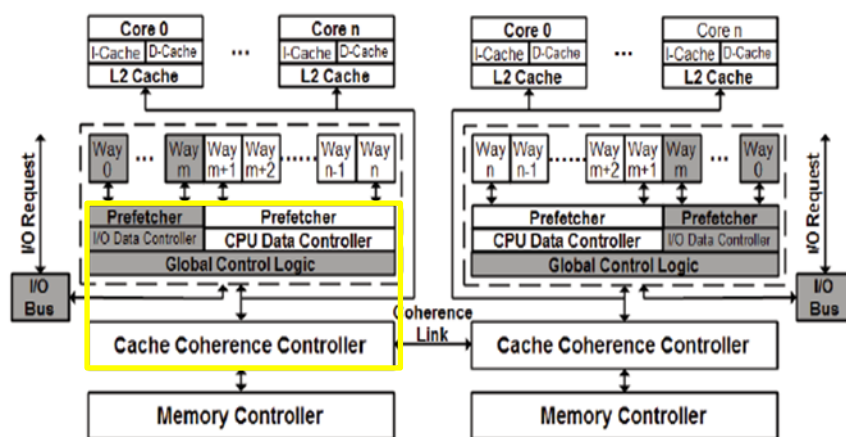


Figure 10 Organization of Partition-Based DMA Cache (PBDC)

(Tang, § 3.2.1 Figure 10 (annotated).)

163. Harikumar, in the same field of art as Tang, discloses a multicore processor with a distributed LLC. (Harikumar ¶ [0031].) Harikumar explicitly discloses a caching agent may have a corresponding home agent, which includes a tracker and data buffers. (*Id.* ¶ [0014].) Harikumar teaches that a "tracker" is

dedicated storage for memory requests from a particular device, which may include multiple entries associated with a given caching agent. (*Id.*) As illustrated in FIG. 3, each caching agent source 310A, ..., or 310D is associated with a tracker that has a predetermined number of tracker entries. (*Id.* ¶ [0025].) The number of tracker entries is limited in size to the number of requests. (*Id.*) In addition, the caching agents can each further include a credit counter, which maintains a count value representative of the number of unused tracker entries in the trackers. (*Id.* ¶ [0029].) For example, when a new transaction is issued by a caching agent to its corresponding home agent, its credit counter is decremented. (*Id.*) If a transaction completes, the credit counter is incremented. (*Id.*) The trackers and credit counters in Harikumar include temporary storages and store pending requests. They are “table of requests” in the claims.²⁴

²⁴ This is consistent with the disclosures in the '895 Patent. It discloses that a “table of requests (TOR)” may be a buffer or other temporary storage for holding incoming requests. ('895 Patent, 2:51-53.) As such, TORs include buffers or temporary storages for storing incoming or pending requests. the '895 Patent discloses the table of requests may include at least one “dedicated resource for post transactions.” ('895 Patent, 2:53-55.) “In one embodiment, dedicated resource 225 may be a single entry of the queue.” (*Id.*, 2:55-57.)

