

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SAP AMERICA, INC.,
Petitioner,

v.

VALTRUS INNOVATIONS LTD. and KEY PATENT INNOVATIONS LTD.
Patent Owners.

IPR2025-00418
U.S. Patent No. 6,871,264

**PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 6,871,264**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	10
II. COMPLIANCE WITH IPR REQUIREMENTS.....	10
A. Grounds for Standing	10
B. Mandatory Notices	10
C. Fee Authorization	13
III. CHALLENGED CLAIMS	13
IV. THE '264 PATENT	13
A. Earliest Effective Filing Date	13
B. Level of Ordinary Skill in the Art	14
C. Overview of the '264 Patent.....	14
V. CLAIM CONSTRUCTION	17
VI. PRINCIPAL PRIOR ART	17
A. Summary of Cherabuddi	17
B. Summary of Dean.....	21
C. Summary of Noel	22
VII. CLAIMS 1-5 AND 8-11 ARE UNPATENTABLE.....	23
A. Ground 1: Claims 1-3 Would Have Been Obvious Based on Cherabuddi	23
1. Claim 1	23
2. Claim 2	43
3. Claim 3	46
B. Ground 2: Claims 4-5 and 8-10 Are Obvious Based on Cherabuddi and Dean	48
1. Motivation to Combine	48
2. Claim 4	53
3. Claim 5	56
4. Claim 8	74

5.	Claim 9.....	75
6.	Claim 10.....	78
C.	Ground 3: Claim 11 Based on Cherabuddi and Noel.....	80
1.	Claim 11	80
VIII.	DISCRETIONARY DENIAL IS NOT APPROPRIATE	87
A.	The Board Should Not Deny Institution Under 35 U.S.C. § 325(d).....	87
B.	The Board Should Not Deny Institution Under 35 U.S.C. § 314(a).....	88
IX.	CONCLUSION.....	91

TABLE OF ABBREVIATIONS AND CONVENTIONS

Petitioner	SAP America, Inc.
Patent Owners	Valtrus Innovations Ltd. (“Valtrus”) and Key Patent innovations Ltd. (“KPI”)
’264 Patent	U.S. Patent No. 6,871,264 (the patent being challenged herein)
Challenged Claims	Claims 1-5 and 8-11
POSITA	Person of Ordinary Skill in the Art

CLAIM LISTING

Claim 1	
1.0	A processor integrated circuit capable of executing more than one instruction stream comprising:
1.1	a first processor, coupled to fetch instructions and access data through a first cache controller;
1.2	a second processor, coupled to fetch instructions and access data through a Second cache controller;
1.3	a plurality of cache memory blocks;
1.4	a high-speed interconnect coupling the plurality of cache memory blocks to the first and second cache controllers such that at least one allocable cache memory block is capable of being used by the first and second cache controllers, and
1.5	a resource allocation controller coupled to determine an accessing cache memory controller selected from the group consisting of the first and second cache memory controllers, whereby the accessing cache memory controller is allowed to access the allocable cache memory block,
1.6	wherein the cache memory blocks are usable by the cache controllers to store data and instructions fetched from a random-access memory.
Claim 2	
2.0	The processor integrated circuit of claim 1, further comprising a plurality of first level cache systems, wherein the first processor fetches instructions and accesses data from the first cache controller through a first first level cache system, and wherein the second processor fetches instructions and accesses data from the second cache controller through a second first level cache system.

Claim 3	
3.0	The processor integrated circuit of claim 1, wherein the cache memory blocks further comprise cache tag memory.
Claim 4	
4.0	The processor integrated circuit of claim 1, wherein each cache controller is provided with cache hit rate monitoring apparatus.
Claim 5	
5.0	A method of dynamically allocating cache on a multiple-processor integrated circuit, where the multiple processor integrated circuit is used in a partitionable multiple-processor system and comprises:
5.1	a plurality of processors each coupled to receive instructions from a first level cache associated therewith,
5.2	a plurality of allocable upper level cache memory blocks, interconnect apparatus for transmitting cache misses at each first level cache to upper level cache memory blocks assigned thereto, and
5.3	allocation apparatus for assigning upper level cache memory blocks to processors;
5.4	the method comprising the steps of: monitoring past cache performance associated with processors and partitions;
5.5	determining desired processor to partition and upper level cache block allocations to processors; and
5.6	repartitioning the system, the step of repartitioning the system including allocation of upper level cache blocks to processors of at least one of the multiple processor integrated circuits,
5.7	wherein the upper level cache blocks are usable to store data and instructions fetched from a random-access main memory.

Claim 8	
8.0	The method of claim 5, wherein the multiple processor integrated circuit further comprises a plurality of nonallocable cache memory blocks.
Claim 9	
9.0	The method of claim 5, wherein the interconnect apparatus further comprises a plurality of upper level cache controllers, and where each upper level cache controller is capable of controlling operation of the allocable cache memory blocks as a writeback cache.
Claim 10	
10.0	The method of claim 5, wherein each upper level allocable cache block further comprises tag memory and cache data memory.
Claim 11	
11.0	The method of claim 10, further comprising the steps of stopping execution of operating systems in each partition, and restarting execution of operating systems in each partition, and wherein the system is capable of being repartitioned without rebooting each operating system.

PETITIONER’S EXHIBIT LIST

No.	Description
1001	U.S. Patent No. 6,871,264
1002	U.S. Patent No. 6,725,336 (“Cherabuddi”)
1003	U.S. Patent No. 6,604,174 (“Dean”)
1004	U.S. Patent No. 6,381,682 (“Noel”)
1005	Declaration of Dr. Bruce Jacob
1006	<i>Curriculum Vitae</i> of Dr. Bruce Jacob
1007	File History of U.S. Patent No. 6,871,264
1008	Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation by Director Katherine K. Vidal
1009	First Amended Docket Control Order entered in <i>Valtrus Innovations Ltd. v. SAP Am. Inc.</i> , No. 2:24-cv-21, ECF 65 (E.D. Tex. Jan. 15, 2024)
1010	SAP America, Inc. and SAP SE’s Motion to Dismiss for Lack of Standing Pursuant to Rule 12(b)(1) and Rule 12(b)(6)
1011	SAP America, Inc. and SAP SE’s Motion to Reconsider Partial Denial of SAP’s Motion to Dismiss and Grant of Valtrus’s Motion for Leave to Amend Complaint (ECF 83)
1012	Valtrus and KPI’s Complaint for Patent Infringement and Jury Demand, filed in <i>Valtrus Innovations Ltd. v. SAP Am, Inc.</i> , No. 2:24-cv-533, ECF 1 (E.D. Tex. July 12, 2024)
1013	Valtrus and KPI’s Statement Regarding <i>Valtrus II</i> (Case No. 2:24-CV-533-JRG), filed in <i>Valtrus Innovations Ltd. v. SAP Am, Inc.</i> , No. 2:24-cv-21, ECF 85 (E.D. Tex. Nov. 15, 2024)
1014	John L. Hennessy and David A. Patterson, COMPUTER ARCHITECTURE: A QUANTITATIVE APPROACH (5th ed. 2012)
1015	Bruce Jacob, Spencer W. Ng, and David T. Wang, MEMORY SYSTEMS: CACHE, DRAM, DISK (2008)

1016	Daniel J. Sorin, Mark D. Hill, David A. Wood, A PRIMER ON MEMORY CONSISTENCY AND CACHE COHERENCY (2011)
1017	Kunle Olukotun, et al., <u>The Case for a Single-Chip Multiprocessor</u> , STANFORD UNIVERSITY: COMPUTER SYSTEMS LABORATORY (1996)
1018	Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun, <u>A Single-Chip Multiprocessor</u> , STANFORD UNIVERSITY (Sept. 1997)
1019	Alan Jay Smith, <u>Cache Memories</u> , 14 COMPUTING SURVEYS 3 (Sept. 1982)

I. INTRODUCTION

Petitioner submits this Petition for *Inter Partes* Review of claims 1-5 and 8-11 of U.S. Patent No. 6,871,264 (Ex-1001), assigned to Patent Owners. Petitioner respectfully submits that the Challenged Claims of the '264 Patent are unpatentable under 35 U.S.C. § 103 in view of the prior art references discussed herein. Accordingly, it is respectfully requested that the Board institute *inter partes* review of the '264 Patent.

