

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TESLA, INC.,
Petitioner

v.

INTELLECTUAL VENTURES II LLC,
Patent Owner

IPR2025-00638
U.S. Patent No. 8,898,395

**PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT 8,898,395**

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	CERTIFICATION OF GROUNDS FOR STANDING	1
III.	NOTE	2
IV.	TECHNOLOGY OVERVIEW	2
V.	THE '395 PATENT	3
	A. Overview	3
	B. Prosecution History	6
	1. Prosecution History of the '798 Patent	6
	2. Prosecution History of the '395 Patent	8
	3. Note Regarding Alleged Point of Novelty	11
VI.	LEVEL OF ORDINARY SKILL IN THE ART	12
VII.	CLAIM CONSTRUCTION.....	13
	A. "said processing" (claims 1, 2).....	13
VIII.	THE CHALLENGED CLAIMS ARE UNPATENTABLE.....	17
	A. Challenged Claims	17
	B. Prior Art Patents and Printed Publications.....	17
	C. Statutory Grounds for Challenges	18
	D. Ground 1: Claims 1-2, 5, 7-8, and 11 would have been obvious over Moir and Martínez.....	18
	1. Moir.....	18

2.	Martínez	25
3.	Analogous Art.....	26
4.	Reasons to Combine Moir and Martínez	29
5.	Claim 1	34
6.	Claim 2	59
7.	Claim 5	64
8.	Claim 7	67
9.	Claim 8	69
10.	Claim 11	69
IX.	MARTÍNEZ IS A PRINTED PUBLICATION	69
X.	DISCRETIONARY DENIAL IS INAPPROPRIATE	71
A.	No Basis For §325(d) Denial	71
B.	No Basis for <i>Fintiv</i> Denial	71
1.	No evidence regarding a stay	71
2.	Parallel proceeding trial date	72
3.	Investment in the parallel proceeding.....	73
4.	Overlapping issues with the parallel proceeding	73
5.	Petitioner is a defendant.....	74
6.	Other circumstances.....	74
C.	No Basis for <i>General Plastic</i> Denial Under §314(a).	74
XI.	CONCLUSION	75

XII. MANDATORY NOTICES UNDER 37 C.F.R. §42.8.....76

 A. Real Party-in-Interest76

 B. Related Matters.....76

 C. Lead and Back-up Counsel and Service Information77

XIII. CERTIFICATE OF WORD COUNT78

PETITIONER'S EXHIBIT LIST

Ex.1001	U.S. Pat. No. 8,898,395 to G. J. Rozas (“the ’395 patent”)
Ex.1002	Prosecution History of U.S. Application No. 12/121,531 (issued as the ’395 patent) (“’395 FH”)
Ex.1003	Declaration of Dr. Robert Colwell under 37 C.F.R. §1.68
Ex.1004	<i>Curriculum Vitae</i> of Dr. Robert Colwell
Ex.1005	U.S. Pat. No. 7,206,903 to Moir et al. (“Moir”)
Ex.1006	“Speculative synchronization: programmability and performance for parallel codes.” J. F. Martínez et al. IEEE Micro, December 2003 (“Martínez”)
Ex.1007	Prosecution History of U.S. Application No. 11/102,127 (issued as US 7,376,798) (“’798 FH”)
Ex.1008	Declaration of Gordon MacPherson
Ex.1009	Landing Page, IEEE Micro Vol. 23, No. 6, Internet Archive Capture, Feb. 1, 2004, https://web.archive.org/web/20040201212609/http://www.computer.org/micro/
Ex.1010	Table of Contents, IEEE Micro Vol. 23, No. 6, Internet Archive Capture, Feb. 14, 2004, https://web.archive.org/web/20040214122853/http://csdl.computer.org/comp/mags/mi/2003/06/m6toc.htm
Ex.1011	U.S. Pat. Pub. No. 2002/0199066 to S. Chaudhry et al.
Ex.1012	“Transactional Memory: Architectural Support for Lock-Free Data Structures,” M. Herlihy and J. E. B. Moss, ISCA '93: Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 289-300, May 1993 (“Herlihy and Moss Paper”)
Ex.1013	U.S. Pat. No. 5,428,761 to Herlihy and Moss. (“Herlihy and Moss Patent”)
Ex.1014	IV’s Complaint, <i>Intellectual Ventures II, LLC v. Tesla, Inc.</i> , No. 6:24-cv-188-ADA (WDTX)

Ex.1015	Proposed Scheduling Order, <i>Intellectual Ventures II, LLC v. Tesla, Inc.</i> , No. 6:24-cv-188-ADA (WDTX)
Ex.1016	Statistics on District Court Timing
Ex.1017	<i>Reserved</i>
Ex.1018	IV's Preliminary Infringement Contentions, <i>Intellectual Ventures II, LLC v. Tesla, Inc.</i> , No. 6:24-cv-188-ADA (WDTX)
Ex.1019	Shen & Lipasti, <i>Modern Processor Design</i> , McGraw Hill, 2005
Ex.1020	<i>Concise Encyclopedia of Computer Science</i> , Wiley, 2004
Ex.1021	<i>The Cache Memory Book</i> , Jim Handy, 2nd ed., Academic Press, 1998
Ex.1022	Hennessy & Patterson <i>Computer Architecture: A Quantitative Approach</i> 3rd ed., Morgan Kaufmann, 2002
Ex.1023	IEEE Authoritative Dictionary of IEEE Standards Terms, 7th edition, 2000 (excerpts)
Ex.1024	Microsoft Computer Dictionary, Fifth Edition, 2002 (excerpts)
Ex.1025	Tesla's Opening Claim Construction Brief, <i>Intellectual Ventures II, LLC v. Tesla, Inc.</i> , No. 6:24-cv-188-ADA (WDTX)
Ex.1026	IV's Responsive Claim Construction Brief, <i>Intellectual Ventures II, LLC v. Tesla, Inc.</i> , No. 6:24-cv-188-ADA (WDTX)
Ex.1027	<i>The SPARC Architecture Manual</i> , Version 9, Prentice-Hall, 2000

I. INTRODUCTION

Pursuant to 35 U.S.C. §§311, 314(a), and 37 C.F.R. §42.100, Tesla, Inc. (“Petitioner”) respectfully requests that the Board review and cancel as unpatentable under (pre-AIA) 35 U.S.C. §103(a) claims 1-2, 5, 7-8, and 11 (the “Challenged Claims”) of U.S. Patent 8,898,395 (the “’395 patent,” Ex.1001).

The ’395 patent is directed to memory consistency, including management of cache memory, in a shared memory multiprocessor. Ex.1001, 1:15-17, 27-35. It is directed to multiprocessor techniques for coordinating transactional (atomic) access to sets of multiple independently-selected memory locations—a technical field known as transactional memory. The ’395 patent is concerned with “high-level processor architectures...[in which] operation reordering, optimization and speculation” are supported and in which “instructions are lumped into instruction groups that can be committed and rolled back atomically.” Ex.1001, 1:47-62.

These concepts were already known in the art, as shown by Moir and Martínez, prior art that was not considered during prosecution of the ’395 patent. Petitioner respectfully requests that the Board institute *inter partes* review (“IPR”) and cancel as unpatentable the Challenged Claims of the ’395 patent.

II. CERTIFICATION OF GROUNDS FOR STANDING

Petitioner certifies that the ’395 patent is available for *inter partes* review (IPR) and that Petitioner is not barred or estopped from requesting IPR of the

patent's claims.

III. NOTE

Petitioner cites to exhibits' original page numbers, unless noted otherwise.

Emphasis in quoted material has been added. Claim terms are presented in *italics*.

IV. TECHNOLOGY OVERVIEW

The '395 patent relates to transactional memory techniques. Techniques to implement transactional memory (*e.g.*, as described by Herlihy and Moss) were known for more than a decade prior. Ex.1003, ¶35; Ex.1012, 289 *et seq.* (Herlihy and Moss Paper describing transactional memory as multiprocessor architecture that may be implemented by straightforward extensions to a multiprocessor cache coherence protocol and that allows programmers to define customized read-modify-write operations that apply to multiple, independently-chosen words of memory); Ex.1013, 3:25-35 (Herlihy and Moss Patent describing transactional memory as providing the ability to implement customized read-modify-write operations that affect arbitrary regions of memory). By 2004, transactional memory and speculative execution techniques had been widely discussed in the academic literature. Dr. Colwell's declaration describes the relevant background technology predating the '395 patent. Ex.1003, ¶¶35-38.

V. THE '395 PATENT

A. Overview

The '395 patent is concerned with managing memory consistency in multiprocessors in which “operation reordering, optimization and speculation” are facilitated by allowing instruction groups to be “committed and rolled back atomically.” Ex.1001, 1:47-62. POSITA would have understood these multiprocessor techniques for coordinating transactional (atomic) access to sets of memory locations to be examples of transactional memory. Ex.1003, ¶23.

More specifically, the '395 patent is directed to “maintain[ing] sequential consistency for lumped (in-order or out-of-order) [execution in multiprocessor] architectures.” Ex.1001, 1:66-2:1, 1:47-48. The '395 patent indicates that “[p]rior attempts to achieve a sequentially consistent, lumped architecture are limited...to a single memory operation per instruction group.” Ex.1001, 2:1-5. To “permit sequentially consistent, lumped architectures in which multiple memory operations can be included in instruction groups,” “observed bits” are provided (in addition to the conventional cache coherence protocol bits) in cache line tags associated with the cache lines of a cache of a processor. Ex.1001, 8:51-53, 4:44-46, 33-34. The '395 patent's Fig. 2 below illustrates an example of a cache and its associated cache line tags including the observed bits. Ex.1001, 4:32-60; Ex.1003, ¶24.

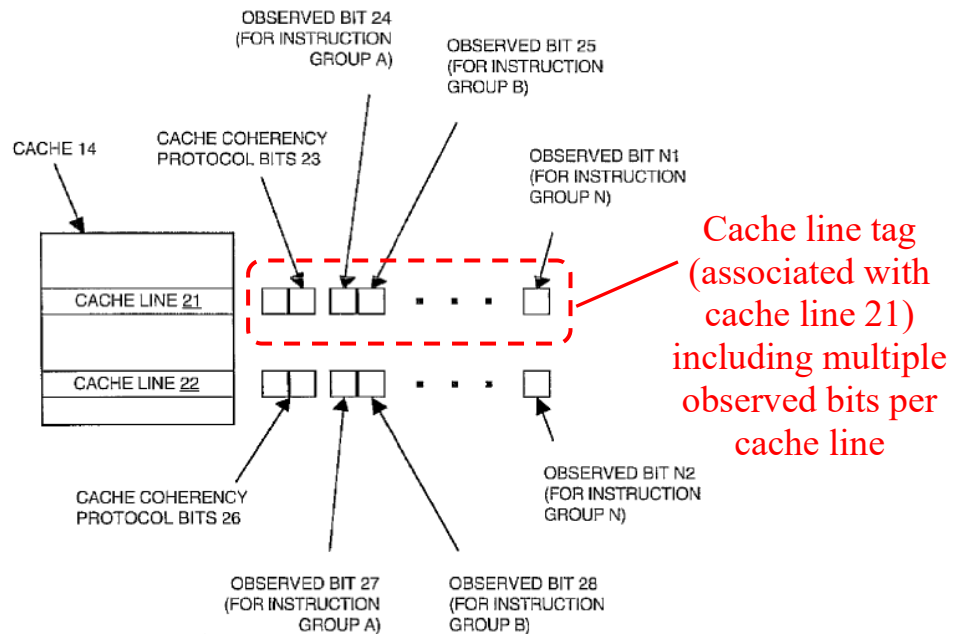


Figure 2

Ex.1001, Fig. 2 (annotated); Ex.1003, ¶24

Fig. 2 shows a cache 14 of a processor having a first cache line 21 and a second cache line 22, where the first cache line 21 is associated with coherency protocol bits 23 and observed bits 24, 25...N1, and the second cache line 22 is associated with coherency protocol bits 26 and observed bits 27, 28...N2. Ex.1001, 4:32-48. The cache coherency protocol bits 23 and 26 are “used to indicate the state of the cache lines 21 and 22, respectively, according to a cache coherency protocol such as the MESI [Modified, Exclusive, Shared, and Invalid] protocol.” Ex.1001, 4:41-43; Ex.1003, ¶25.

Each observed bit (e.g., observed bit 24 for the first cache line 21) extends the

cache line tag of a cache line for each group of instructions that may be executed by the processor (e.g., processor 10 with its associated cache 14). Ex.1001, 4:51-56. For example, in Fig. 2, the cache line tag associated with cache line 21 is extended by N bits for N groups of instructions that may be executed by processor 10 associated with cache 14. Ex.1001, 4:51-56. Groups of instructions are committed or rolled back atomically—*i.e.*, as a unit, all together or not at all. Ex.1001, 1:51-60, 4:11-12; Ex.1003, ¶26; Ex.1024 (defining “atomic transaction [as a] set of operations that follow an ‘all or nothing’ principle, in which either all of the operations are successfully executed or none of them is executed.”).

In this way, for a given instruction group, either all instructions are executed to completion and all state changes are committed, or the entire group of instructions is atomically rolled back. Ex.1001, 4:11-12, 7:59-62. If execution of an instruction in a group of instructions causes a cache line to be accessed (e.g., a load, store, read, or write), the observed bit is set. The particular observed bit set is that associated with the particular group of instructions and belonging to the cache line tag of the particular cache line accessed. Ex.1001, 4:61-65; Ex.1003, ¶26. With respect to Fig. 2, for example, “observed bit 24 is set if an instruction in instruction group A, when executed by processor 10, causes cache line 21 to be accessed.” Ex.1001, 4:66-5:1, Fig. 2; Ex.1003, ¶26.

The ’395 patent explains that “when execution of a group of instructions is

ended (e.g., the instruction group is committed, rolled back, or aborted), each observed bit set by that group of instructions is cleared....” Ex.1001, 5:19-27. A group of instructions is atomically roll backed and reissued when the following two conditions are met:

- (1) an observed bit (e.g., observed bit 24) is set (indicating that execution of an instruction of the group of instructions associated with the observed bit (e.g., instruction group A) has caused the associated cache line (e.g., cache line 21) to be accessed), and
- (2) another processor or a peripheral device executes an instruction that requests access to that same cache line (e.g., cache line 21) or causes the cache line (cache line 21) to be accessed.

Ex.1001, 5:50-56; Ex.1003, ¶27.

B. Prosecution History

The '395 patent issued from Application No. 12/121,531 (the '531 application). The '531 application is a continuation of Application No. 11/102,127 (the '127 application), which issued as U.S. Patent 7,376,798 (the '798 patent).

Ex.1002, 235, 1; Ex.1007, 2.

1. Prosecution History of the '798 Patent

The '127 application filed with claims reciting the use of an indicator, associated with a first group of instructions, to indicate a cache line of a cache

memory in a computer system has been read when an instruction of the first group of instructions is executed:

1. A method of managing memory in a computer system, said method comprising:

executing a first group of instructions, wherein said first group of instructions comprises multiple memory operations and a first instruction that when executed causes a cache line of a cache memory to be read; and

in response to said first instruction causing said cache line to be read, changing an indicator associated with said first group of instructions to indicate that said cache line has been read, wherein said cache line is so indicated as having been read until execution of said first group of instructions is ended.