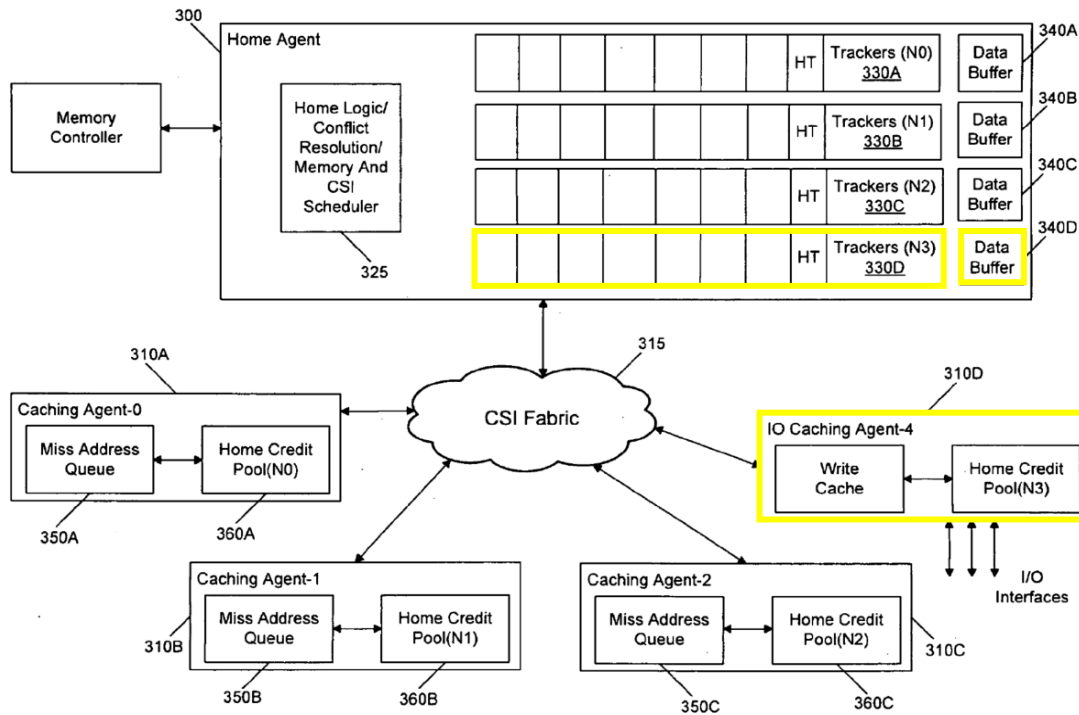


FIG. 3

(Harikumar at FIG. 3 (annotated).)

164. It would have been obvious to a POSITA to use the tracker or credit counter disclosed in Harikumar in the LLC-Controller of Tang. Harikumar and Tang are analogous art; both disclose multicore processors with distributed LLCs, each distributed portion associated with a corresponding core.

165. Furthermore, a POSITA would have recognized that using the tracker or credit counter disclosed in Harikumar in the LLC-Controller of Tang (part of the caching agent) would be the expected use of such well-known temporary storage for holding incoming requests to manage read/write transactions for the cache controllers. A POSITA would have been motivated to use the tracker or credit counter taught in Harikumar in the LLC-Controller of Tang to achieve the same

purpose to provide temporary storage of incoming requests and better manage pending read/write requests in the cache controllers.

166. A POSITA would have a reasonable expectation of success, and such would have been a simple substitution, as the tracker or credit counter of Harikumar, when used in Tang's LLC-Controller, would continue to perform their same function: storing incoming requests to manage and route read/write requests. They are temporary storages and would have been easy to implement in the LLC-Controller of Tang. Applying the teachings of Harikumar to Tang's caching agent would have merely involved replacing one known technique of providing temporary data storage for incoming requests to yield a predictable result, to store pending read/write requests for the caching agent.

c. Dependent Claim 4

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

167. In my opinion, Tang in view of Harikumar teaches Claim 4. As discussed in Claim 3 *supra*, it would have been obvious to a POSITA to use the tracker or credit counter disclosed in Harikumar in the LLC-Controller of Tang. (See *supra* Section V.B.2.b (Claim 3).) When the transaction is a write transaction to be stored in the tracker or credit counter (i.e., table of request), the tracker or

credit counter has a dedicated entry for a “posted transaction,” i.e., the write transaction²⁵.

d. Dependent Claim 16

16: The system of claim 15, 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.

168. In my opinion, Tang in view of Harikumar teaches Claim 16 for the reasons stated in *supra* Section V.B.2.b-c (Claims 3 and 4).

²⁵ This is consistent with the disclosure in the '895 Patent. It discloses “a posted transaction is a transaction which when sent by a source is considered complete by the source and the source does not receive a completion or other confirmation message regarding the transaction. One such example of a posted transaction may be a write transaction.” ('895 Patent, 5:12-17.)

VI. CERTIFICATION

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true and, further, that these statements were made with the knowledge that willful false statements and the like are punishable by fine, imprisonment, or both, under Section 1001 of Title 18 of the U.S. Code.

A handwritten signature in blue ink, appearing to read "Robert W. Horst", written in a cursive style.

Dr. Robert W. Horst

Dated: August 23, 2023

APPENDIX A

Materials Considered

Exhibit No.	Description
1001	U.S. Patent No. 9,575,895 (“’895 Patent”)
1002	File History for the ’895 Patent
1008	Curriculum Vitae of Dr. Robert W. Horst
1003	<i>IBM POWER7 multicore server processor</i> to Sinharoy et al. (“Sinharoy”)
1004	<i>DMA Cache: Using On-Chip Storage to Architecturally Separate I/O Data from CPU Data for Improving I/O Performance</i> to Tang et al. (“Tang”)
1005	<i>Explorations of Optimal Core and Cache Placements for Chip Multiprocessor</i> to Xu et al. (“Xu”)
1006	U.S. Patent Application Publication No. 2009/0164739 to Harikumar (“Harikumar”)
1016	<i>Architectural Implications of Cache Coherence Protocols with Network Applications on Chip Microprocessors</i> to Kyueun Yi et al. (“Yi”)