II. COMPLIANCE WITH IPR REQUIREMENTS

A. Grounds for Standing

As required by 37 C.F.R. § 42.104(a), Petitioner certifies that the '264 Patent “is available for inter partes review and that the Petitioner is not barred or estopped from requesting an inter partes review challenging the patent claims on the grounds identified in the petition.” Neither Petitioner nor any real party in interest has filed a civil action challenging the validity of a claim of the '264 Patent. This petition is being filed no more than one year after the date on which Petitioner or real parties in interest or privies of Petitioner were served with a complaint alleging infringement of the '264 Patent.

B. Mandatory Notices

Real Party-in-Interest: Petitioner identifies SAP America, Inc. and SAP SE (collectively, “SAP”) as real parties-in-interest.

Related Matters: No previous IPR or post-grant proceeding has been filed challenging the '264 Patent. Valtrus has asserted the '264 Patent in litigation against SAP in *Valtrus Innovations Ltd. v. SAP America, Inc. et al*, 2:24-cv-00021-JRG (E.D. Tex.); Patent Owners have asserted the '264 Patent in litigation against SAP in *Valtrus Innovations Ltd. v. SAP America, Inc. et al*, 2:24-cv-00533 (E.D. Tex.); and SAP filed a declaratory judgment action of non-infringement of the '264 Patent against Patent Owners and Patent Platform Services, LLC in *SAP America, Inc. v. Valtrus Innovations Limited et al*, 1:24-cv-00756-GBW (D. Del.).

Lead and Back-up Counsel:

Lead Counsel	Back-up Counsel
Louis L. Campbell (Reg. # 59,963) WINSTON & STRAWN LLP 255 Shoreline Drive, Suite 520 Redwood City, CA 94065 Tel: (650) 858-6500 LLCampbell@winston.com	Scott M. Border (Reg. # 77,744) WINSTON & STRAWN LLP 1901 L Street NW Washington, DC 20036 Tel: (202) 282-5000 SBorder@winston.com
	Samuel W. Riebe (Reg. # 78,044) WINSTON & STRAWN LLP 2121 N. Pearl Street, Suite 900 Dallas, TX 75201 Tel: (214) 453-6500 SRiebe@winston.com
	Michael Bittner (<i>Pro hac vice</i> to be submitted) WINSTON & STRAWN LLP 2121 N. Pearl Street, Suite 900 Dallas, TX 75201 Tel: (214) 453-6500 MBittner@winston.com

Lead Counsel	Back-up Counsel
	Matt R. McCullough (<i>Pro hac vice</i> to be submitted) WINSTON & STRAWN LLP <i>Pro hac vice</i> to be submitted Winston & Strawn LLP 255 Shoreline Drive, Suite 520 Redwood City, CA 94065 Tel: (650) 858-6500 MRMcCullough@winston.com
	William Logan (<i>Pro hac vice</i> to be submitted) <i>Pro hac vice</i> to be submitted WINSTON & STRAWN LLP 800 Capitol Street, Suite 2400 Houston, TX 77002 Tel: (713) 651-2600 WLogan@winston.com
	Patrick Clark (<i>Pro hac vice</i> to be submitted) WINSTON & STRAWN LLP <i>Pro hac vice</i> to be submitted 2121 N. Pearl Street, Suite 900 Dallas, TX 75201 Tel: (214) 453-6500 PClark@winston.com

Service Information: Service information is provided in the designation of counsel above. Petitioner consents to service of all documents via electronic mail at IPR-SAP-Val@winston.com.

C. Fee Authorization

The Office is authorized to charge all fees due in connection with this proceeding to Deposit Account No. 50-1814.

III. CHALLENGED CLAIMS

Under 37 C.F.R. §§ 42.104(b) and 42.22, Petitioner requests that the Board institute this IPR on the Challenged Claims of the '264 Patent and cancel those claims as unpatentable on the following grounds:

Ground	Claims	Basis for Unpatentability
1	1-3	Obviousness under 35 U.S.C. § 103 by <u>Cherabuddi</u>
2	4-5 and 8-10	Obviousness under 35 U.S.C. § 103 over <u>Cherabuddi</u> in view of <u>Dean</u>
3	11	Obviousness under 35 U.S.C. § 103 over <u>Cherabuddi</u> in view of <u>Noel</u>

As discussed below, Cherabuddi, Dean, and Noel qualify as prior art and are properly relied upon for showing unpatentability of the '264 Patent.

IV. THE '264 PATENT

A. Earliest Effective Filing Date

March 6, 2002, is the effective filing date because this is the filing date of the application that issued as the '264 Patent, and it does not claim priority to any other application. In parallel litigation, Patent Owners have asserted a conception date of May 15, 2001. While Petitioner does not waive its right to dispute any assertion that

the claims are entitled to an earlier invention date in the parallel litigation, the dispute is immaterial to this petition because all of the prior art references asserted here qualify as prior art to either date.