Ex.1007, 178-186.

The Examiner rejected all claims as being anticipated and obvious. Ex.1007, 117-133. The applicant amended the claims to recite that “multiple groups of instructions are executable in parallel by [the processor and that] said indicator comprises a bit per cache line for each group in said groups of instructions,” (Ex.1007, 102), and further amended the claims to recite (1) a second group of instructions and (2) a first and second bit of the indicator, where each such bit is set to indicate that a respective first or second group of instructions has caused the cache line to be read. Ex.1007, 43-53. The Examiner then allowed the amended claims, stating that the limitation “in response to said second instruction causing said cache line to be read, setting a second bit of said indicator while said first bit remains set to indicate said cache line has been accessed by both said first group and said second

group of instructions” was not taught by the prior art of record. Ex.1007, 17-18.

Allowed claim 1 of the '127 application read as:

1. (Currently Amended) A method of managing memory in a computer system, said method comprising:

executing a first group of instructions on a first processor, wherein said first group of instructions comprises multiple memory operations and a first instruction that when executed causes a cache line of a cache memory to be read; and

in response to said first instruction causing said cache line to be read, ~~changing~~ setting a first bit of an indicator associated with said cache line first group of instructions to indicate that said cache line has been read, wherein said cache line is so indicated as having been read until execution of said first group of instructions is ended, wherein multiple groups of instructions are executable in parallel by said first processor and wherein said indicator comprises a bit ~~per~~ cache line for each group in said groups of instructions;

executing a second group of instructions on said first processor, wherein said second group of instructions comprises multiple memory operations and a second instruction that when executed causes said cache line to be read; and

in response to said second instruction causing said cache line to be read, setting a second bit of said indicator while said first bit remains set to indicate said cache line has been accessed by both said first group and said second group of instructions.

Ex.1007, 43.

2. Prosecution History of the '395 Patent

The '531 application, which issued as the '395 patent, was originally filed with claims incorporating elements found in the allowed claims of its parent application, including the first group of instructions and the second group of instructions, as well as the cache-line associated first and second bits of the indicator.

Ex.1002, 241. The Examiner rejected all the claims for obviousness-type double patenting and rejected independent claims as being anticipated and rendered obvious by prior art references, while indicating some of the dependent claims contained allowable subject matter. Ex.1002, 197-212.

The Applicant amended some of the claims; however, several claims (including all the independent claims) were again rejected as obvious. Ex.1002, 179-187 (claim amendment), 151-165 (rejections). The applicant appealed the rejections, but the Board affirmed the rejections. Ex.1002, 66-117 (appeal), 58-63 (appeal decision). Application claim 35, which depends on application claim 33 via intervening application claim 34, was one of the claims indicated as containing allowable subject matter. Ex.1002, 165-166. Claims 33-35 read as:

33. (Previously Presented) A method of managing memory in a computer system, said method comprising:
 setting a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions; and
 setting a second bit of said indicator while said first bit remains set if said cache line has also been accessed in response to said processor executing an instruction in a second group of instructions.
34. (Previously Presented) The method of Claim 33 further comprising:
 executing a third group of instructions that causes said cache line to be accessed; and
 processing said first group and said second group of instructions according to a value of said indicator.
35. (Previously Presented) The method of Claim 34 wherein said third group of instructions is executed by an agent other than said processor, wherein said processing comprises rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set before allowing an instruction in said third group to access said cache line, and otherwise granting said access.

antecedent
basis

Ex.1002, 135 (annotated to emphasize antecedent basis¹).

After the Board's decision, the Applicant incorporated the subject matter of dependent claims into the independent claims to place the application in condition for allowance. Ex.1002, 31-43. Application claims 34 and 35 were incorporated into

¹ As acknowledged in the Claim Construction section (§VII), Patent Owner appears to have a different position regarding antecedent basis for "said processing" in the issued claim language.

independent claim 33, which issued as claim 1 of the '395 patent. Ex.1002, 32; Ex.1001, Claim 1. The Examiner identified the limitations of claim 35 and the limitation of claim 34 reciting, “executing a third group of instructions that causes said cache line to be accessed,” as the limitations that were not taught by the prior art considered during prosecution. Ex.1002, 13. Independent claim 1 of the '395 patent corresponds to application claim 33 (with limitations of claims 34 and 35 incorporated therein).

However, claim 1 recites well-known concepts that were taught by prior art not considered by the Examiner. Ex.1003, ¶¶28-34.

3. Note Regarding Alleged Point of Novelty

The alleged novelty, according to the '395 patent as issued (and the parent '127 application), was handling more than a single memory operation per instruction group. Ex.1001, 2:1-7 (alleging that “[p]rior attempts to achieve sequential consisten[cy were]...limited to a single memory operation per instruction group” and that “[e]mbodiments in accordance with the present invention overcome this disadvantage”); Ex.1007, 160 (same explanation in '127 application, as filed). During prosecution of the '127 application, when faced with clear teaching of prior art U.S. Patent 6,938,130 to Jacobsen—a transactional memory reference—the applicant pivoted to assert that novelty instead derived from the utilization of “multiple indicator bits associated with each cache line in a cache.” Ex.1007, 54-55

(arguing that “Jacobsen appears to utilize only one ‘store-marking’ bit per cache line, in contrast to the claimed invention”).

However, neither idea—not multiple memory operations per instruction group and not multiple indicator bits per cache line—was novel in April of 2005. Ex.1003, ¶¶37-38 (citing Ex.1012, Ex.1013, discussing Herlihy and Moss’s transactional memory teachings). More than a decade before the ’395 patent’s priority date (April 7, 2005), extensions to conventional cache coherence techniques using bits associated with cache lines to mark memory locations involved in transactional accesses, to detect memory access conflicts between groups of instructions, and to atomically commit or abort multi-instruction sequences were all well-documented. Claim recitations of multiple bits per cache line, of rollback, and of MESI-based cache coherence protocols are mere design or linguistic variations on the basic techniques and terminology, as Dr. Colwell explains. Ex.1003, ¶¶37-38. As detailed in the grounds which follow, all limitations of the challenged claims are disclosed and rendered obvious by Moir (Ex.1005) and Martínez (Ex.1006)—two complementary prior art references that detail concrete implementations of Herlihy and Moss’s transactional memory in modern shared memory multiprocessors.

VI. LEVEL OF ORDINARY SKILL IN THE ART

Persons of ordinary skill in the art at, and before, the earliest claimed priority date of the ’395 patent (April 7, 2005) (“POSITA”) would have had a bachelor’s

degree in electrical engineering, computer engineering, computer science, or a related field and 2-3 years of experience with speculative execution of instruction groups or threads in a shared memory multiprocessor. Less experience may be sufficient with additional education, and likewise, less education may be sufficient with additional work experience. Ex.1003, ¶20.

VII. CLAIM CONSTRUCTION

Claim terms in IPR are construed according to their “ordinary and customary meaning” to those of skill in the art. 37 C.F.R. §42.100(b).

A. “said processing” (claims 1, 2)

In ongoing litigation between the parties, antecedent basis for the term “said processing” is contested.² Thus, proper treatment of limitations that involve “said processing comprises rolling back said first group of instructions...and rolling back said second group of instructions” (*see infra*, limitations [1.3B], [2.1]-[2.2]) will likely require claim construction. *See Abbott Diabetes Care Inc. v. Dexcom, Inc.*, IPR2024-00890, Paper 9, n. 4 (Dec. 16, 2024) (“[c]laim construction arguments

² Ex.1025, 18-24 (arguing that issued claim 1 introduces confusion as to antecedent basis and therefore scope); Ex.1026, 19-23 (Patent Owner arguing definiteness based on proximity to phrase “executing a third group of instructions...by an agent other than said processor”).

should not be relegated to patentability arguments on the facts,” *citing* CTPG 46, 49; *O2 Micro Int’l Ltd. v. Beyond Innovation Tech. Co.*, 521 F.3d 1351, 1361 (Fed. Cir. 2008)).

Although this term lacks proper antecedent basis, for purposes of this proceeding³ and to demonstrate applicability of the grounds to the claims, Petitioner construes “*said processing*” in [1.3B] as referring to the [1.4] “*processing of said first and said second group of instructions according to a value of said indicator,*” *i.e.*, the only other mention of *processing* in the claim.

Given the textual ambiguity of the claim language itself, Petitioner relies on the prosecution history. *See Nautilus, Inc. v. Biosig Instruments, Inc.*, 572 U.S. 898, 911 (2014). As detailed above (*see supra*, Section V.B.2—Prosecution History of the ’395 Patent), claim 1 issued from subject matter recited in a dependent claim (claim 35) that was indicated allowable if rewritten in independent form. Ex.1002, 156, 165 (indicating claim 35 would be allowable if rewritten in independent form to include intervening and rejected claims 33-34). The relevant claims 33-35 have been annotated (below) to demonstrate the antecedent basis of “*said processing*” in the allowable subject matter.

³ Petitioner reserves the right to challenge the claim language as indefinite or otherwise invalid under 35 U.S.C. §112 in other proceedings.

33. (Previously Presented) A method of managing memory in a computer system, said method comprising:
- setting a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions; and
 - setting a second bit of said indicator while said first bit remains set if said cache line has also been accessed in response to said processor executing an instruction in a second group of instructions.
34. (Previously Presented) The method of Claim 33 further comprising:
- executing a third group of instructions that causes said cache line to be accessed; and
 - processing said first group and said second group of instructions according to a value of said indicator.
35. (Previously Presented) The method of Claim 34 wherein said third group of instructions is executed by an agent other than said processor, wherein said processing comprises rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set before allowing an instruction in said third group to access said cache line, and otherwise granting said access.

antecedent
basis

Ex.1002, 135 (claims from after-final response annotated to trace antecedent basis); Ex.1003, ¶42 (Dr. Colwell confirming POSITA would have understood “said processing” in allowable claims to refer to antecedent language of claim 34, as annotated). Applicant appealed from the Examiner’s final rejection, but the rejection was affirmed by the Board. Ex.1002, 122, 63.

After the Board affirmed the Examiner’s rejection, the Applicant acquiesced, acknowledging the prior indication that claim 35 was allowable if rewritten, and representing that “Claim 33 is amended to incorporate the subject matter of

allowable Claim 35 and intervening Claim 34.” Ex.1002, 122, 40-41. Claim 33 so amended⁴ issued as claim 1 of the ’395 patent. Ex.1002, 32; Ex.1001, Claim 1.

Nautilus indicates that the proper analysis should also consider the specification insofar as it informs with reasonable certainty claim scope. 572 U.S. at 911. However, the ’395 specification fails to inform POSITA with any reasonable certainty which structure performs “*said processing*” that “*comprises rolling back said first group of instructions...and rolling back said second group of instructions.*” See, e.g., Ex.1001, 5:54–56 (describing the result—“then the instruction groups associated with the observed bits that are set are forced to roll back”—without identifying the responsible structure), 5:58–61 (same), 6:6–11 (same), 7:59–62 (same, with reference to step 43 in Figure 4); Ex.1003, ¶41. Accordingly, based on

⁴ The dispute between the parties in the co-pending litigation regarding proper construction (including indefiniteness) stems from ambiguity created by the Applicant itself when it changed the order of phrases in the rewritten claim to place the limitation, “*said processing comprises rolling back said first group of instructions...and rolling back said second group of instructions...*” (compare allowable claim 35 with [1.3B], *infra*), *before* its antecedent in the allowable claims, “*processing said first group and said second group of instructions according to a value of said indicator*” (compare intervening claim 34 with [1.4], *infra*)).

the foregoing, in the grounds that follow, limitation [1.4] is addressed prior to limitation [1.3] so as to present a coherent analysis of the term “said processing,” consistent with the construction and prosecution history.

In addition, this Petition addresses Patent Owner’s apparent interpretation in support of definiteness that “*said processing*” refers to the “*executing*” step. *See infra*, [1.3B-alternate]). As demonstrated below, regardless of the applied construction, the claims are obvious in view of Moir and Martínez.

Petitioner submits that, for the purposes of this proceeding and the grounds presented herein, no other claim term requires express construction. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017). Ex.1003, ¶43.

VIII. THE CHALLENGED CLAIMS ARE UNPATENTABLE

A. Challenged Claims

Petitioner challenges claims 1-2, 5, 7-8, and 11.⁵ Each reference below predates April 7, 2005, the earliest priority date of the ’395 patent.

B. Prior Art Patents and Printed Publications

1. U.S. Patent 7,206,903 to Moir et al. (“Moir,” Ex.1005); filed July 20, 2004; issued April 17, 2007; prior art under §102(e).

⁵ The pre-AIA statutory framework applies to the Challenged Claims.

2. “Speculative synchronization: programmability and performance for parallel codes” to J. F. Martínez et al., IEEE Micro (“Martínez,” Ex.1006); dated December 2003. As explained in Section IX below, Martínez was publicly available at least by January 21, 2004. Martínez is prior art under §§102(a) and 102(b).

C. Statutory Grounds for Challenges

This Petition, supported by the declaration of Dr. Robert Colwell (Ex.1003), demonstrates unpatentability of the Challenged Claims on the following Ground:

Ground	Claim	Basis (Pre-AIA)
#1	1-2, 5, 7-8, 11	§103 over Moir in view of Martínez

D. Ground 1: Claims 1-2, 5, 7-8, and 11 would have been obvious over Moir and Martínez

1. Moir

Moir is directed to techniques for improving performance of shared memory multiprocessor computer systems by managing accesses to memory locations during transactional program execution. Ex.1005, 1:25-34. Moir describes techniques that seek to avoid the overhead and complexity of acquiring and releasing locks to coordinate multithreaded access to shared data structures. Ex.1005, 1:43-2:4. Moir “transactionally execute[s] a critical section, wherein changes made during the transactional execution are not committed to the architectural state of the processor

until the transactional execution completes without encountering an interfering data access from another thread.” Ex.1005, 2:23-29. During transactional execution, cache lines that are accessed are marked, allowing interfering data access to be determined. Ex.1005, 2:34-44.

Moir proposes advanced techniques to “facilitate early release of memory locations during transactional program execution,” to “reduce[] the number of cache lines that need to be marked during transactional program execution” using “explicit time-to-live value” codings, to accommodate nested transactions. Ex.1005, 2:20-22, 1:27-29, 3:1-6, 3:35-38, 4:11-22; Ex.1003, ¶¶44-45. However, Moir also describes the basics of a transactional memory implementation, delimiting a group or critical section of instructions to be atomically committed or rolled back by “execut[ing] a start-transactional-execution (STE) instruction before entering the critical section. If the critical section is successfully completed without interference from other threads, the thread performs a commit operation, to commit changes made during transactional execution.” Ex.1005, 7:26-32.

Moir’s Fig. 3 below illustrates the transactional execution process, where “[a] thread first executes an STE instruction prior to entering of a critical section of code (step 302). Next, the system transactionally executes code within the critical section, without committing results of the transactional execution [block 304].” Ex.1005, 7:63-67; Ex.1003, ¶46. If no interfering data access occurs, changes made during the

transactional execution are committed (step 308); otherwise, if interfering data access does occur, changes made during the transactional execution are discarded (step 312). Ex.1005, 8:1-12; Ex.1003, ¶46 (explaining that discarding changes in Moir’s Fig. 3 is roll back).