B. Level of Ordinary Skill in the Art

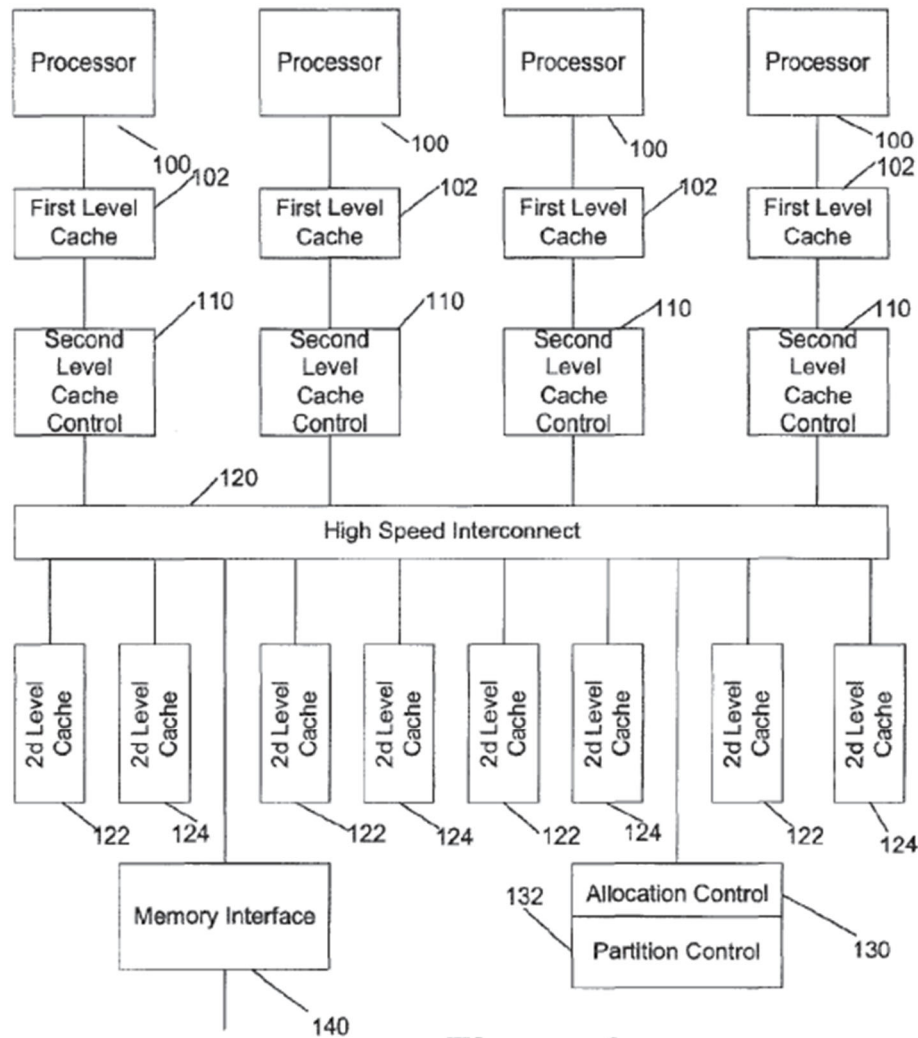
A POSITA in May 2001 or March 2002 would have had an understanding of multiprocessor design and/or multicore microprocessor design, as well as multiprocessor cache design. Ex-1006, ¶ 12. The POSITA would have had a bachelor's degree in computer science, electrical engineering, or computer engineering, and two years of relevant experience in memory-systems design, and related fields. Ex-1006, ¶ 12. Lack of professional experience can be remedied by additional education, and vice versa. Ex-1006, ¶ 12.

C. Overview of the '264 Patent

The '264 Patent describes a multi-processor system that is “capable of executing more than one instruction stream,” wherein each processor is connected through a cache controller to multiple cache memory blocks, some of which may be directly attached to specific controllers. Ex-1001, Abstract. These memory blocks are accessible via a high-speed interconnect, allowing dynamic allocation among multiple cache controllers. Ex-1001, Abstract. “A resource allocation controller determines which cache memory controller has access to the dynamically allocable cache memory block.” Ex-1001, Abstract.

In at least one embodiment, “there are four processors, four fixed-size first level caches each associated with a processor, four fixed-allocation second-level cache blocks, and four dynamically-allocable second-level cache blocks.” Ex-1001, 3:28-32. Dynamically allocable cache units are part of the second-level cache system. Ex. 1001, 3:27-28. The second-level cache system also includes “an operating system driver” associated with each monitor, which monitors and tracks “hit rates.” Ex-1001, 3:33-36.

Figure 1 depicts embodiments of the invention described in the '264 Patent.

**Figure 1**

Ex-1001, Fig. 1.

Figure 1 includes “[a] partitionable multiple-processor integrated circuit 98,” which “has several processors 100.” Ex-1001, 4:9–10. Each processor performs memory references for instructions data via a first-level cache 102. Ex-1001, 4:10–12. References that miss in the first-level cache are directed to second-level cache controllers 110. Ex-1001, 4:13–14. Each second-level cache controller 110 is

connected to a high-speed interconnect 120, enabling access to multiple cache memory blocks 122. Ex-1001, 4:15-19. Among these blocks, at least one can be dynamically allocated to any cache controller 110, while other memory blocks 122 may have fixed allocations. Ex-1001, 4:19-22. The allocation controller 130 specifies which dynamically allocated cache blocks 124 are accessed by each controller. Ex-1001, 4:23-25. The partition control 132 identifies which processors belong to each system partition, working with the allocation controller 130 to assign cache blocks to specific second-level cache controllers, processors, and system partitions. Ex-1001, 5:25-31.

V. CLAIM CONSTRUCTION

Claims are given their “ordinary and customary meaning” as understood by a POSITA and the prosecution history pertaining to the patent. 37 C.F.R. § 42.100(b). Because a POSITA would find the challenged claims unpatentable under any interpretation consistent with their plain and ordinary meaning in the context of the ’264 Patent, the Board need not expressly construe the claim terms. *See Vivid Techs., Inc. v. Am. Sci. & Eng’g. Inc.*, 200 F.3d 795, 803 (Fed. Cir. 1999).

VI. PRINCIPAL PRIOR ART

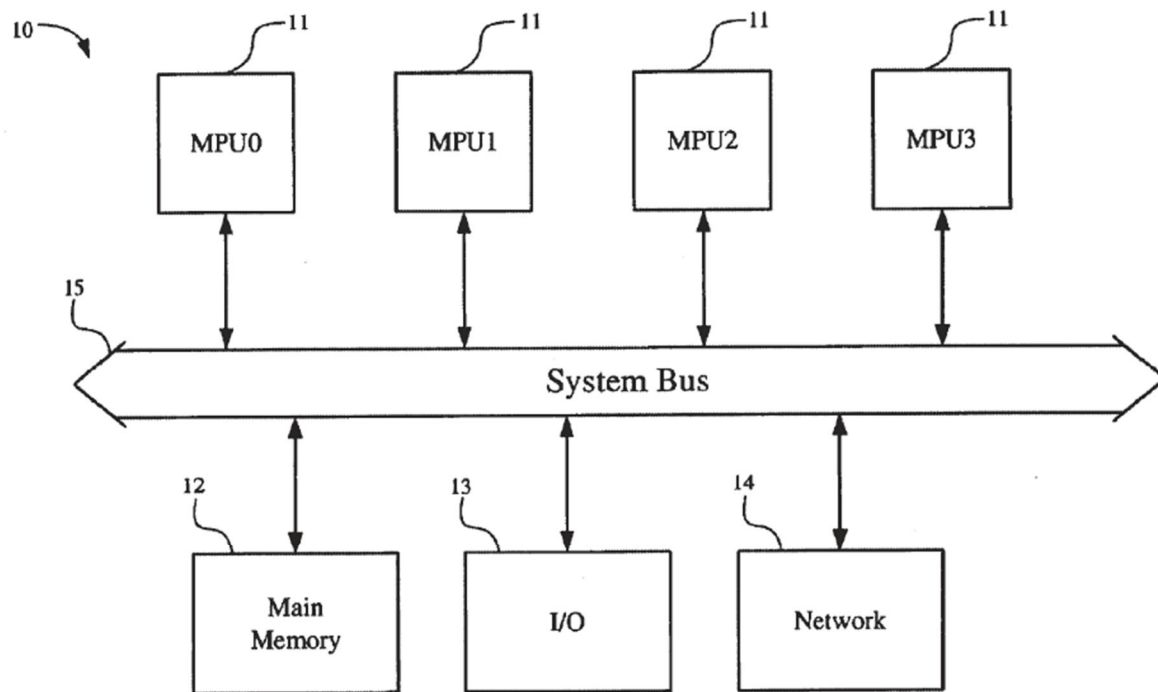
A. Summary of Cherabuddi

U.S. Patent No. 6,725,336 to Cherabuddi (Ex-1002) was filed on April 20, 2001, and issued on April 20, 2004. Cherabuddi qualifies as prior art to the ’264 Patent at least under pre-AIA 35 U.S.C. § 102(e).

Cherabuddi relates “generally to multiprocessor computer system and specifically to cache memory of multiprocessor computer systems.” Ex-1002, 1:8-10. Cherabuddi describes a multi-processor unit (“MPU”), comprising two or more central processing units (CPUs), including a cache memory and main memory. Ex-1002, 1:12-18. Cherabuddi explains that cache memory is smaller and faster than main memory, and when there is a cache miss (i.e., the requested data by the CPUs cannot be located on the cache memory), the CPU must request information from the slower main memory, thereby reducing performance and speed. Ex-1002, 1:17-27. Cherabuddi also explains that in some MPU architectures, including a single cache memory shared among multiple CPUs, because more than one CPU can access the same cache memory locations, “chip-level snoop operations are required between the CPUs on each MPU,” which “undesirably increase[s] the size and complexity of the associated cache controllers.” Ex-1002, 1:36-38, 41-42. Cherabuddi also explains that in other MPU architectures, “only one CPU has access to any given cache memory location,” and “if one of the CPUs becomes defective . . . its dedicated cache memory is not accessible by the other CPUs, thereby wasting cache resources.” Ex-1002, 1:45-46, 50-53.

As a solution, Cherabuddi proposes partitioning cache memory dynamically between two or more processors on an MPU based on “a desired system configuration or to the processing needs of the processors.” Ex-1002, 1:61-66. In

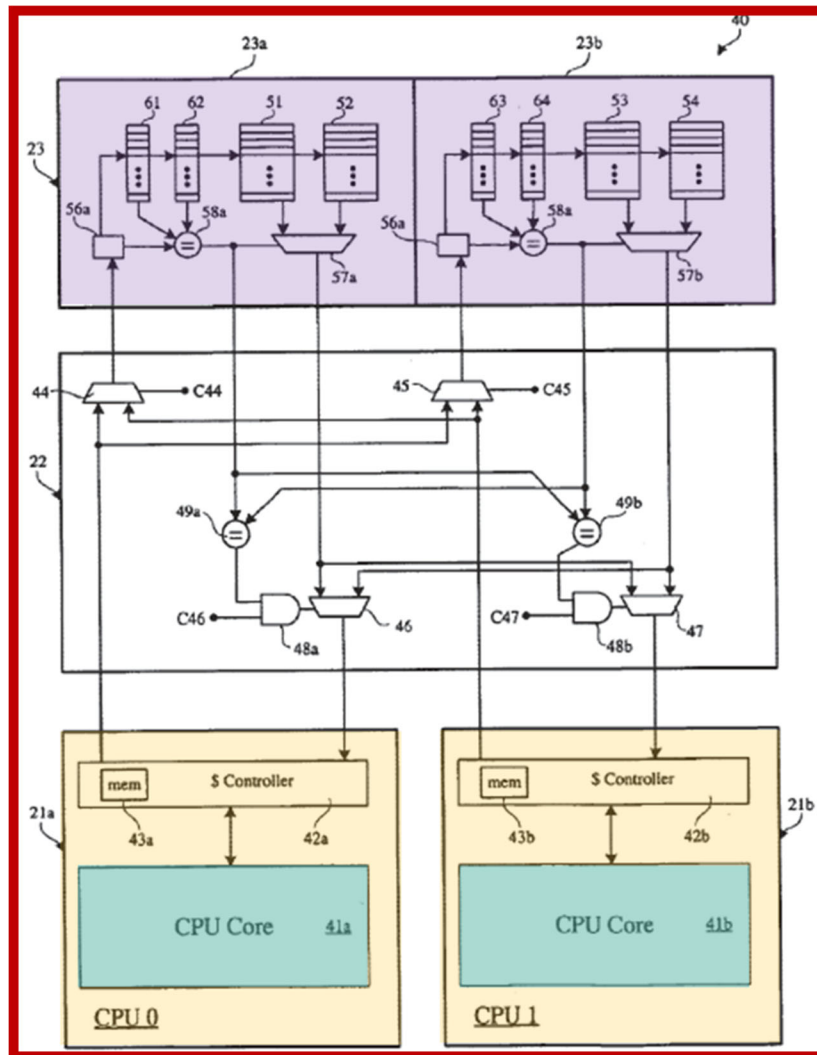
particular, a cache memory is partitioned between two or more processors and the system treats each cache memory partition as a dedicated cache memory. Ex-1002, 2:2-12. The entire cache memory can be dynamically allocated between multiple processors to maximize cache resources while allowing each processor and corresponding allocated cache memory partition to operate independently. Ex-1002, 2:39-46. This invention also allows for any processor to access the entire cache memory in case another processor is found to be defective, maximizing cache resources. Ex-1002, 2:46-49. The basic architecture of system 10 includes four MPUs 11 connected to each other and to a main memory 12, an input/output device 13 (e.g., “a computer monitor, keyboard, and/or mouse input”), and a network 14 (e.g., “a local area network, a wide area network, and/or the Internet”) via a system bus 15. Ex-1002, 3:13-17, 19-24.

**FIG. 1**

Ex-1002, Fig. 1.

Another embodiment of Cherabuddi is depicted by Figure 4, which includes CPUs 21a-21b, cache access circuit 22, and cache memory partitions 23a-23b. Ex-1002, 4:66-5:1. Each CPU 21 includes a CPU core 41 and a cache controller 42. Ex-1002, 5:1-2. “Each cache controller 42 transfers address and data between its associated CPU core 41 and cache access circuit 22, and includes (or is associated with) a memory element 43.” Ex-1002, 5:2-6. Each “CPU core 41 includes other well-known elements of CPU 21 including, for instance, L1 cache memory,

instruction units, fetch and decode units, execution units, register files, write cache(s), and so on.” Ex-1002, 5:10-14.



dynamically allocated
cache memory

“multiple-processor
integrated circuit”

multiple processors

CPU Core

FIG. 4

Ex-1002, Fig. 4.

B. Summary of Dean

U.S. Patent No. 6,604,174 to Dean (Ex-1003) was filed on November 10, 2000, and issued on August 5, 2003. Dean qualifies as prior art to the '264 Patent at least under pre-AIA 35 U.S.C. § 102(e).

Dean “generally relates to cache circuits, and more specifically relates to a performance based system and method for dynamic allocation of a unified multiport cache.” Ex-1003, 1:8–10. Dean describes “a performance based system and method for dynamic allocation of a unified multiport cache.” Ex-1003, 2:66-3:1. Dean explains that the cache can be allocated to different processors and then dynamically reallocated based on performance. Ex-1003, 3:6-8. Dean teaches using a relational cache miss percentage to reallocate the cache. Ex-1003, 3:8-11.

C. Summary of Noel

U.S. Patent No. 6,381,682 to Noel (Ex-1004) was filed on June 10, 1998, and issued on April 30, 2002. Noel qualifies as prior art to the ’264 Patent at least under pre-AIA 35 U.S.C. § 102(e).