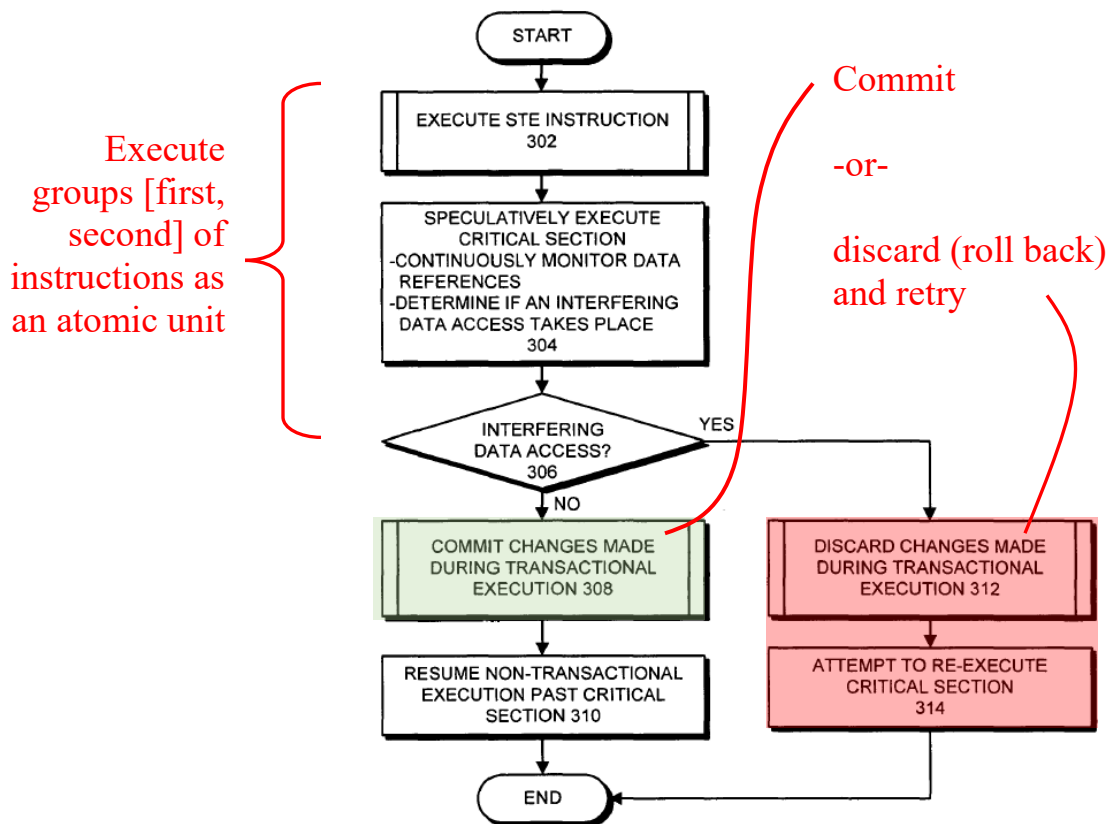


FIG. 3

Ex.1005, Fig. 3 (annotated); Ex.1003, ¶46

Transactional execution after “an STE instruction prior to entering of a critical section of code,” shown as step 302 in Fig. 3 above, “involves load-marking and store-marking cache lines, if necessary, as well as monitoring data references in order to detect interfering references.” Ex.1005, 7:63-64, 8:65-67. Fig. 1 of Moir

below shows a “computer system 100 [that] includes processors 101 and level 2 (L2) cache 120,” where “[p]rocessor 101 additionally includes a level one (L1) data cache 115, which stores data items that are likely to be used by processor 101.” Ex.1005, 6:9-11. “[L]ines in L1 data cache 115 include load-marking bits 116, which indicate that a data value from the line has been loaded during transactional execution. These load-marking bits 116 are used to determine whether any interfering memory references take place during transactional execution.” Ex.1005, 6:29-34. The computer system 100 can have additional processors such as processor 102 which are similar to processor 101. Ex.1005, 6:11-13; Ex.1003, ¶47.

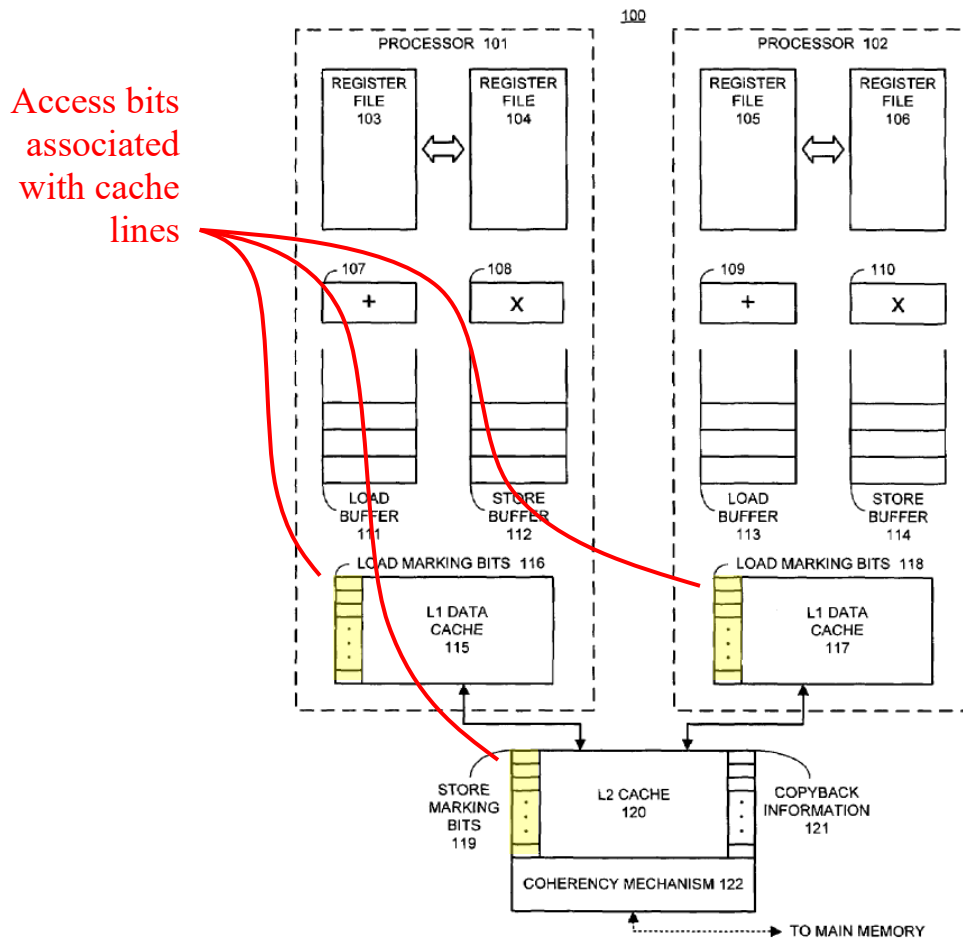


FIG. 1

Ex.1005, Fig. 1 (annotated); Ex.1003, ¶47

The load-marking operation is performed on a per-word basis within a cache line. “If [a] load causes a cache hit, the system ‘load-marks’ the **corresponding word** in the cache line in L1 data cache 115 []. This involves setting the load-marking bit for the cache line. Otherwise, if the load causes a cache miss, the system retrieves the word accessed in the cache line from further levels of the memory hierarchy [], and proceeds to step 506 to load-mark the cache line in L1 data cache 115.” Ex.1005, 9:6-16; Ex.1003, ¶48.

Moir provides for load marks on a word-granular basis. Ex.1005, Fig. 5 (“load mark word in cache line”), 4:5-7 (“a load-marked cache line contains a bit for each word indicating whether the word has been load-marked”), 4:36-29 (“system load-marks a corresponding word in the cache line to facilitate subsequent detection of an interfering data access to the cache line from another thread”), 9:5-16 (system load-marks the corresponding word in the cache line in L1 data cache). Accordingly, and because it is fundamental knowledge that a cache line contains multiple words, POSITA would have understood Moir to disclose multiple load marking bits per cache line. Ex.1003, ¶49. Store-marking is also performed, in which case “the system first prefetches a corresponding cache line for exclusive use.” Ex.1005, 9:23-25. Moir’s illustrated embodiment store-marks at an L2 cache for a configuration with write-through L1 semantics, but POSITA would understand write-marking at any appropriate level in a caching hierarchy. Ex.1003, ¶49.

Committing all changes made during transactional execution includes “the system treat[ing] store-marked cache lines as though they are locked []. This means other threads that request a store-marked line must wait until the line is no longer locked before they can access the line ... Next, the system clears load-marks from L1 data cache 115.” Further, the system “commits entries from store buffer 112 for stores that are identified as needing to be marked, which were generated during the transactional execution, into the memory hierarchy []. As each entry is committed,

a corresponding line in L2 cache 120 is unlocked.” Ex.1005, 9:65-10:9; Ex.1003, ¶50.

Discarding changes made during the transactional execution includes the system “clear[ing] load-marks from cache lines in L1 data cache 115 [], and drain[ing] store buffer entries generated during transactional execution without committing them to the memory hierarchy.” Ex.1005, 10:60-63. Further, “the system unmarks corresponding L2 cache lines.” Ex.1005, 10:65-66. In addition, “the system [may] branch[] to a target location specified by the outermost STE instruction []. The code at this target location optionally attempts to re-execute the critical section...or takes other action in response to the failure, for example backing off to reduce contention.” Ex.1005, 10:67-11:5; Ex.1003, ¶51.

Moir’s above-discussed techniques for handling critical sections of code allow “early release of memory locations during transactional program execution” because when “the system receives a release instruction during transactional execution of a sequence of instructions within an application,” the system clears a bit that is set and, “if all bits associated with a cache line are cleared, the cache line is unmarked by virtue thereof.” Ex.1005, 1:28-29, 13:4-6, 9-11. The benefits of early release include “cache lines that are unmarked [not having] to remain in cache memory until the transaction completes (or is killed), and false failures [being] less likely to occur due to a large number of cache lines being marked.” Ex.1005, 13:18-21; Ex.1003, ¶52.

2. Martínez

Martínez is directed to allowing “[a]pplication threads [to] execute speculatively past active barriers, busy locks, and unset flags instead of waiting.” Ex.1006, 126, 127. To do so, threads are extracted from an application and submitted “for speculative execution in parallel with a safe thread” of the application. Ex.1006, 126. This guarantees continual progress of the application’s execution because “safe threads cannot get squashed or stall” while “offending speculative threads [can be] squash[ed] and restart[ed] on the fly.” Ex.1006, 127 (about safe threads), 126 (about speculative threads); Ex.1003, ¶53.

Martínez discloses that its techniques for allowing processors to execute code speculatively can be implemented in shared-memory multiprocessors using a “speculative synchronization unit (SSU), which consists of some storage and some control logic that [Martínez adds] to the cache hierarchy of each processor in a shared-memory multiprocessor.” Ex.1006, 129. A processor uses the SSU to execute code speculatively, and “[w]hen a processor reaches an acquire point,” such as the one depicted in Fig. 1a, it requests the SSU for a lock. Ex.1006, 129. “The SSU sets its Acquire and Release bits, fetches the lock variable into its extra cache line,” and performs additional steps “to obtain lock ownership.” Ex.1006, 129; Ex.1003, ¶54.

After the processor crosses the acquire point, it “continues execution into the critical section. As long as the Acquire bit is set, the SSU deems speculative all the

processor's memory accesses after the acquire point in program order." Ex.1006, 130. When the processor accesses a cache line (e.g., of L1 or L2 above in Fig. 2) speculatively, "[t]he SSU uses the Speculative bit to locally mark cache lines that the processor accesses speculatively." Ex.1006, 130; Ex.1003, ¶55 (explaining that Martínez' speculative bits are analogous to Moir's load- and store-marking bits).

Access conflicts manifest as "a thread receiving an external invalidation to a cached line, or an external intervention (read) to a dirty cached line." Ex.1006, 130. An originator thread of such external messages to lines not marked speculative is not quashed whether it is a safe thread or a speculative one; however, "[i]f a speculative thread receives an external message for a line marked speculative, the SSU at the receiving node squashes the local thread." Ex.1006, 130; Ex.1003, ¶56 (explaining that cache coherence protocols use messages to convey state information). If the speculative thread that triggered the squash is an external read to a dirty speculative line in the cache, "the node replies without supplying any data. The coherence protocol then regards the state for that cache line as stale and supplies a clean copy from memory to the requester. This is similar to the case in conventional MESI (modified-exclusive-shared-invalid) protocols." Ex.1006, 130; Ex.1003, ¶56.

3. Analogous Art

Moir and Martínez are analogous art to the '395 patent because Moir and Martínez each pertain to the same field of endeavor as the '395 patent—processor

methods and techniques for managing memory consistency in a multiprocessor where groups of instructions execute atomically—also known as transactional execution or transactional memory. Ex.1003, ¶66.

The '395 patent describes this transaction approach to memory in terms of cache consistency and atomic commitment or rollback. Ex.1001, abstract (“Methods and systems for maintaining cache consistency are described.”), 1:15-17 (embodiments “relate to computer system memory, in particular the management of cache memory”), 1:51-60 (in lumped architectures, “instructions are lumped into instruction groups that can be committed and rolled back atomically”), 8:48-50 (seeking to “maintain sequential consistency for lumped [] architectures”); Ex.1003, ¶67.

Moir is likewise directed to this transactional memory technique. Ex.1005, 3:45-47 (Moir provides “a method and apparatus to facilitate early release of transactional memory”), abstract (providing “a system for releasing a memory location from transactional program execution” operating by “executing a sequence of instructions during transactional program execution”), 8:4-8 (Moir “atomically commits all changes made during transactional execution”); Ex.1003, ¶68.

Martínez also presents the transactional memory technique. Ex.1006, 127 (explaining that its thread-level speculation (TLS) mechanism builds on “[s]peculative synchronization [to] deal[] with explicitly parallel application threads

that compete to access a synchronized region” of memory. ... [I]f two speculative threads issue conflicting accesses, the hardware always squashes one of them and rolls it back to the synchronization point....A speculative thread keeps its (speculative) memory state in a cache until it becomes safe. At that time, it commits (makes visible) its memory state to the system.”), 128 (“all the speculative threads beyond the release point, one by one in some nondeterministic order, execute the critical section atomically”); Ex.1003, ¶69. Accordingly, Moir and Martínez are in the same filed of endeavor as, and analogous art to, the ’395 patent. Ex.1003, ¶69.