Noel “relates to multiprocessor computer architectures in which processors and other computer hardware resources are grouped in partitions, each of which has an operating system instance and, more specifically, to methods and apparatus for allocating computer hardware resources to partitions.” Ex-1004, 1:6-11.

VII. CLAIMS 1-5 AND 8-11 ARE UNPATENTABLE**A. Ground 1: Claims 1-3 Would Have Been Obvious Based on Cherabuddi****1. Claim 1**

- i. ***[1.0] A processor integrated circuit capable of executing more than one instruction stream comprising:***

The '264 Patent describes “*a processor integrated circuit*” as being a hardware component containing multiple cache units and multiple instruction and fetch and decode units, where each instruction fetch and decode unit is associated with a real or virtual processor.” Ex-1001, 3:15-18. The processor integrated circuit also has “at least one dynamically allocable cache unit” and a “high-speed interconnect.” Ex-1001, 3:18-22. The MPU taught in Cherabuddi includes “two or more central processing units (CPUs) on a single chip” (“*a processor integrated circuit*”). Ex-1002, 1:12-14; Ex-1005, ¶ 87. The MPU is illustrated in Figure 4.

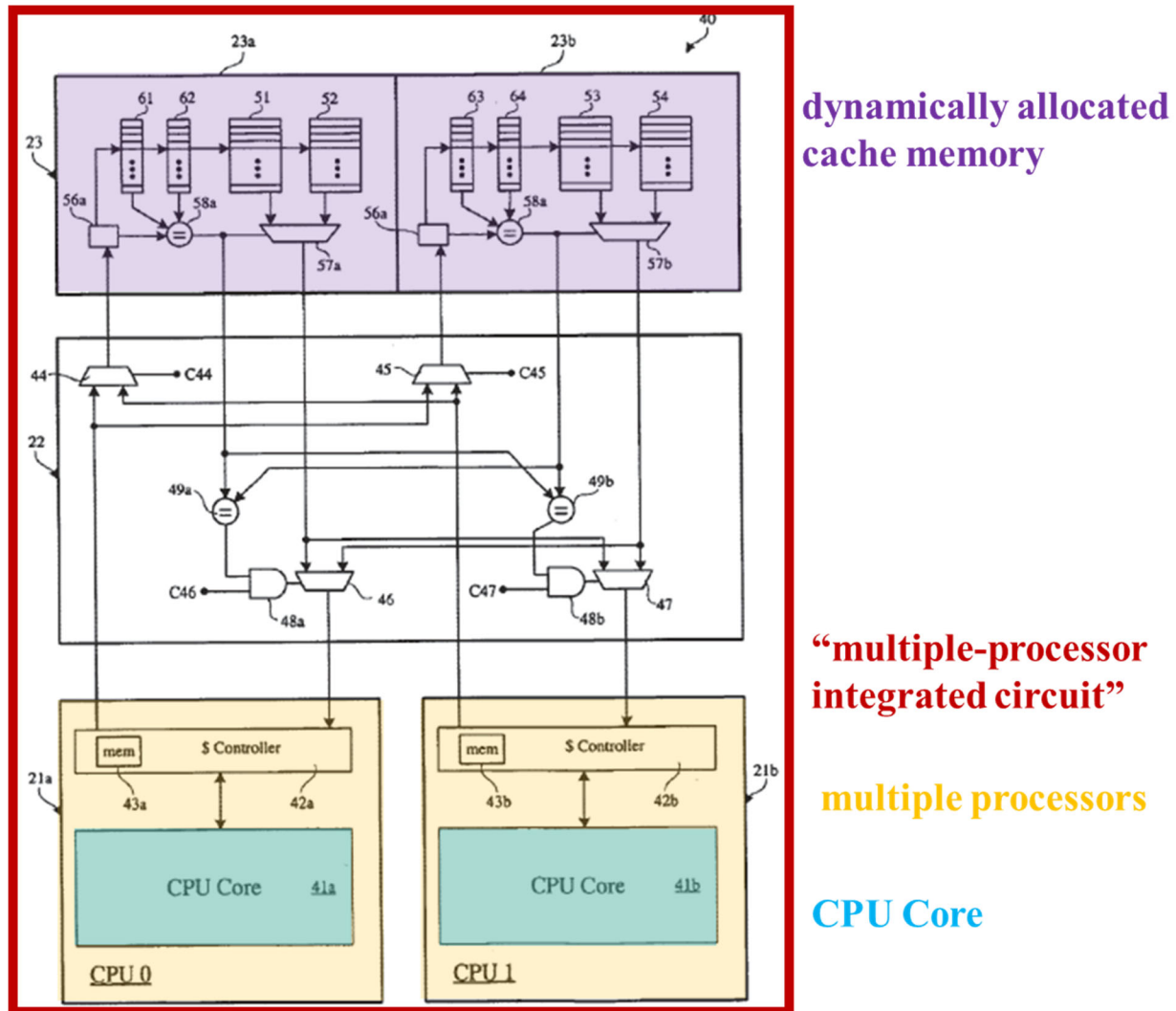


FIG. 4

Ex-1002, Fig. 4 (annotated).

Figure 4 shows “an MPU 40 . . . and includes CPUs 21a-21b, cache access circuit 22, and cache memory partitions 23a-23b.” Ex-1002, 4:66-5:1. And “[e]ach CPU 21 is shown to include a CPU core 41 and a cache controller 42.” Ex-1002, 5:1-2. Each CPU core contains L1 cache memory (cache units), instruction units, and fetch and decode units, which are connected to the particular CPU within.

Ex-1002, 5:9-13. Figure 4 also depicts a dynamically allocated cache memory 23 (dynamically allocable cache unit) and a high-speed interconnect, as shown in claim [1.4]. Ex-1002, 3:2-5; Ex-1005, ¶ 90.

Consistent with Figure 4, the “processors [of the MPU] are set as active, and may simultaneously execute separate instruction threads” (“*a processor integrated circuit capable of executing more than one instruction stream*”). Ex-1002, 2:8-9; *see also* Ex-1002, 2:39-44¹ (“The ability to dynamically allocate cache resources between multiple processors advantageously allows the entire cache memory to be used, irrespective of whether one or both processors are currently active, thereby maximizing cache resources while **allowing for** both one-thread and **two-thread execution modes**.”). Ex-1005, ¶ 88.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 88.

- i. ***[1.1] a first processor, coupled to fetch instructions and access data through a first cache controller;***

Cherabuddi teaches that MPU 20 includes “CPUs 21a-21b, cache access circuit 22, and cache memory partitions 23a-23b.” Ex-1002, 4:66-5:1. Each CPU 21 contains “a CPU core 41 and a cache controller 42.” Ex-1002, 5:1-2. That is, CPU 21a (“*a first processor*”) includes (“*coupled to*”) “cache controller 42a” (“*a first cache controller*”). Ex-1002, 5:1-2l; Ex-1005, ¶¶ 89, 91.

¹ Emphasis added in bold throughout unless otherwise stated.

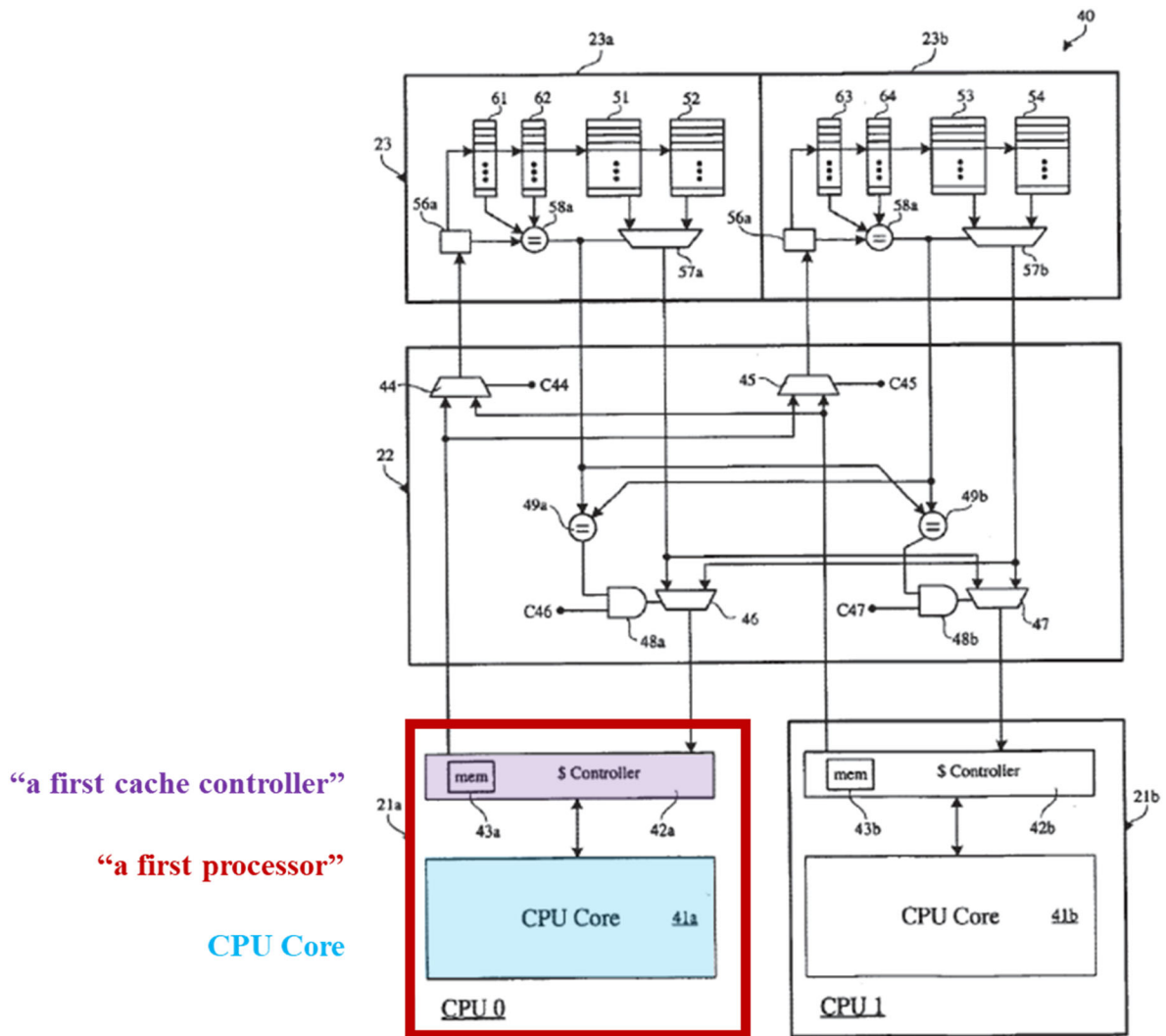


FIG. 4

Ex. 1002, Fig. 4 (annotated).

Cherabuddi explains that the first processor is “coupled to fetch instructions and access data through a first cache controller.” A POSITA would understand that cache memory 23 is a unified cache that holds both instructions and data, and a processor accesses cached instructions and data through a cache controller. Ex-1005, ¶¶ 20, 32, 91; Ex-1002, 5:2-6 (“[e]ach cache controller 42 . . . transfers

address and data between its associated CPU core 41 and cache access circuit 22). Indeed, an MPU contains cache memory “to store data in anticipation of future use by the CPUs.” Ex-1002, 1:16-17. For CPU 21a to request data from cache memory partition 23a, “CPU 21a provides a main memory address to address converter 56a via MUX 44.” Ex-1002, 6:42-44. Cherabuddi further explains how CPU 21a (“*a first processor*”) is “*coupled . . . to access data*” (Ex-1002, 6:57-59 (teaching that “[s]ince C46=0, MUX 46 provides matching data from cache memory partition 23a to cache controller 42a of CPU 21a”)) “*through a first cache controller.*”

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 89.

ii. [1.2] *a second processor, coupled to fetch instructions and access data through a second cache controller;*

As depicted in Figure 4, CPU 21b (“*a second processor*”) includes “CPU core 41b” and “cache controller 42b” (“*a second cache controller*”). Ex-1002, 5:1-2, Fig.

4.