Moir and Martínez are also reasonably pertinent to a particular problem addressed by the ’395 patent, namely, maintaining sequential consistency despite interfering access by competing groups of multiple instructions that include multiple memory operations. Ex.1001, 1:66-2:7; *see also* Ex.1001, 2:17-21 (’395 patent stating that if “the cache line...is indicated as having been accessed, then the instruction group is rolled back and reissued”); Ex.1005, 8:9-12 (Moir stating that “if an interfering data access is detected, the system discards changes made during the transactional execution...and attempts to re-execute the critical section”); Ex.1006, 127 (Martínez stating that “[i]f two conflicting accesses cause a dependency violation, the hardware rolls the offending speculative thread back to the synchronization point and restarts it on the fly.”). Accordingly, and for this

reason as well, Moir and Martínez are analogous art to the '395 patent. Ex.1003, ¶70.

4. Reasons to Combine Moir and Martínez

As discussed above in Section VIII.D.1, Moir discloses that if there is an interfering data access (or other type of failure) during transactional execution of a critical section of code by a thread, changes made during the transactional execution are discarded, allowing the thread “to revert back to the conventional technique of acquiring a lock on the critical section before entering the critical section.” Ex.1005, 8:10-22. This is enabled by a “flash copy operation [that] checkpoints enough state to be able to restore the state to the state before the beginning of the outermost transaction⁶” and corresponding STE operation executed by the thread. Ex.1005, 8:55-57; Ex.1003, ¶71.

Moir does not explicitly discuss, however, the timing of the reversion or rolling back of the thread *vis-à-vis* the timing of the interfering data access. As such, although POSITA would recognize from Moir alone to revert a thread executing a

⁶ Moir’s implementation of transactional memory supports nested transactions. For purposes of the present unpatentability analysis of the Challenged Claims, restoring state applies with or without transaction nesting—including where “the outermost transaction” devolves to just “the transaction.” Ex.1003, ¶71.

critical section of code back to its state before the critical section when there is an interfering data access, the conventional implementation details of the *timing* of the reversion or roll back are not the focus of Moir. Ex.1003, ¶72 (explaining that once an interfering access is identified, further execution of the critical section by the interfered with thread is doomed and wasted computation). POSITA would have recognized that design choices regarding timing of reversion or roll back would affect overall system computational efficiency and that timing taught by Martínez beneficially avoids wasted computation. Ex.1003, ¶72.

POSITA therefore would have looked to Martínez for such conventional implementation details regarding timing of the reversion or roll back. As discussed above in Section VIII.D.2, Martínez suggests the roll back timing because it provides that the reversion or roll back of a thread having an access conflict would occur before supplying any data, e.g., stalling the requester pending rollback. More specifically, Martínez discloses that “[i]f a speculative thread receives an external message for a line marked speculative, the SSU at the receiving node squashes the local thread.” Ex.1006, 130. Further, “[o]nce the SSU triggers a squash, it...**forces the processor to restore** its checkpointed register state, which **results in the thread’s rapid rollback to the acquire point.**” Ex.1006, 130. And “[i]f an external read to a dirty speculative line in the cache triggered the squash, the node replies **without supplying any data.**” Ex.1006, 130. A “clean copy from memory [is then

supplied] to the requester.” Ex.1006, 130. Thus, Martínez provides implementation details of the timing of the reversion or roll back of a thread having an access conflict with another thread. Ex.1003, ¶73 (explaining benefit of timing to limit unnecessarily holding up useful computation).

POSITA would have been motivated to combine the teachings of Moir with Martínez because it is the combination of prior art elements (Moir teaching reverting a thread executing a critical section of a code back to a point before entering the critical section if there is an interfering data access from another thread, with Martínez teaching that such roll back be performed before supplying data to the external, and interfering, thread). Moreover, the combination with Moir involves known techniques according to known methods (rolling back a speculative thread before supplying data to another thread causing an access conflict with the speculative thread, as taught by Martínez) to yield the predictable result of reverting transactional execution of multiple critical sections based on an interfering data access from another thread and word-level marking of the involved cache line (as taught by Moir), before supplying data to the interfering thread. *See KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 416 (2007); Ex.1003, ¶74.

Further, POSITA would have had a reasonable expectation of success in making the combination because the combination provides the inevitable outcome of the rollback) in a manner that avoids unnecessary inefficiency or wasted

execution. As noted above, continued execution at the expense of delayed rollback would result in 1) wasted effort by the processor in executing instructions that will be rolled back, 2) stalling the execution of the thread where instructions are being executed due to the wasted effort, and 3) stalling the execution of other threads that need to access a dirty cache line until the cache line interference can be resolved. As Dr. Colwell notes, the combination of Moir and Martínez simply brings together a small set of design techniques that are individually already well understood and interoperable (e.g., marking words in a cache line, detecting an interfering access, observing standard cache coherence protocols, and reverting a processor's execution back to a previously saved state). *KSR*, 550 U.S. at 421; Ex.1003, ¶75. Accordingly, because Martínez provides efficiency-improving timing in the context of an analogous rollback, POSITA would have had a reasonable expectation of success in the combination. Ex.1003, ¶75.

POSITA also would have been motivated to combine the teachings of Moir with Martínez because Martínez describes the use of a known technique (coherence protocols that track the state of the cache line) to improve similar methods in the same way (as a coherency mechanism in Moir). Moir employs “a coherency mechanism,” and while Moir does not explicitly describe the conventional protocols involved, it does suggest an implementation of “coherency mechanism 122” and reference description of a co-pending application (published as Ex.1011). That

referenced application, in turn, provides an example of a cache coherence protocol . Ex.1005, 6:45-59; *see* Ex.1011, [0139] (describing a “cache coherency protocol [that] includes all of the usual state transitions between the following MOESI states: modified (M), owned (O), exclusive (E), shared (S) and invalid (I) states”); Ex.1003, ¶76 (citing Ex.1023, explaining cache coherence). Consistent with Moir’s reference to a co-pending application, Martínez itself expressly teaches a “coherence protocol [that] regards the state for that cache line as stale and supplies a clean copy from memory” in a manner similar to conventional MESI (modified-exclusive-shared-invalid) protocols. Ex.1006, 130. Thus, in making the combination, POSITA would have been motivated to implement a coherence protocol that uses the state for a cache line, per Martínez, in its coherency mechanism. Ex.1003, ¶76 (explaining that Moir’s suggestion, by reference, of a conventional coherence protocol is express in Martínez).

Moreover, POSITA would have had a reasonable expectation of success in making the combination because Moir explicitly calls for the use of a coherency protocol and, at minimum, strongly suggests implementing a conventional MESI/MOESI protocol. Ex.1005, 6:45-59. Martínez provides such a protocol, explaining that its approach “is similar to the case in conventional MESI (modified-exclusive-shared-invalid) protocols[.]” Ex.1006, 130. Ex.1003, ¶77 (citing Ex.1022, explaining coherence protocols). Accordingly, because Moir calls for a coherency

protocol and suggests that a conventional protocol would be implemented, and because Martínez provides such a coherence protocol, POSITA would have had a reasonable expectation of success in the combination. Ex.1003, ¶77.

5. Claim 1

[1.0] *A method of managing memory in a computer system, said method comprising:*

To the extent that the preamble, [1.0], is limiting, Moir discloses it. Ex.1003, ¶¶78-80.

Specifically, Moir discloses methods in the context of “a system for releasing a memory location from transactional program execution.” Ex.1005, abstract, 3:45-47. Thus, the disclosed methods by which that system operates constitute a *method of managing memory in a computer system*. More specifically, such methods cause “[t]he system to operate[] by executing a sequence of instructions during transactional program execution, wherein memory locations involved in the transactional program execution are monitored to detect interfering accesses from other threads, and wherein changes made during transactional execution are not committed until transactional execution completes without encountering an interfering data access from another thread.” Ex.1005, 3:45-54. Moir’s system is a *computer system*, and the release of memory locations involved in the execution of a transactional program thereon is a *method of managing memory in a computer*

system. Ex.1003, ¶79.

Thus, Moir discloses *a method of managing memory in a computer system.*

Ex.1003, ¶¶78-80.

[1.1] *setting a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions;*

Moir discloses limitation [1.1] because Moir describes and illustrates, relative to Fig. 5, “how load-marking is performed during transactional execution.” Ex.1005, 9:2-4, Fig. 5; Ex.1003, ¶¶81-89.

More specifically, Moir describes setting load-marking bits for an L1 cache line (*setting a first bit of an indicator associated with a cache line in a cache memory*) when a load accesses a cache line:

In performing this load operation [during transactional execution] ..., the system first attempts to load a data item from [level one] L1 data cache 115 (step 502). If the load causes a cache hit, the system ‘**load-marks**’ the corresponding word in the cache line in L1 data cache 115 (step 506). This involves **setting the load-marking bit** for the cache line. Otherwise, if the load causes a cache miss, the system retrieves the word accessed in the cache line from further levels of the memory hierarchy (step 508), and proceeds to step 506 to load-mark the cache line in L1 data cache 115.

Ex.1005, 9:5-16. Figure 5 of Moir, below, further reflects load-marking on the sub-cache line basis, *e.g.*, per word, within a cache line. Ex.1005, Fig. 5; *see also*

Ex.1005, Fig. 6 (likewise illustrating the analogous store-marking for write- or store-type memory access operations); Ex.1003, ¶82.

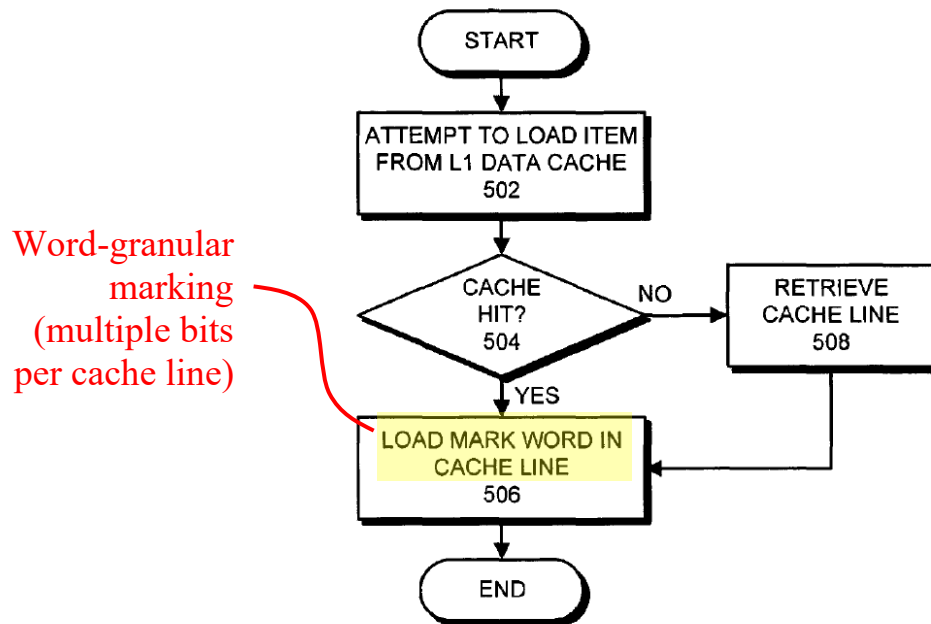


FIG. 5

Ex.1005, Fig. 5 (annotated); Ex.1003, ¶82

Moir discloses multiple load-marking bits per cache line because Moir describes load-marking of individual words within a cache line, and a cache line includes multiple words within its cache line width. Ex.1003, ¶83 (citing Ex.1020, explaining typical cache line widths and typical word counts per cache line). For example, in a SPARC-V9 instruction set architecture that would have been familiar to the Sun Microsystems' inventors in Moir, cache lines in the data cache were 64 bytes (512 bits) and integer load and store instructions supported byte, halfword (16-bit), word (32-bit), and doubleword (64-bit) accesses. Accordingly, POSITA would

understand Moir to be disclosing embodiments at least consistent with then-conventional, instruction set architectures (ISAs) such as SPARC-V9, *e.g.*, 16 words per cache line in the data cache and 16 word-granular, load-marking bits per cache line in the data cache. Ex.1003, ¶83 (citing Ex.1027).

Thus, the set of load-marking bits for an L1 cache line constitute *an indicator associated with that cache line*, and load marking a particular word corresponding to the addressable word target of a particular load access instruction is *setting a first bit⁷ of an indicator associated with a cache line in a cache memory*. Ex.1003, ¶84.

Moir describes and illustrates, relative to Fig. 1, an exemplary multiprocessor computer system 100. Ex.1005, 6:1-35 (explaining that Fig. 1 “illustrates a computer system 100” that includes processors 101 and 102). Each processor “includes a level one (L1) data cache” (*e.g.*, L1 data cache 115 and 117) that include “load-marking bits” (*e.g.*, load marking bits 116 and 118) “which indicate that a data value from the line has been loaded during transactional execution.” Ex.1005, 6:29-32. Moir explains that the load-marking bits “are used to determine whether any interfering

⁷ Ordinals *first* and *second* are understood to consistently identify respective bits throughout the claim language, without implication of an order or position of the respective bits within the indicator. *See, e.g.*, Ex.1001, 8:37-39 (“an observed bit can be set for each cache line accessed”).

memory references take place during transaction execution[.]” Ex.1005, 6:33-35.

Fig. 1 of Moir, below, shows illustrative processors 101 and 102 and the load-marking bits for each cache line of the respective L1 caches for the processors:

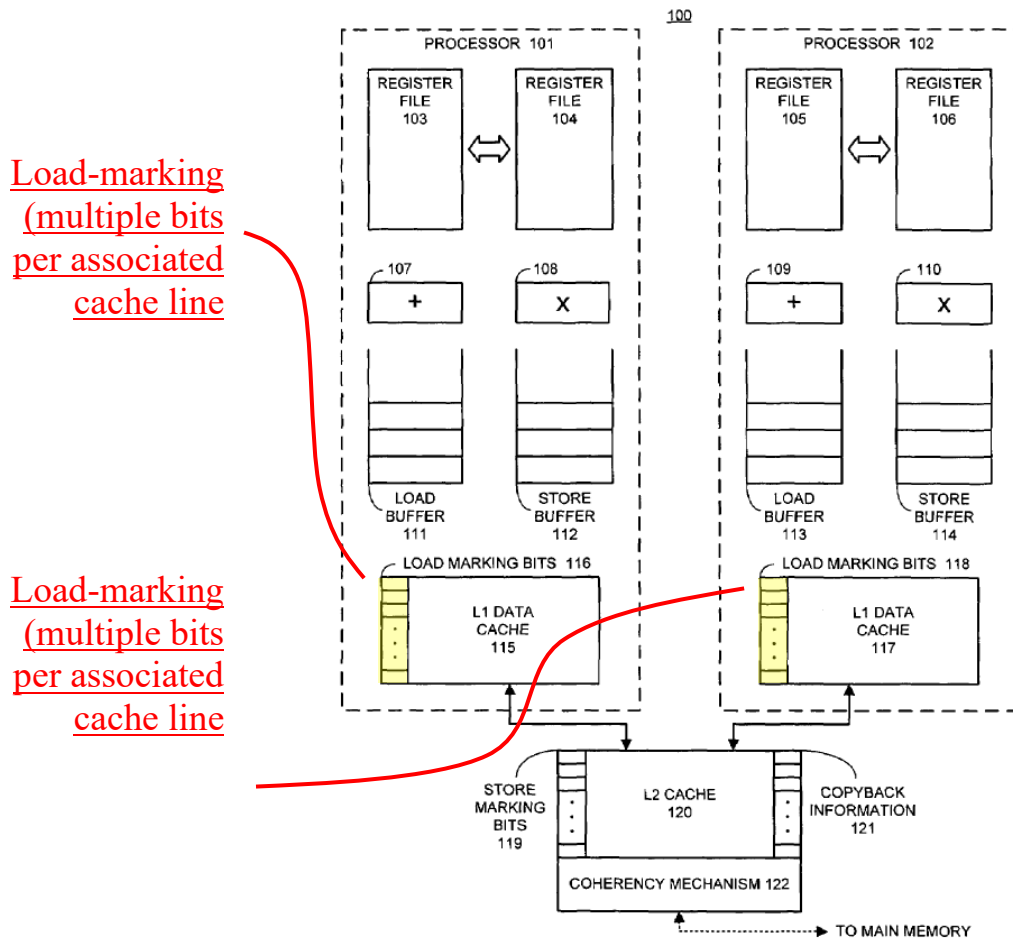


FIG. 1

Ex.1005, Fig. 1 (annotated); Ex.1003, ¶85

It would have been well-understood by POSITA that processors such as processors 101 and 102 execute instructions. Ex.1003, ¶86. For example, Moir discloses multiple instructions executed in its system that are pertinent to this discussion, including (1) a start-transactional-execution (STE) instruction (*see*

Ex.1005, 7:60-8:67, Figs. 3, 4); (2) a monitored load instruction (*see* Ex.1005, 11:59-12:10, Fig. 9B) that performs load-marking; and (3) a commit instruction (*see* Ex.1005, 9:48-10:13, Fig. 7). Ex.1003, ¶86. Figure 3, below, is “a flow chart illustrating how transactional execution takes place[.]” Ex.1005, 7:61-63.