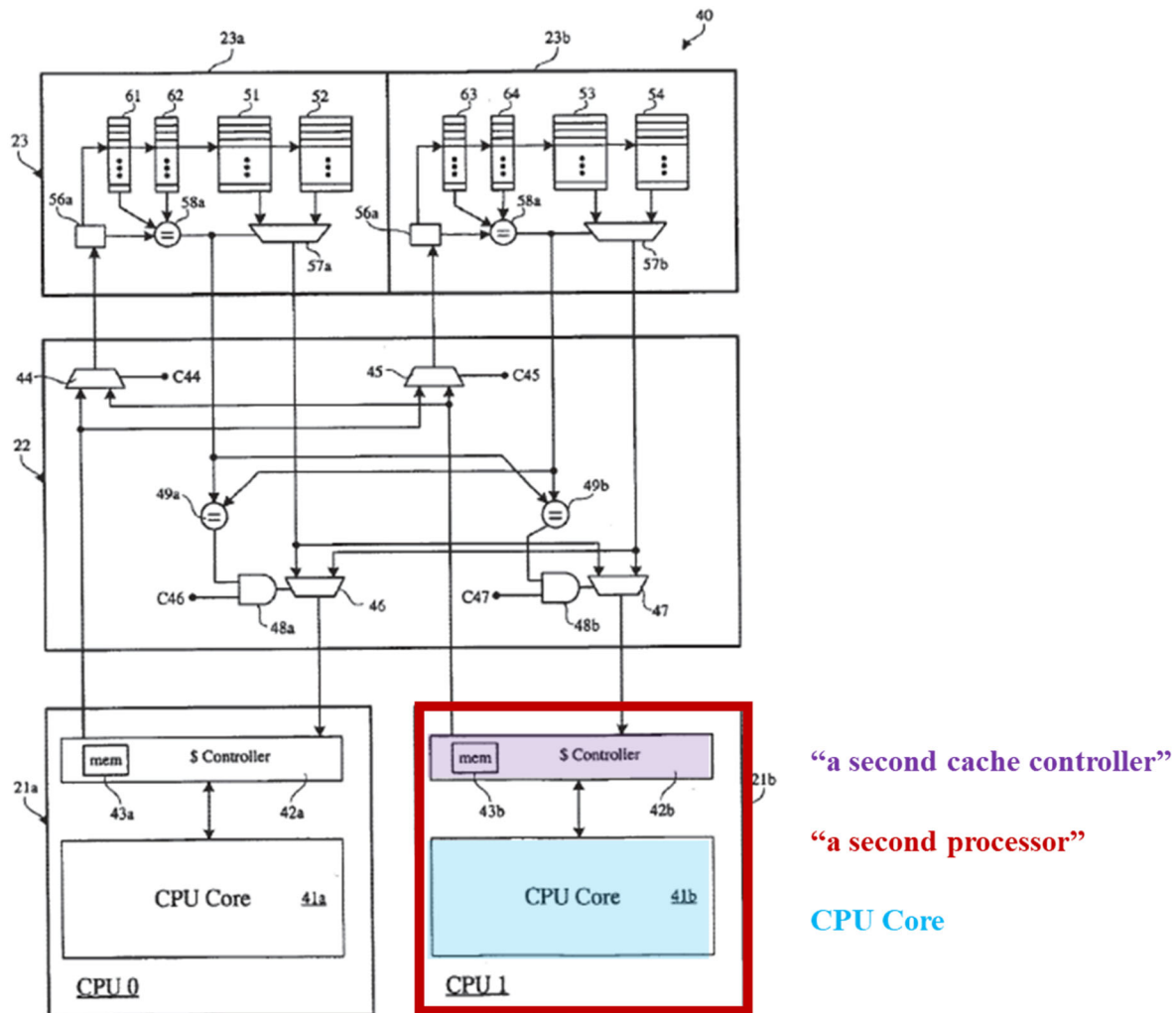


FIG. 4

Ex-1002, Fig. 4 (annotated); Ex-1005, ¶ 91.

Further, Cherabuddi teaches that the second processor is “coupled to fetch instructions and access data through a second cache controller” in the same way as the first processor, described for limitation [1.1] above. Ex-1005, ¶ 91.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 89.

iii. [1.3] a plurality of cache memory blocks;

Figure 4 depicts an MPU 40 and includes cache memory partitions 23a-23b.

Ex-1002, 4:66-5:1.

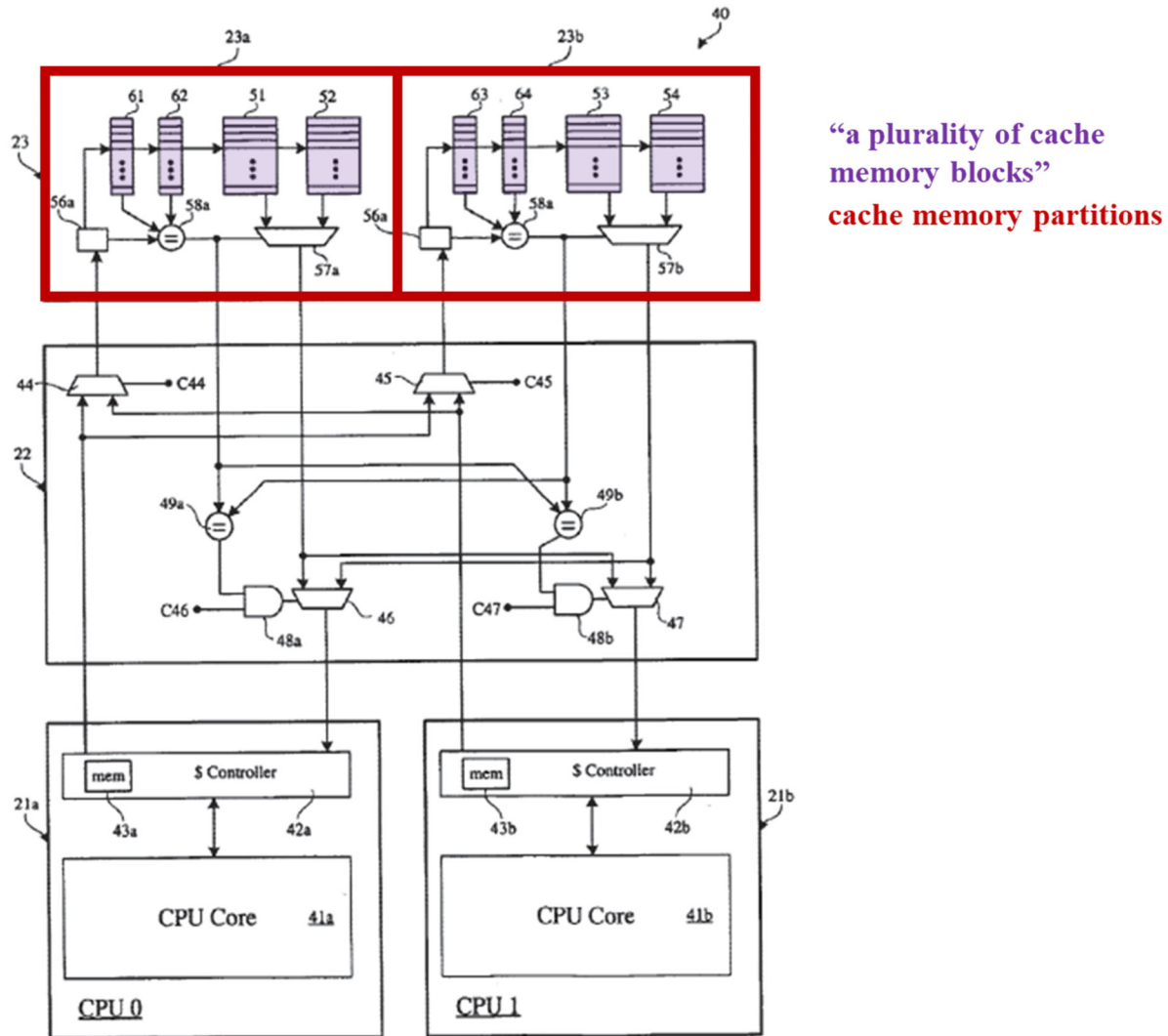


FIG. 4

Ex-1002, Fig. 4 (annotated); Ex-1005, ¶ 92.

Cherabuddi teaches that “cache memory partition 23a includes two data RAM arrays 51-52 having corresponding searchable tag arrays 61-62, respectively, while cache memory partition 23b includes two data RAM arrays 53-54 having

corresponding searchable tag arrays 63-64, respectively.” Ex-1002, 5:15-19. These “[d]ata arrays 51-54 each include a plurality of cache lines for storing data received from main memory 12.” Ex-1002, 5:29-30, 31-32. Further, “[t]ag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively.” Ex-1002, 5:39-41. The ’264 Patent describes that cache memory blocks 198 “incorporate[] data memory 200” and “tag memory 202.” Ex-1001, 4:37-39. Therefore, a POSITA would understand that data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*a plurality of cache memory blocks*” as they include the entire tag/data set for each cache memory partition. Ex-1005, ¶¶ 92, 93, 94 (citing Ex-1002, 4:66-5:14, 5:15-49).

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 92.