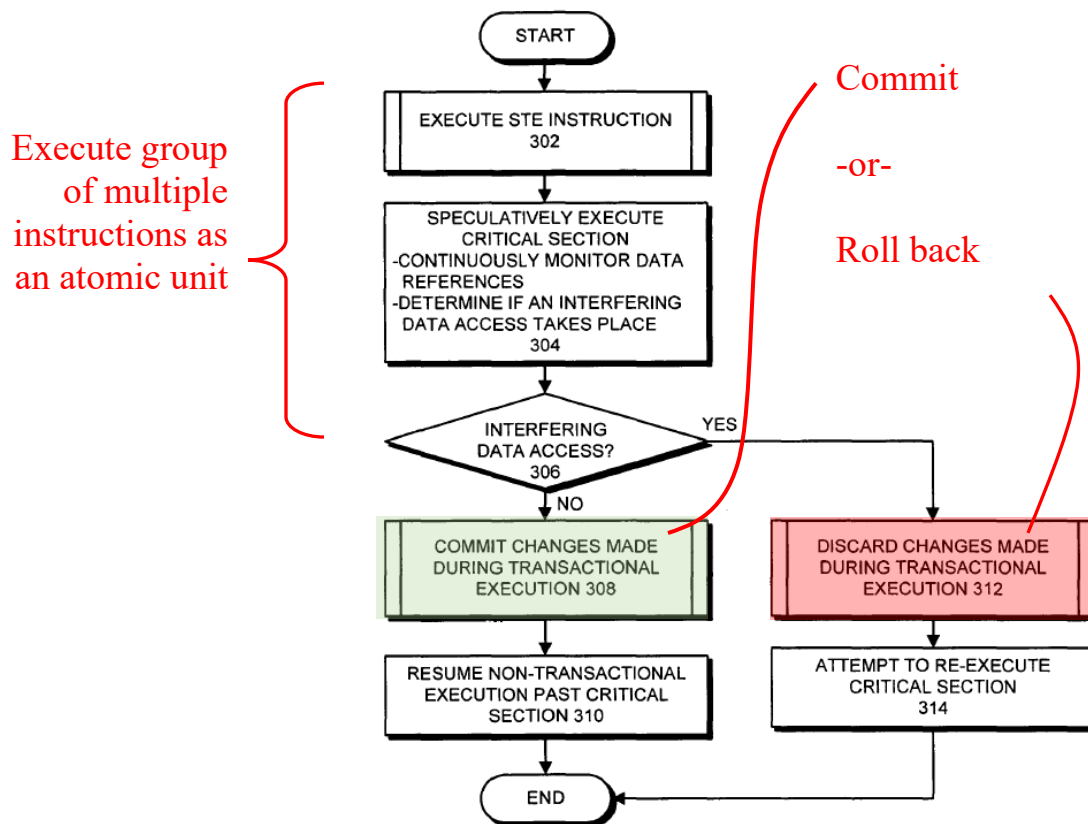


FIG. 3

Ex.1005, Fig. 3 (annotated); Ex.1003, ¶86

Moir also explains how its load-marking (and store-marking) of cache lines is used to detect interfering access and trigger roll back. As reflected in Fig. 3, “[a] thread first executes an STE instruction prior to entering of [sic] a critical section of code (step 302).” Ex.1005, 7:63-64. “During [] transactional execution, the system

continually monitors data references made by other threads, and determines if an interfering data access...takes place during transactional execution. If not, the system atomically commits all changes made during transactional execution.... On the other hand, if an interfering data access is detected, the system discards changes made during the transactional execution (step 312), and attempts to re-execute the critical section (step 314).” Ex.1005, 8:1-8:12. Moir explains that “that an interfering data access can include a store by another thread to a cache line that has been load-marked by the thread. It can also include a load or a store by another thread to a cache line that has been store-marked by the thread.” Ex.1005, 8:23-27.

Insofar as Moir’s transactional execution is concerned, POSITA would have understood that an STE instruction and those that follow in a critical section of instructions through commit or discard/re-execute constitute a *group of instructions* because multiple instructions handled together are a group. Ex.1003, ¶88. Moir specifically explains that “transactional execution...involves load-marking and store-marking cache lines, if necessary, as well as monitoring data references in order to detect interfering references.” Ex.1005, 8:64-67. Thus, the critical section that follows an STE instruction includes the monitored load instruction(s) that *sets a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions.* Ex.1003, ¶88.

Thus, Moir discloses or renders obvious *setting a first bit* (load marking a particular word corresponding to the addressable word target of a particular load access instruction) *of an indicator associated with a cache line in a cache memory* (set of load-marking bits for a cache line) *if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions* (following an STE instruction). Ex.1003, ¶¶81-89.

[1.2] *setting a second bit of said indicator while said first bit remains set if said cache line has also been accessed in response to said processor executing an instruction in a second group of instructions;*

Moir discloses or renders obvious limitation [1.2] because, as discussed for limitation [1.1], Moir describes and illustrates, relative to Fig. 5, “how load-marking is performed during transactional execution,” and Moir allows multiple critical sections to separately mark addressable words during transactional execution. Ex.1005, 9:2-4, Fig. 5; Ex.1003, ¶¶90-94.

For the reasons detailed above with respect to limitation [1.1], Moir’s set of load-marking bits for an L1 cache line constitute *said indicator*, and load marking another addressable word within *said (same) cache line* is *setting a second bit of said indicator*. Ex.1003, ¶91. Moreover, the first bit remains set and the setting of a second bit is therefore performed *while said first bit remains set* until the critical section that follows the STE instruction, which includes a monitored load that *set the first bit of said indicator* (as described above relative to limitation [1.1]), has

been committed or the mark has been released. Ex.1003, ¶91 (explaining Moir’s operation where multiple critical sections access addressable words within a same cache line). In general, a nested critical section of the same thread (or a critical section of another thread entirely) that includes a second monitored load instruction targeting a second addressable word of the same cache line (e.g., another addressable word representing another variable) also accesses the *same cache line* and, in Moir’s system, sets *a second bit* (the load-marking bit for the second addressable word) of the same cache line. Ex.1003, ¶91.

Moir specifically contemplates executing sequences of instructions during transactional program execution, “wherein memory locations involved in the transactional program execution are monitored to detect interfering accesses from other threads, and wherein changes made during transactional execution are not committed until transactional execution completes without encountering an interfering data access from another thread.” Ex.1005, 3:45-54. POSITA would have understood that in the ordinary course of multithreaded computations contemplated by Moir, nested critical sections (each constituting a group of instructions and each including a load-marking instruction) execute on a same processor. Ex.1003, ¶92; *see* Ex.1005, 8:38-67 (describing nested transaction support).

More generally, POSITA would have understood that another monitored load targeting another addressable word within the same cache line—whether as part of

the same critical section, or another thread executing its own critical section on the same processor after a context switch, or as second critical section nested within the first—constitutes an instruction that *also accesses said cache line in response to said processor executing that instruction in a second group of instructions*. Ex.1003, ¶93 (explaining that in each case, the *second group of instructions* is executed on *said* (same) *processor* as the first group of instructions).

Thus, because Moir allows multiple critical sections to separately mark addressable words within a cache line, and for the reasons explained above with reference to limitation [1.1], a second critical section that follows another STE instruction executed by the same processor (whether for a same or separate thread or as a nested critical section of the same thread) includes the monitored load instruction that *sets a second bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to said processor executing an instruction in a second group of instructions*. Ex.1003, ¶¶90-94.

[1.4]⁸ *processing said first group and said second group of instructions according to a value of said indicator.*

Moir discloses or renders obvious limitation [1.4] because Moir discloses that it atomically commits or discards, for transactionally executed sequences of instructions, based on the load-marking bits for an L1 cache line. Ex.1003, ¶¶95-100.

First, as explained above relative to limitations [1.1] and [1.2], Moir discloses or renders obvious transactional execution on a processor of first and second critical

⁸ As noted above in Section VII (Claim Construction), for purposes of this proceeding, limitation [1.4] (“***processing said first group and said second group of instructions***”) recites the textual antecedent for limitation [1.3] (“***said processing comprises rolling back said first...and said second group of instructions...***”). Accordingly, Petitioner treats the antecedent term “said processing,” first, in a manner consistent with the prosecution history. *See* Section VII; Ex.1003, ¶¶95, 41-43 (explaining how POSITA would have understood the term in context of allowable claim). Petitioner reserves its right to assert invalidity under pre-AIA 35 U.S.C. §112 in related proceedings before the district court. Petitioner’s treatment of the antecedent [1.4] first is consistent with the prosecution history, but does not suggest an order for recited aspects of the *processing*.

sections that each include the monitored load instructions which perform word-granular, load-marking of cache lines. *See* Ex.1005, 9:5-16 (as applied to load-marking of a first word of cache line accessed by first critical section and as also applied to load-marking of a second word of the same cache line accessed by a second critical section). These first and second critical sections constitute *first and second groups of instructions*, as claimed. Ex.1003, ¶96.

Second, Moir’s technique operates *according to a value of said indicator* because the first and second load-marking bits that together make up the indicator are used to decide whether to commit or roll back. Moir specifically describes relative to block 304 of Fig. 3, below, that it “transactionally executes code within the critical section[s], without committing results of the transactional execution.” Ex.1005, 7:65-67, Fig. 3. “During this transactional execution, the system continually monitors data references made by other threads, and determines if an interfering data access...takes place during transactional execution. **If not, the system atomically commits** all changes made during transactional execution (step 308) On the other hand, **if an interfering data access is detected, the system discards changes** made during the transactional execution (step 312), and attempts to re-execute the critical section[block 314].” Ex.1005, 8:1-12, *see also* Fig. 3.

Dr. Colwell explains that discarding changes made during transaction execution is rolling back. Ex.1003, ¶98 (comparing Moir’s discarding of changes

and re-execution with the '395 patent's roll back and reissue). Like the '395 patent, Moir atomically commits or rolls back (discards) transactionally executed sequences of instructions. Ex.1005, 7:65-67, 8:1-12; Ex.1003, ¶98 (explaining Moir's transactional execution process and Fig. 3). Moreover, like the '395 patent, Moir's decision regarding whether to atomically commit or roll back (discard) its transactionally executed sequences is based on monitoring to detect, using the cache line markings described above relative to [1.1] and [1.2], an interfering access by another thread. Ex.1005, 7:65-67, 8:1-12; Ex.1003, ¶98.

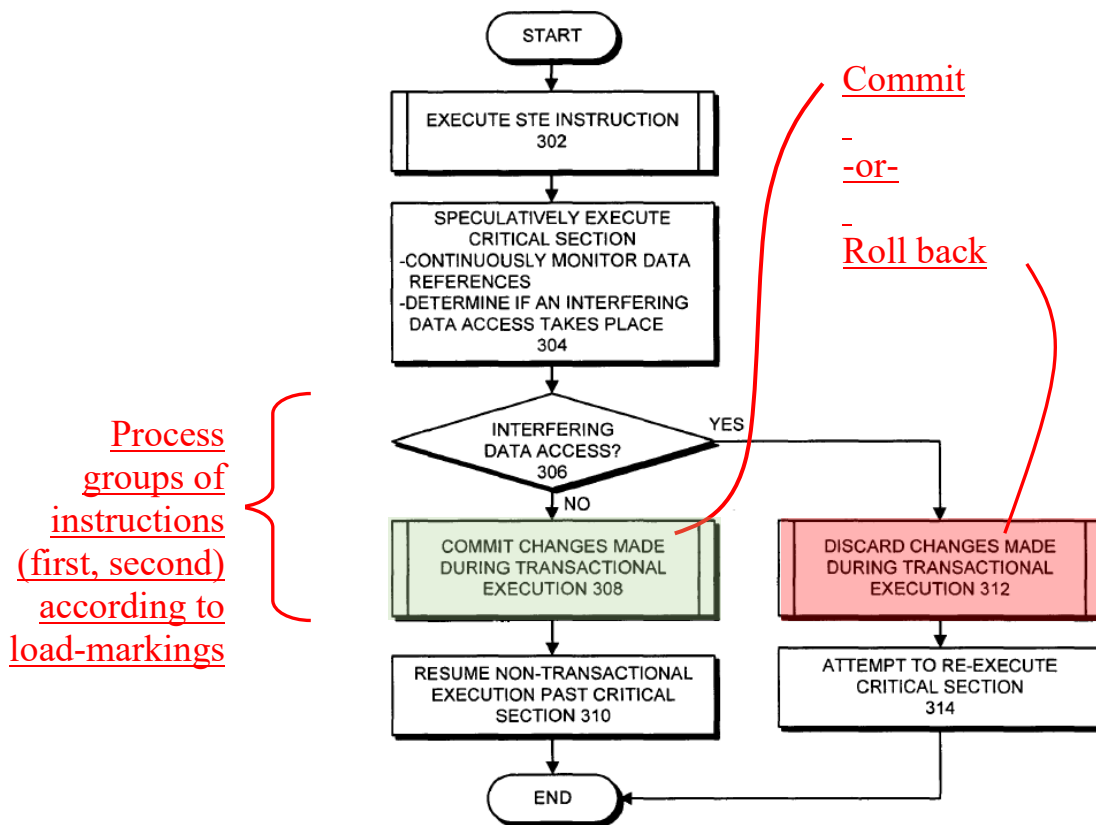


FIG. 3

Ex.1005, Fig. 3 (annotated); Ex.1003, ¶98

This process in Moir is *processing said first group and said second group of instructions according to* the word-granular, load-marking of L1 cache lines previously described. Ex.1003, ¶99. As explained in this section and for limitations [1.1], [1.2], Moir uses load-marking memory access instructions (monitored loads) to mark respective word-granular bits associated with a given cache line. And as further explained in Moir, “an interfering data access can include a store by another thread to a cache line that has been load-marked by the thread [e.g., by execution of a load marking memory access instruction]. It can also include a load or a store by another thread to a cache line that has been store-marked by the thread.” See Ex.1005, 8:23-27; Ex.1003, ¶99.

Thus, Moir’s decision to atomically commit or discard (such as for transactionally executed sequences of instructions relative to [1.1] and [1.2]) discloses or renders obvious *processing said first group and said second group of instructions according to a value of said indicator* (load marking bits associated with the L1 cache line). Ex.1003, ¶¶95-100.

[1.3A] *executing a third group of instructions that causes said cache line to be accessed, wherein said third group of instructions is executed by an agent other than said processor,*

Moir and Martínez render obvious limitation [1.3A] because both Moir and Martínez disclose speculative execution techniques for multithreaded computations in a shared memory multiprocessor, and as such, each contemplate *first, second,*

third ... and, indeed, *nth groups of instructions* executing as individual threads of a computation running on respective processors of a multiprocessor. Ex.1003, ¶101 (citing Ex.1021, noting that both Moir and Martínez describe multiprocessors in which memory access by a group of instructions executing on one processor interfere with access by instructions executing on another processor). As mapped above, the first and second groups of instructions execute on a first processor (*see supra*, discussion of limitations [1.1], [1.2]), while the third group of instructions executes on a second processor.

More specifically, Moir and Martínez each contemplate speculative execution of groups of instructions (*e.g.*, transactional execution of critical sections or threads) that access shared memory with interfering or conflicting accesses detected based on marked cache lines in which data corresponding to shared addressable memory locations are cached. *See* Ex.1005, Figs. 1, 3, 7:60-8:12, 8:23-35; *compare* Ex.1006, 127 (“[a] speculative thread uses its processor’s caches to buffer speculatively accessed data.... The hardware looks for conflicting accesses—accesses from two threads to the same location that include at least one write....”), Fig. 2, 129 (“local cache...serves as the buffer for speculative data.... [Speculative synchronization unit] SSU keeps one Speculative bit (S) per line in the local cache hierarchy...[and] sets the Speculative bit of a line when the processor reads or writes the line speculatively”). Ex.1003, ¶102.

As reflected in Moir's Fig. 1, below, Moir discloses a system implementing multiple processors, such as processor 101 and processor 102. Ex.1005, 6:1-36, Fig. 1. Moreover, the '395 patent itself establishes that its use of the term agent encompasses another processor. Ex.1001, 2:15-25 ("external agent (e.g., another processor or a DMA system)"), Fig. 3 (same), Fig. 1 (processors 11, 12 alternatively referenced as agents 11, 12 in associated description).

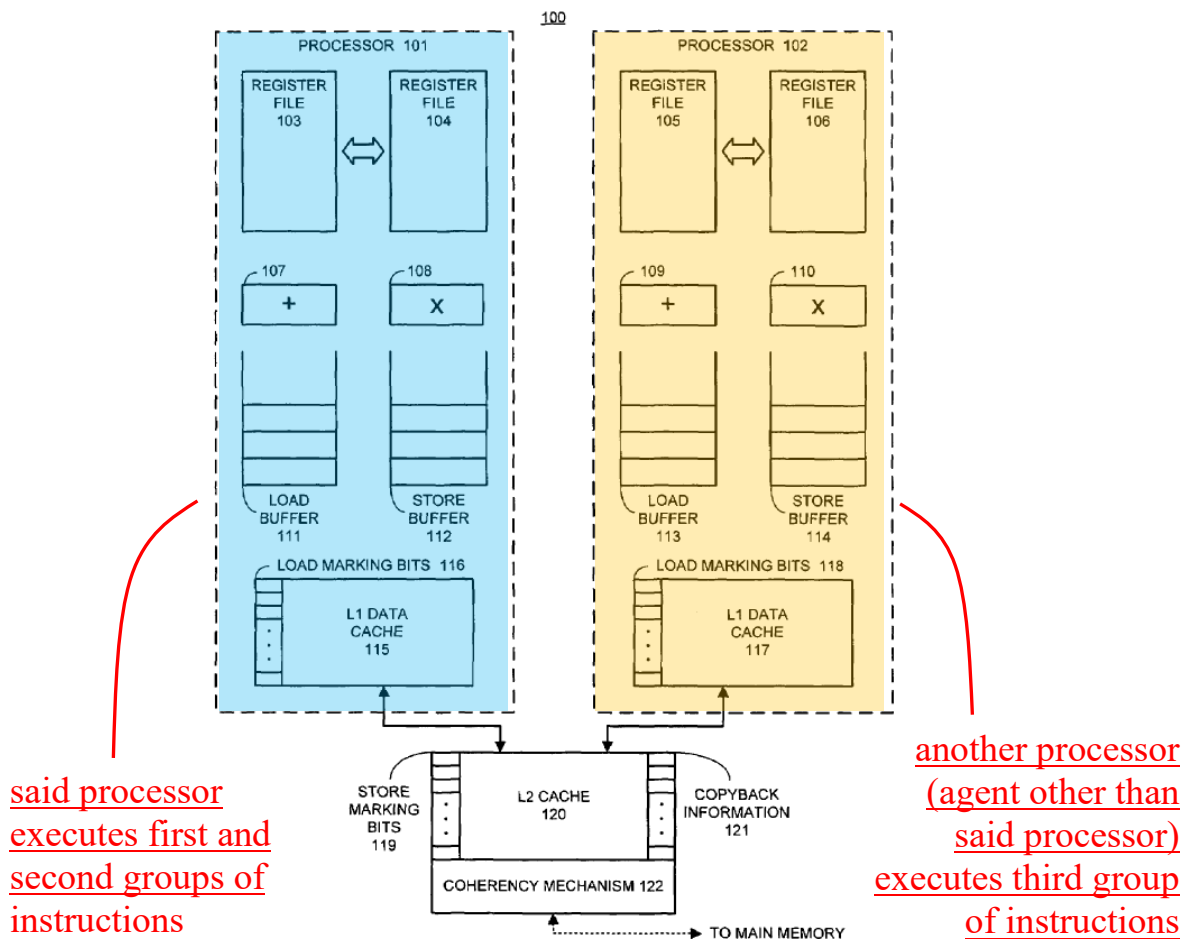


FIG. 1

Ex.1005, Fig. 1 (annotated); Ex.1003, ¶103

Further, Moir explicitly contemplates multiple processors accessing a given cache line because Moir explains that “conventional cache coherence circuitry presently generates signals indicating whether a given cache line has been accessed by another processor” to “determine whether an interfering data access has taken place.” Ex.1005, 8:30-35; Ex.1003, ¶104.

Thus, Moir discloses, and Moir in view of Martínez renders obvious,⁹ *executing a third group of instructions* (a thread with an interfering or conflicting instruction as in Moir or Martínez) *that causes said cache line to be accessed, wherein said third¹⁰ group of instructions is executed by an agent other than said processor* (e.g., a second processor 102 in Moir or any processor in Martínez on

⁹ Moir and Martínez disclose the basic framework of a multiprocessor in which memory access by a group of instructions executing on one processor interfere with access by instructions executing on another processor. The Moir-Martínez combination is noted here for consistency with the analysis of the next limitation of [1.3B] in which the timing of rollback is addressed given the combined teachings of the two references.

¹⁰ The ordinals *first*, *second* and *third* are understood to identify respective groups of instructions. *See* note 7, *supra*.

which a thread executes an instruction that attempts or performs an interfering or conflicting access). Ex.1003, ¶¶101-105.

[1.3B] wherein said processing comprises rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set before allowing an instruction in said third group to access said cache line, and otherwise granting said access;

Moir and Martínez render obvious limitation [1.3B] because both Moir and Martínez describe what happens when another group of instructions (e.g., *said third group of instructions*) is executed by another processor and causes said cache line to be accessed in a manner that conflicts or interferes with other accesses such as by the *first* or *second* groups of instructions discussed above with reference to limitations [1.1] and [1.2]. Ex.1003, ¶¶106-119.

Specifically, Moir explains that “if an interfering data access is detected, the **system discards changes made** during the transactional execution...**and attempts to re-execute** the critical section.” Ex.1005, 8:9-12 (explaining relative to Fig.3). Likewise, though with greater detail, Martínez explains that “hardware looks for conflicting accesses—accesses from two threads to the same location that include at least one write.... If two conflicting accesses cause a dependency violation, the hardware **rolls the offending speculative thread back** to the synchronization point **and restarts it on the fly.**” Ex.1006, 127; Ex.1003, ¶107.

Given the mapping of first and second groups of instructions to Moir’s

processor 101, processor 102 constitutes another processor (an agent other than said processor, *see* [1.3A]) that executes a third group of instructions that initiate the interfering data access. *See* Ex.1005, Fig. 1; Ex.1003, ¶108. The analysis applies equally if the mapping to processors 101 and 102 is reversed. *See* Ex.1005, 6:11-12 (“Processor 102 is similar in structure to processor 101”). Likewise, POSITA would understand a processor corresponding to any processor in the cache-coherent multiprocessors contemplated by Martínez to be illustrative of another processor that initiates the conflicting access. *See* Ex.1006, 133; Ex.1003, ¶108.

Because Moir explicitly describes multiple load marking bits associated with a cache line, *e.g.*, word-granular load marking within the cache line, POSITA would have understood Moir to teach that its discard and re-execute processing relative to a conflicting access to a cache line *comprises* both (i) *rolling back said first group of instructions provided said first bit is set* and (ii) *rolling back said second group of instructions provided said second bit is set*. Ex.1003, ¶109; *see supra*, limitations [1.1] and [1.2] (explaining first and second bits set based on first and second groups of instruction that each include a monitored load to respective words within the same cache line).

Martínez, on the other hand, more completely describes the interaction of a conflicting access (*e.g.*, by an instruction in *said third group of instructions*) with a speculatively executed *first or second group of instructions*. Specifically, Martínez

explains that “conflicts manifest as a thread receiving an external invalidation to a cached line...[and i]f a speculative thread receives an external message for a line marked speculative, the SSU at the receiving node squashes the local thread.” Ex.1006, 130. “Once the SSU triggers a squash, it...gang-invalidates all dirty cache lines with the Speculative bit set...and...forces the processor to restore its checkpointed register state, which results in the thread’s rapid rollback to the acquire point.” Ex.1006, 130. Moreover, “[i]f an external read to a dirty speculative line in the cache triggered the squash, the node replies without supplying any data.” Ex.1006, 130. Indeed, Martínez teaches that its “coherence protocol then regards the state for that cache line as stale and supplies a clean copy from memory” in a manner similar to conventional MESI (modified-exclusive-shared-invalid) protocols. Ex.1006, 130. Thus, Martínez teaches rollback *before allowing an instruction in said third group to access said cache line*. Ex.1003, ¶¶110-111 (explaining that “without supplying any data” means *before allowing the conflicting instruction to access the cache line*).

Both Moir and Martínez rely on cache line marking and cache coherence circuitry to manage speculative execution of individual threads of computation executing in parallel on respective processors and ultimately roll-back an interfered with thread. Ex.1005, 7:65-67, 8:1-12; Ex.1006, 129-30; Ex.1003, ¶112. Where Moir and Martínez differ is in the aggressiveness of roll-back strategy. Moir does

not explicitly require a particular timing of roll back. *See* Ex.1005, 8:1-12, Fig. 3; Ex.1003, ¶112.

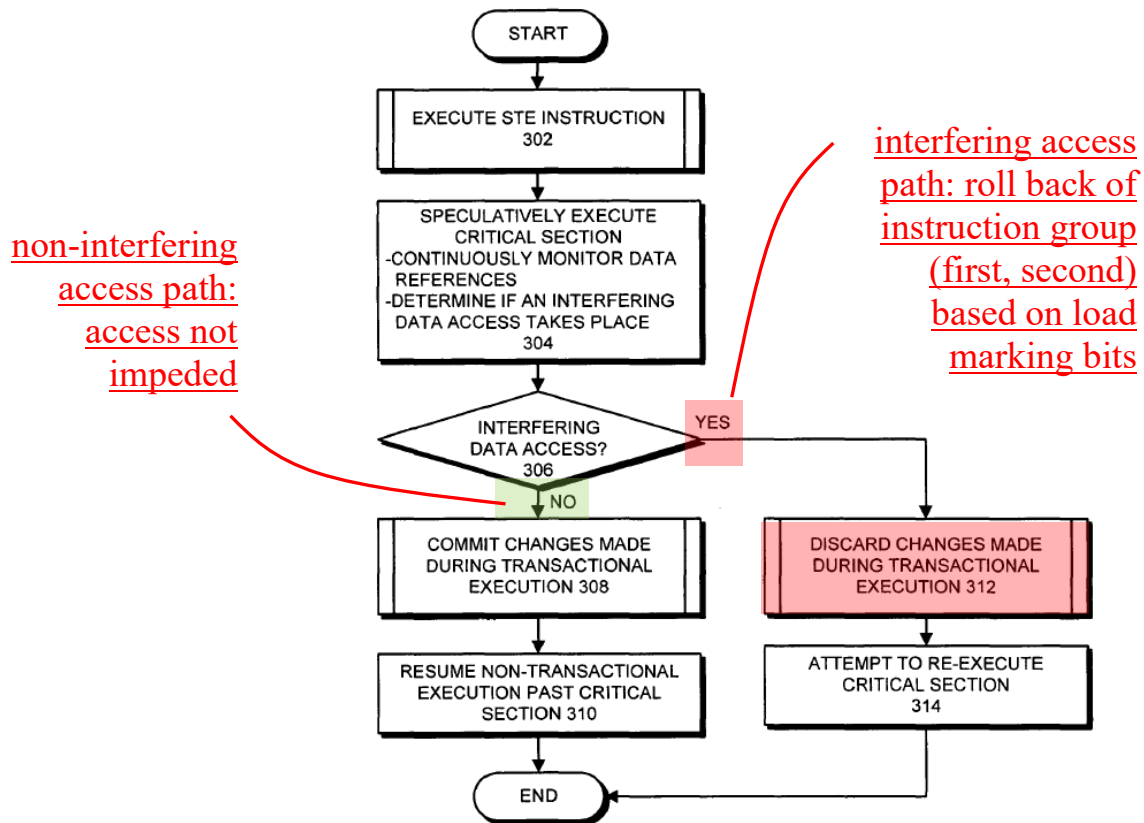


FIG. 3

Ex.1005, Fig. 3 (annotated); Ex.1003, ¶112

Because Moir is silent as to the status of an interfering third group of instruction's access to a cache-line for which first and/or second groups of instructions have marked respective first and second word-granular load marking bits, POSITA would reasonably understand the "interfering data access?" check 306 of Fig. 3 to be detection of the **attempt** to access a memory location within a marked

cached line. Ex.1003, ¶¶113-114 (noting consistency with conventional cache line invalidation handling, coherence protocols and snooping and explaining that continued execution would be wasted effort).

Moreover, Martínez more explicitly teaches an interaction of the conflicting access operation (*e.g.*, of *instructions in said third group*) with speculatively executed *first* or *second groups of instructions* such that the interfering access attempt stalls and such that rollback and retry of an interfered-with thread has immediate effect *before allowing* the conflicting operation to actually *access* valid data for the addressable memory location(s) associated with *the cache line*. Ex.1003, ¶¶110-111 (citing Ex.1006, 130, and explaining that “without supplying any data” means *before allowing* the conflicting instruction to *access the cache line*), ¶115. POSITA would have been motivated to perform rollback *before allowing* the conflict triggering instruction to *access said cache line* (as taught by Martínez) in the context of Moir’s system because Moir’s discard and re-execution (rollback) of instructions for the interfered with threads or critical sections would be performed more promptly without the wasted execution cycles for additional instructions to reach commit points of the interfered-with threads or critical sections and only then roll back. Ex.1003, ¶115 (explaining that any further operations by the critical section of a first or second thread do not advance the overall computational state, as their results would also need to be rolled back if executed, post-interference).

Benefits of more promptly rolling back before allowing access therefore include fewer wasted compute cycles the ability to more quickly retry/reissue instructions once rolled back.

Finally, POSITA would have understood that where accesses are non-interfering or non-conflicting (*e.g.*, because they do not exhibit a dependency hazard or because they are not executed with transactional memory or speculative execution controls that result in load- or store-marking)—whether in Moir, Martínez, or the combination—access is not impeded. Accordingly, Moir in view of Martínez *otherwise grants said access*. Ex.1003, ¶116.

Thus, Moir in view of Martínez renders obvious *wherein said processing comprises rolling back* (roll back and restart as taught by Martínez) *said first group of instructions provided said first bit is set* (the first word-granular load marking bit associated with a first processor’s cache line as per [1.1]) *and rolling back* (roll back and restart as taught by Martínez) *said second group of instructions provided said second bit is set* (the second word-granular load marking bit associated with the same processor’s same cache line as per [1.2]) *before allowing an instruction in said third group to access said cache line* (as taught by Martínez), *and otherwise* (in the non-interfering case) *granting said access*. Ex.1003, ¶¶106-119.

[1.3B-alternate] Note Regarding Possible Alternative Constructions

The forgoing analysis treats the claim language “*wherein said processing*

comprises rolling back said first group...and rolling back said second group...” as finding antecedent basis in limitation [1.4] (“**processing** *said first group and said second group of instructions according to [cache line marking]*”). This interpretation¹¹ (in which **said processing** is performed by **said processor**, rather than by “*an agent other than said processor*”) is consistent with the scope of originally presented claim 35 for which the Examiner indicated allowability and which forms the basis for the re-written claim that ultimately issued as claim 1. *See* Ex.1002, 241-242. As explained in Section V.A.2 above, application claims 33, 34 and 35 were combined as the claim that issued as claim 1. Ex.1002, 32.

Accordingly, the foregoing analysis explains how the Moir-Martínez combination’s handling of first and second groups of instructions provides roll back(s). Nonetheless, Patent Owner has sought to avoid a finding of indefiniteness in other proceedings by arguing that the plain and ordinary meaning of “*said processing*” is somehow tethered instead to [1.3A] “*executing a **third group of instructions** that causes said cache line to be accessed, wherein said third group of instructions is **executed by an agent other than said processor***” (Ex.1026, 19-23 (briefing opposing indefiniteness in district court). Even under such an argument (if

¹¹ Petitioner specifically reserves its right to assert invalidity under 35 U.S.C. §112 in related proceedings before the district court.

advanced here), the ground presented herein renders the claim obvious. Petitioner’s ground encompasses the possibility that “*rolling back*” is initiated or triggered by “*execut[ion of the] third group of instructions...by an agent other than said processor.*” Petitioner and its expert have explained that roll back is triggered by a conflicting instruction of the *third group* executed by another processor that seeks to *access said cache line*. Therefore, the Moir-Martínez ground embraces “*said processing comprising rolling back said first group [if first bit set] and rolling back second group [if second bit set],*” whether (a) the processor/agent executing the third group initiates or triggers the roll back(s); (b) the processor executing the first group and second group performs respective roll back(s); or both (a) and (b). Ex.1003, ¶121.

To date, Patent Owner appears to avoid taking a clear position on which processor/agent performs rollback by simply alleging plain and ordinary meaning in district court. However, that does not resolve the question. Petitioner has advanced a properly predicated construction that addresses the current dispute (*see, supra*, Section V.A.2), but reserves the right to more completely respond to an alternative construction of “*said processing*” if subsequently advanced by Patent Owner in this proceeding.

6. Claim 2

[2.1] *The method of claim 1 wherein said processing further comprises: determining a state of said cache line, wherein said state is specified according to a cache coherency protocol comprising at least a modified state, a shared state, and an invalid state; and*

As discussed above, the combination of Moir and Martínez renders obvious *the method of claim 1*. Additionally, the Moir-Martínez combination renders obvious limitation [2.1] because each reference describes or incorporates cache coherence mechanisms that POSITA would have understood to *determine state of a cache line* via snooping techniques and using *states specified according* to then-conventional MESI/MOESI (*modified*, [*owned*], *exclusive*, *shared*, *invalid*) *cache coherency protocols*. Ex.1003, ¶¶122-125.

For example, Moir explains that its “L2 cache 120 operates in concert with L1 data cache 115...in processor 101...and with L1 data cache 117...in processor 102” and employs “a coherency mechanism 122” such as described in a co-pending application (published as Ex.1011) naming two of his co-inventors. Ex.1005, 6:45-59 (specifically citing Ex.1011 for supporting directory structure). POSITA would have understood the referenced coherency mechanism to involve *cache coherency protocols* amongst the various caches and, indeed, the referenced co-pending application indicates that its “**cache coherency protocol** includes all of the usual

state transitions between the following MOESI **states: modified (M)**, owned (O), exclusive (E), **shared (S)** and **invalid (I)** states.”; *see also* Ex.1011, [0139].

Further, Moir explains that its “coherency mechanism 122 maintains ‘copyback information’ 121 for each cache line. This copyback information 121 facilitates sending a cache line from L2 cache 120 to a requesting processor in cases where a cache line must be sent to another processor.” Ex.1005, 6:55-59. In this way, separate processors (*i.e.*, a processor and an agent other than the processor) are capable of accessing the same cache lines, and a coherency mechanism is used to determine “whether any interfering memory references take place during transactional execution,” as is discussed throughout Moir. Ex.1005, 6:60-65; Ex.1003, ¶124.

Thus, Martínez and/or Moir disclose (or render obvious to POSITA) both the use of *states* that are *specified according to a cache coherency protocol comprising at least a modified state, a shared state, and an invalid state* and the *determining of a state of said cache line* using such protocols. Ex.1003, ¶¶122-125.

[2.2] rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set, and then granting said access provided said cache line is in other than said shared state, and otherwise granting said access only when said cache line is in said shared state.

The combination of Moir and Martínez renders obvious limitation [2.2] because, as discussed in limitation [1.3B] and [1.3B-alternative], Moir in view of

Martínez renders obvious rolling back respective first and second groups of instructions under conditions where respective first and second bits are set before allowing the referenced access. And consistent with the discussion of limitation [1.3B], POSITA would have further understood that, after rolling back any affected instruction groups, said access is *then granted*. Ex.1003, ¶126 (rollback **before** access also means rollback if applicable, *then* access). Accordingly, Moir in view of Martínez renders obvious *rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set, and then granting said access*. Ex.1003, ¶¶126-130.

As reflected in Figure 4 of the '395 patent, below, POSITA would have understood that in line with the claim language, a first group of instructions and/or a second group of instructions are rolled back provided the cache line is in a state other than a shared state, such as a modified or invalid state, while no rollback would occur if the cache line is in a shared state. Ex.1001, 7:53-62; Ex.1003, ¶127 (MESI dictates a transition from shared to modified or exclusive on an access that could interfere). Thus, POSITA would have understood that, in cases where *said cache line is in said shared state*, the “*otherwise*” condition of *granting said access* simply based on *said shared state* applies, and in cases where *said cache line is in other than said shared state*, the previously explained dependence of roll back on observed bits applies. Ex.1003, ¶127.

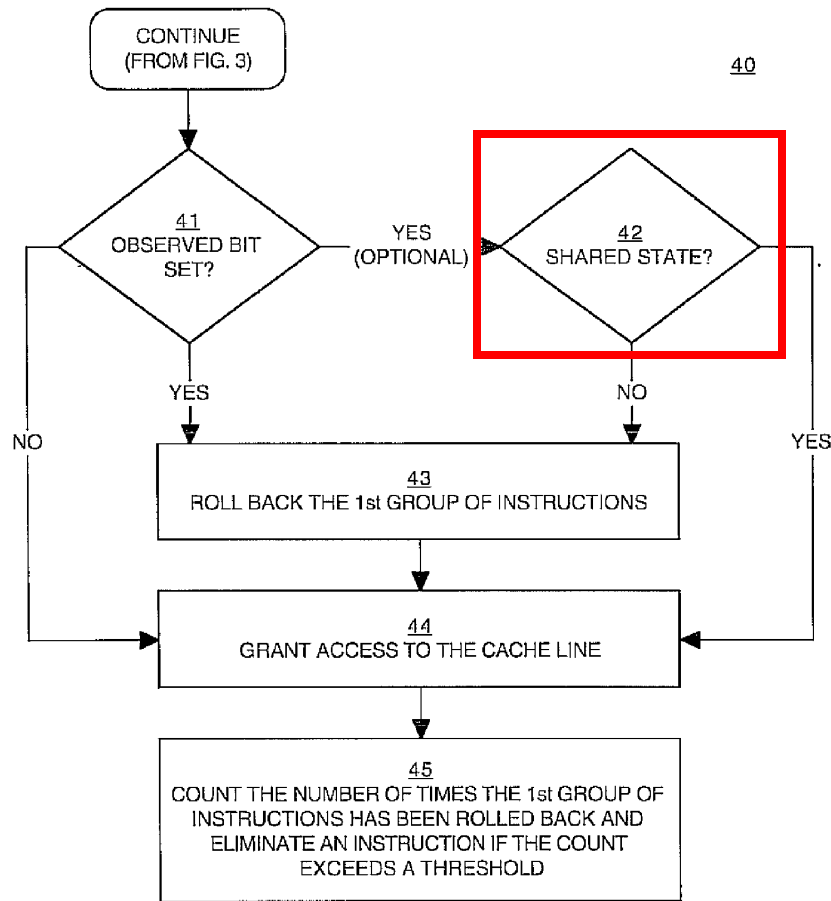


Figure 4

Ex.1001, Fig. 4 (annotated); Ex.1003, ¶127

Regarding the applied Moir-Martínez combination and execution sequence dependence on whether *said cache line is in said shared state*, POSITA would have understood the claimed dependence to follow from a conventional understanding of the MESI coherency protocol and its *shared* (S) state. Ex.1003, ¶128 (**shared state** indicates that the cache line may be stored in other caches of the machine and is clean—that is, its contents match main memory and it is not **modified** or **owned** or

exclusive to particular processor or cache—and thus does not present access conflict issues); Ex.1003, ¶¶128-129 (citing Ex.1019, explaining MESI/MOESI protocols). For an implementation that tracks cache line state in accord with MESI/MOESI coherency protocols (such as Moir and/or Martínez), the states other than the shared state, *e.g.*, **modified**/dirty (M) or **owned** (O), if provided, or **exclusive** (E) state(s), would have been the appropriate states for which access conflicts would be managed in the manner discussed in limitation [1.3B] and reprised in the language of limitation [2.2]. Ex.1003, ¶¶128-129.

Thus, Moir and Martínez render obvious *rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set, and then granting said access provided said cache line is in other than said shared state, (rolling back if interfering access is detected through load-marks as described relative to [1.3B], [1.3B-alternative]) and otherwise granting said access only when said cache line is in said shared state (skipping checks for interfering access for cache lines marked as shared)*. Ex.1003, ¶¶126-130.

7. Claim 5

[5.1] *The method of claim 1 further comprising clearing said first bit when said execution of said first group of instructions is ended, wherein said second bit remains set until execution of said second group of instructions is ended.*

As discussed in Section VIII.C.5 above, the combination of Moir and Martínez renders obvious *the method of claim 1*. Additionally, Moir renders obvious claim 5 because Moir explains that “the system clears load-marks from [the] L1 data cache” after a transaction execution completes. Ex.1005, 9:49-52, 10:3-4, 10:50-53, 10:60-64, Figs. 7, 8; Ex.1003, ¶¶131-135.

As discussed above in limitation [1.4], Moir explains that “[d]uring this transactional execution, the system continually monitors data references made by other threads, and determines if an interfering data access...takes place during transactional execution. **If not, the system atomically commits** all changes made during transactional execution (step 308)[.]” Ex.1005, 8:1-8, *see also* Fig. 3. As part of the process of committing changes made during the transaction execution (step 308 in Fig. 3), “the system clears load-marks from [the] L1 data cache” (step 705 in Fig. 7). Ex.1005, 9:52-53, 10:3-4; Ex.1003, ¶132.

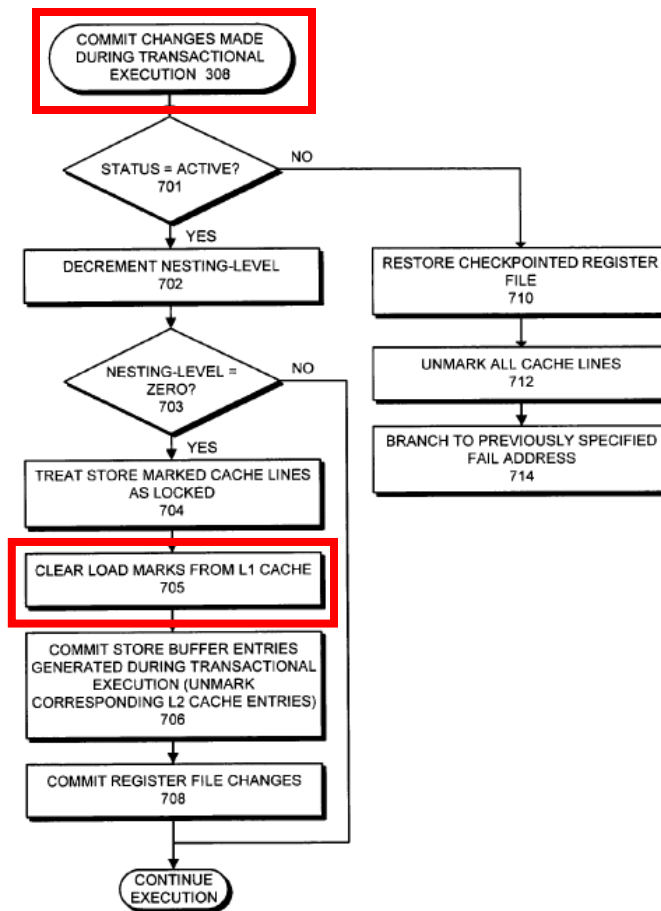


FIG. 7

Ex.1005, Fig. 7 (annotated); Ex.1003, ¶132

Similarly, as reflected in Figure 8, below, Moir discloses that “[t]he system also clears load-marks from cache lines in [the] L1 data cache” when changes are discarded (i.e., rolled back). Ex.1005, 10:60-64. This occurs if there is an interfering data access (or other type of failure), such that “the system discards changes made during the transactional execution (step 312).” Ex.1005, 8:10-12; Ex.1003, ¶133. As part of the process of discarding changes made during the transaction execution (step

312 in Fig. 3), “the system also clears load-marks from [the] L1 data cache” (step 805 in Fig. 7). Ex.1005, 10:53-54, 10:60-64; Ex.1003, ¶133.

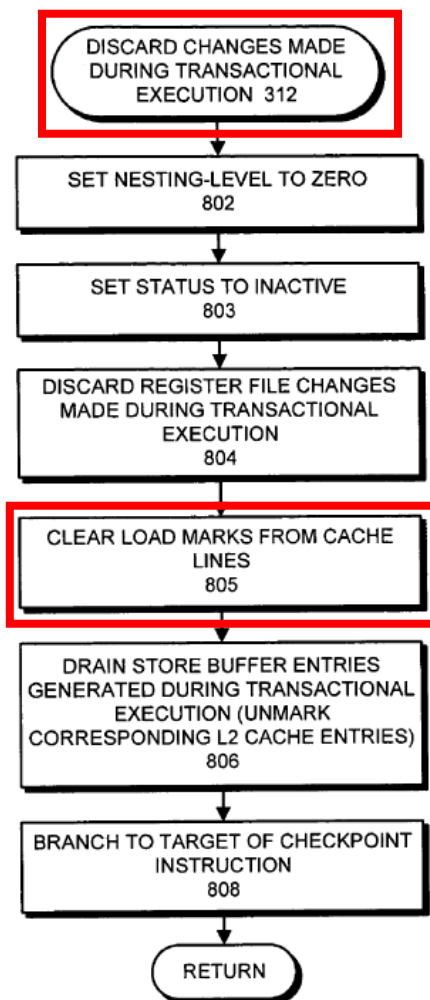


FIG. 8

Ex.1005, Fig. 8 (annotated); Ex.1003, ¶133

Moreover, Moir clears load-marks from the L1 data cache that are used to mark respective word-granular bits associated with a given cache line. As discussed above in limitations [1.1] and [1.2], Moir allows multiple critical sections to separately mark addressable words within a cache line. Accordingly, POSITA would

have understood that, following either the successful or unsuccessful completion of a transactional execution for a group of instructions, the load-marks for the corresponding words within a cache line would be cleared, *clearing said first bit when said execution of said first group of instructions is ended*, because the processor would have proceeded through steps following completion that include clearing load-marks from the L1 cache. Ex.1005, 9:49-52, 10:3-4, 10:50-53, 10:60-64, Figs. 7, 8; Ex.1003, ¶134. However, for other groups of instructions for which the transactional execution has not yet completed, *said second bit remains set until execution of said second group of instructions is ended* because changes have neither been committed nor discarded, and thus the processor would not yet have cleared the load-marks corresponding to those instructions. Ex.1003, ¶134.

Thus, Moir's disclosure of clearing load-marks from the L1 data cache after a transactional execution completes renders obvious *clearing said first bit when said execution of said first group of instructions is ended, wherein said second bit remains set until execution of said second group of instructions is ended*. Ex.1003, ¶¶131-135.

8. Claim 7

Claim 7 is a system claim and is obvious over Moir and Martínez for the reasons detailed in the corresponding limitations of corresponding method claim 1.

The limitations “*a processor*” and “*cache memory for use by said processor*” are disclosed by Moir as explained in [1.0]. Ex.1003, ¶136.

Moreover, the additionally recited “*memory unit coupled to said processor and having stored therein instructions, said instructions comprising: instructions to [perform each of the steps recited in the corresponding elements of claim 1]*” is disclosed by Moir in the form of an L1 instruction cache (“*a memory under coupled to said processor*”). Ex.1005, 6:35-36 (explaining that “[p]rocessor 101 also includes an L1 instruction cache (not shown [in Fig. 1])”). POSITA would have understood that the L1 instruction cache stores instructions executed by Moir’s processor (*i.e.*, “*instructions to execute...set...execute...set...execute...roll back [and] process...*” as recited in the respective limitations of claim 7). Ex.1003, ¶137 (explaining instruction cache). Because each of those limitations of claim 7 recites a function that corresponds to limitation [1.1], [1.2], [1.4], [1.3A] or [1.3B] of corresponding method claim 1, and because POSITA would have understood Moir’s processor to perform those functions based on instructions executed from its instruction cache, claim 7 is obvious over Moir and Martínez for analogous reasons to those detailed above with respect to claim 1. Ex.1003, ¶137. Moreover, Martínez’ safe and speculative threads also include (and render obvious in the combination) the recited “*instructions to...*” perform recited functions. *Id.*

9. Claim 8

Claim 8 depends from claim 7 and is obvious over Moir and Martínez for the reasons detailed above, including those detailed for corresponding limitations [2.1] and [2.2] of corresponding method claim 2. Each of these limitations of claim 8 recites a function that corresponds to a limitation of corresponding method claim 2 that POSITA would have understood Moir's processor to perform based on execution of instructions executed from its L1 instruction cache. Ex.1003, ¶138.

10. Claim 11

Claim 11 depends from claim 7 and is obvious over Moir and Martínez for the reasons detailed above, including those detailed for limitation [5.1] of corresponding method claim 5. Claim 11 recites the same *clearing* function as method claim 5, and POSITA would have understood Moir's processor to “clear” the recited bit associated with the cache line as explained relative to [5.1] based on execution of instructions executed from L1 instruction cache of Moir's processor. Ex.1003, ¶139.

IX. MARTÍNEZ IS A PRINTED PUBLICATION

Petitioner submits the declaration of Gordon MacPherson (Ex.1008), along with Internet Archive captures of the IEEE webpage (Ex.1009 and Ex.1010) and testimony from Dr. Colwell to show that Martínez was publicly accessible to POSITAs more than a year before the earliest priority date of the '395 patent (April

7, 2005).

Martínez was publicly accessible by January 21, 2004, because Martínez was published in the Nov-Dec 2003 issue of IEEE Micro (a peer-reviewed journal), and subsequently published on-line IEEE Xplore on January 21, 2004. Ex.1008, 13. Mr. MacPherson explains that “[t]he article and abstract from IEEE Xplore show the date of publication.” Ex.1008, ¶10. Internet Archive Wayback Machine captures of the IEEE webpages for the IEEE Micro publication confirm that Martínez was publicly available for IEEE subscribers and for purchase following its publication date. *See* Ex.1009 (February 1, 2004 Wayback Machine capture of the IEEE Micro webpage showing the November/December 2003 issue); Ex.1010 (February 14, 2004 Wayback Machine capture of the table of contents of the November/December 2003 issue, with links to view or purchase articles, including Martínez); *Valve Corp. v. Ironburg Inventions Ltd.*, 8 F.4th 1364, 1374 (Fed. Cir. 2021) (Internet Archive’s Wayback Machine presents “facts that can be accurately and readily determined from sources whose accuracy cannot reasonably be questioned.”). As Dr. Colwell explains, notwithstanding the IEEE paywall, a substantial portion of POSITAs would have been IEEE and/or IEEE computer society members at the time and would have had IEEE digital library access to IEEE publications such as Martínez and the other publications appearing in the November/December 2003 issue (Vol. 23, No. 6) as well as profession interest therein. Ex.1003, ¶¶57-60. As an IEEE

Member—and IEEE Fellow—Dr. Colwell himself would have accessed and reviewed IEEE publications and has authored numerous articles appearing in IEEE publications and journals. Ex.1003, ¶¶57-60. Accordingly, Martínez is a printed publication.

X. DISCRETIONARY DENIAL IS INAPPROPRIATE

A. No Basis For §325(d) Denial

The Board should not deny this Petition under §325(d) because the Examiner did not consider Moir or Martínez, or the combination of references as presented in this Petition.

B. No Basis for *Fintiv* Denial

The Board balances six factors in considering denial under §314. *Apple Inc. v. Fintiv, Inc.*, IPR2020-00019, Paper 11 (Mar. 20, 2020) (precedential). These factors strongly favor institution. The district court case is in its early stages. Patent Owner filed its complaint on April 12, 2024 (Ex.1014), and Petitioner has worked diligently to prepare this Petition, which is being filed within the one-year timeframe allowed by Congress.

1. No evidence regarding a stay

No motion to stay has been filed, so the Board should not infer the outcome of such a motion. This factor is neutral.

2. Parallel proceeding trial date

This factor weighs against discretionary denial because the projected trial date—based on median time-to-trial statistics—is in January of 2027¹², which is after the projected date for the Board to issue a Final Written Decision in September 2026. While trial is currently proposed¹³ for May 14, 2026 (Ex.1015, 4), Petitioner notes that “a court’s general ability to set a fast-paced schedule is not particularly relevant...where, like here, the forum itself has not historically resolved cases so quickly.” *In re Apple Inc.*, 979 F.3d 1332, 1344 (Fed. Cir. 2020). The co-pending litigation involves 12 asserted patents, which may cause further delays in the trial schedule or asserted patents to be withdrawn. Even if the proposed trial date occurs approximately four months before the Board would issue a final written decision, the Board has found that “the efficiency and fairness concerns that underlie the *Fintiv* analysis are not as strong.” *Zynga Inc. v. IGT*, IPR2022-00199, Paper 11, 14 (June 14, 2022) (no discretionary denial when four-month earlier scheduled trial date was “at or around the same time” as projected IPR final written decision date).

¹² The current average time-to-trial for the district court hearing the co-pending litigation is 33.9 months. *See* Ex.1016, 37.

¹³ The district court has not entered an order scheduling a trial date.

3. Investment in the parallel proceeding

The investment in the co-pending litigation has been minimal. A claim construction hearing has not yet occurred, fact discovery is not proposed to open until April 11, 2025, and expert discovery is not proposed to open until November 20, 2025. Ex.1015, 2-3; *see PEAG LLC v. Varta Microbattery GmbH*, IPR2020-01214, Paper 8 at 17 (Jan. 6, 2021). This lack of investment favors institution.

Moreover, Petitioner only learned which claims were being asserted in August 2024. *See* Ex.1018 (infringement contentions cover pleading). Since then, Petitioner has worked expeditiously to file this petition. As of this filing, Patent Owner has not yet served its final infringement contentions. Under *Fintiv*, Petitioner’s prompt filing “weigh[s] against exercising the authority to deny institution.” *Fintiv*, Paper 11 at 11.

4. Overlapping issues with the parallel proceeding

The co-pending litigation is in its early stages. Petitioner served its preliminary invalidity contentions on October 24, 2024. However, Petitioner will not be serving its final invalidity contentions until May 22, 2025. Ex.1015, 3. The extent of overlap is thus speculative. Moreover, Petitioner is challenging more claims here (claims 7, 8, and 11) than those asserted in the co-pending litigation (claims 1, 2, and 5), which minimizes overlap between the proceedings.

5. Petitioner is a defendant

Petitioner is a defendant in the co-pending litigation. That is true of most Petitioners in IPR proceedings, making this factor neutral. *See HP Inc. v. Slingshot Printing LLC*, IPR2020-01084, Paper 13 at 9 (Jan. 14, 2021).

6. Other circumstances

Here, the petition—along with Dr. Colwell’s expert testimony—plainly shows that the ’395 patent claims no more than well-known subject matter. This compelling evidence of unpatentability weighs against discretionary denial.

C. No Basis for *General Plastic* Denial Under §314(a).

The ’395 patent has not been challenged in any prior IPR petition, so *General Plastic* discretionary institution does not apply to this Petition. *General Plastic*, IPR2016-01357, Paper 19, 16 (Sept. 6, 2016) (precedential).

XI. CONCLUSION

The Challenged Claims are unpatentable.

Respectfully submitted,

Dated: March 7, 2025
HAYNES AND BOONE, LLP
2801 N. Harwood St., Suite 2300
Dallas, Texas 75201

/ Jonathan R. Bowser /
Jonathan R. Bowser
Lead Counsel for Petitioner
Registration No. 54,574

XII. MANDATORY NOTICES UNDER 37 C.F.R. §42.8**A. Real Party-in-Interest**

Pursuant to 37 C.F.R. §42.8(b)(1), Petitioner Tesla Inc. certifies that it is the real party-in-interest in this proceeding.

B. Related Matters

According to assignment records, the '395 patent is currently assigned to Intellectual Ventures II LLC ("Patent Owner").

As of the filing date of this Petition, and to the best knowledge of Petitioner, the '395 patent is or has been involved in:

Case Heading	Number	Court	Filed
<i>Intellectual Ventures II LLC v. Tesla, Inc</i>	<i>WDTX-1-24-cv-00390</i> (terminated)	WDTX	Apr. 11, 2024
<i>Intellectual Ventures II LLC v. Tesla, Inc.</i>	<i>WDTX-6-24-cv-00188</i> (pending)	WDTX	Apr. 12, 2024
<i>Intellectual Ventures II LLC v. Tesla, Inc.</i>	<i>WDTX-1-24-cv-00884</i> (transferred)	WDTX	Apr. 12, 2024

C. Lead and Back-up Counsel and Service Information

<u>Lead Counsel</u> Jonathan R. Bowser HAYNES AND BOONE, LLP 2801 N. Harwood Street, Suite 2300 Dallas, TX 75201	Phone: 202-654-4503 Fax: 214-200-0853 jon.bowser.ipr@haynesboone.com USPTO Reg. No. 54,574
<u>Backup Counsel</u> David O'Brien HAYNES AND BOONE, LLP 2801 N. Harwood Street, Suite 2300 Dallas, TX 75201 Scott Jarratt HAYNES AND BOONE, LLP 2801 N. Harwood Street, Suite 2300 Dallas, TX 75201 Matthew Beck Haynes and Boone, LLP 2801 N. Harwood Street, Suite 2300 Dallas, TX 75201 Ashraf Fawzy TESLA, INC. 800 Connecticut Ave. NW Washington, DC 20006	Phone: 512-867-8457 Fax: 214-200-0853 david.obrien.ipr@haynesboone.com USPTO Reg. No. 40,107 Phone: 972-739-8663 Fax: 214-200-0853 scott.jarratt.ipr@haynesboone.com USPTO Reg. No. 70,297 Phone: (202) 654-4573 Fax: (214) 200-0853 matthew.beck.ipr@haynesboone.com USPTO Reg. No. 79,791 Phone: (202) 905-9221 afawzy@tesla.com USPTO Reg. No. 67,914

Please address all correspondence to lead and back-up counsel. Petitioner consents to electronic service and asks Patent Owner to do the same.

XIII. CERTIFICATE OF WORD COUNT

Under 37 C.F.R. §42.24, the undersigned attorney for the Petitioner, declares that the argument section of this Petition (§§I-XI) has 13,971 words, according to the word count tool in Microsoft Word™.

/Jonathan R. Bowser/
Jonathan R. Bowser
Counsel for Petitioner
Reg. No. 54,574