- iv. ***[1.4] a high-speed interconnect coupling the plurality of cache memory blocks to the first and second cache controllers such that at least one allocable cache memory block is capable of being used by the first and second cache controllers; and***
 - a. **“a high-speed interconnect coupling the plurality of cache memory blocks to the first and second cache controllers . . .”**

The ’264 Patent describes that “*a high-speed interconnect*” “permits use of the dynamically allocable cache units by more than one real or virtual processor.” Ex-1001, 3:22-24. As discussed for limitation [1.0], Figure 4 shows MPU 40, which

includes “CPUs 21a-21b, cache access circuit 22, and cache memory partitions 23a-23b.” Ex-1002, 4:66-5:1. As depicted in Figure 4, a POSITA would understand that “*a high-speed interconnect*” is shown based on the wires that connect the first cache controller 42a and the second cache controller 42b to the cache memory partition 23a and the cache memory partition 23b, respectively over the cache access circuit 22. Ex-1005, ¶¶ 95, 29; Ex-1002, Fig. 4; Ex-1002, 3:40-42 (“Cache access circuit 22 selectively couples the first and/or second CPUs 21a-21b to the first and/or second cache memory 23a-23b.”).

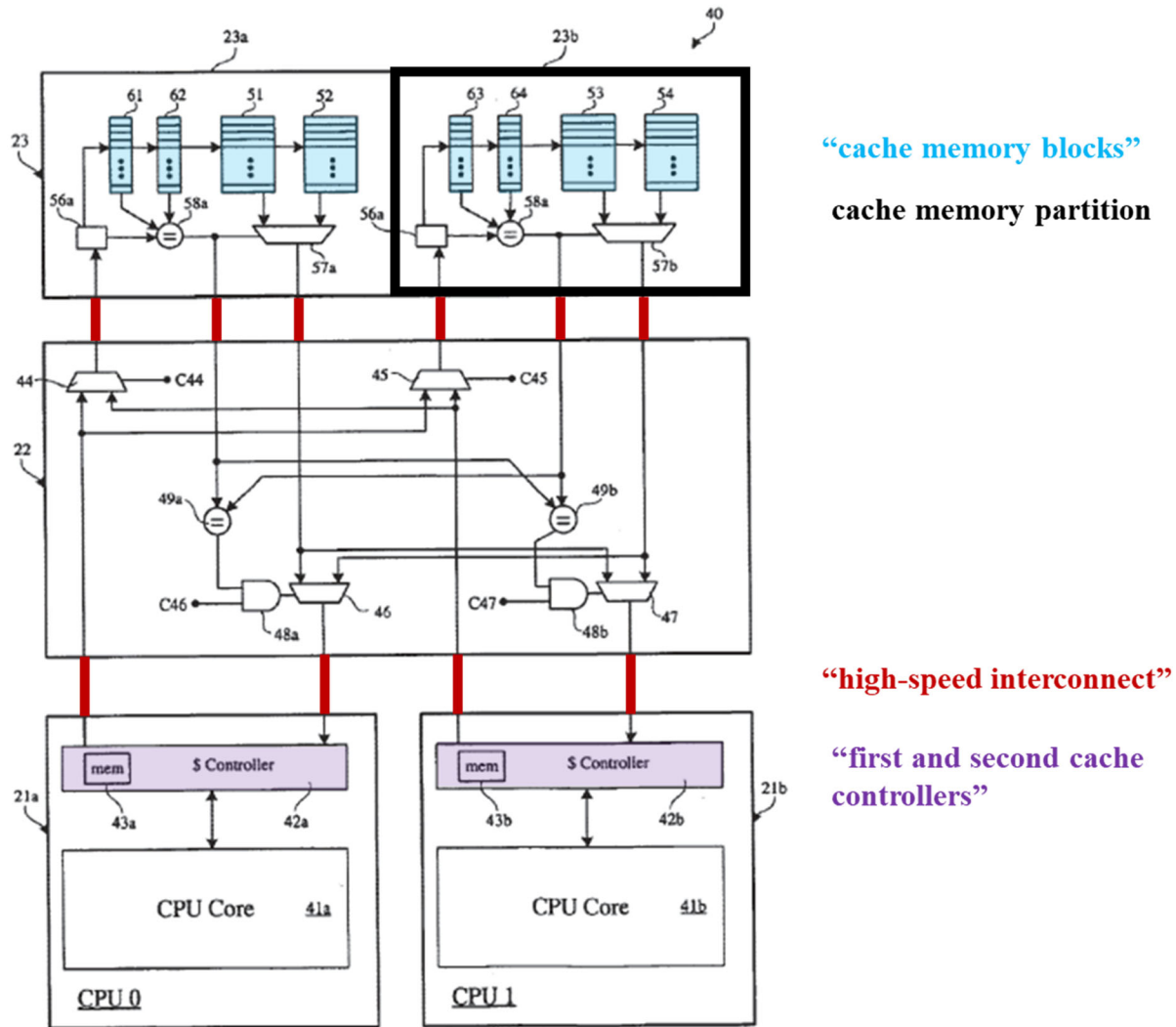


FIG. 4

Ex-1002, Fig. 4 (annotated). As discussed in limitations [1.1] and [1.2], Cherabuddi explains cache controller 42a (“*the first cache controller*”) and cache controller 42b (“*the second cache controller*”) “transfers address and data between its associated CPU core 41 and cache access circuit 22.” Ex-1002, 5:4-5. As discussed for limitation [1.3], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*a plurality of cache memory blocks.*” Ex-1005, ¶¶ 93,

94 (citing Ex-1002, 4:66-5:14, 5:15-49). The “*high-speed interconnect*” depicted in Figure 4 is connecting (“*coupling*”) cache memory partitions, containing data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively (“*the plurality of cache memory blocks*”), to cache controllers 42a and 42b (“*to the first and second cache controllers*”). Ex-1002, Fig. 4; Ex-1005, ¶ 98.

b. “...such that at least one allocable cache memory block is capable of being used by the first and second cache controllers”

Cherabuddi teaches two modes: (1) In the 1T state (where $M=0$), the kernel sets one of the CPUs to an active state and sets the other CPU to an inactive state, and (2) in the 2T state (where $M=1$), the kernel sets both CPUs to an active state. Ex-1002, 4:4-10, 4:15-17. In the 1T state ($M=0$), cache access circuit 22 couples the active CPU 21 to both the first and second cache memory partitions 23a-23b to allow the active CPU 21a to use all sources of cache memory 23. Ex-1002, 4:7-10. In the 1T state, the kernel can set either CPU 21a or 21b, containing cache memory controllers 42a and 42b (“*first and second cache controllers*”), respectively, as active to utilize data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively (“*at least one allocable cache memory block is capable of being used*”). Ex-1002, 4:7-10; Ex-1005, ¶ 96.

Indeed, Cherabuddi teaches how “*at least one allocable cache memory block*” can be used by “*the first and second cache controllers.*” Cherabuddi explains that

“[t]o request data from both cache memory partitions 23a–23b, CPU 21a provides a main memory address to address converter 56a via MUX 44 and to address converter 56b via MUX45. Ex-1002, 7:30-33. “[T]he cache address is provided to data arrays 51-54 and to tag arrays 61-64.” Ex-1002, 7:33-34. Figure 4 shows that MUX 44 and 45 are connected to cache controller 42a (“*first cache controller*”) and cache controller 42b (“*second cache controller*”), which “transfer[] address and data between [their] associated CPU core 41 and cache access circuit 22.” Ex-1002, 5:2-5; Ex-1005, ¶¶ 97-98.

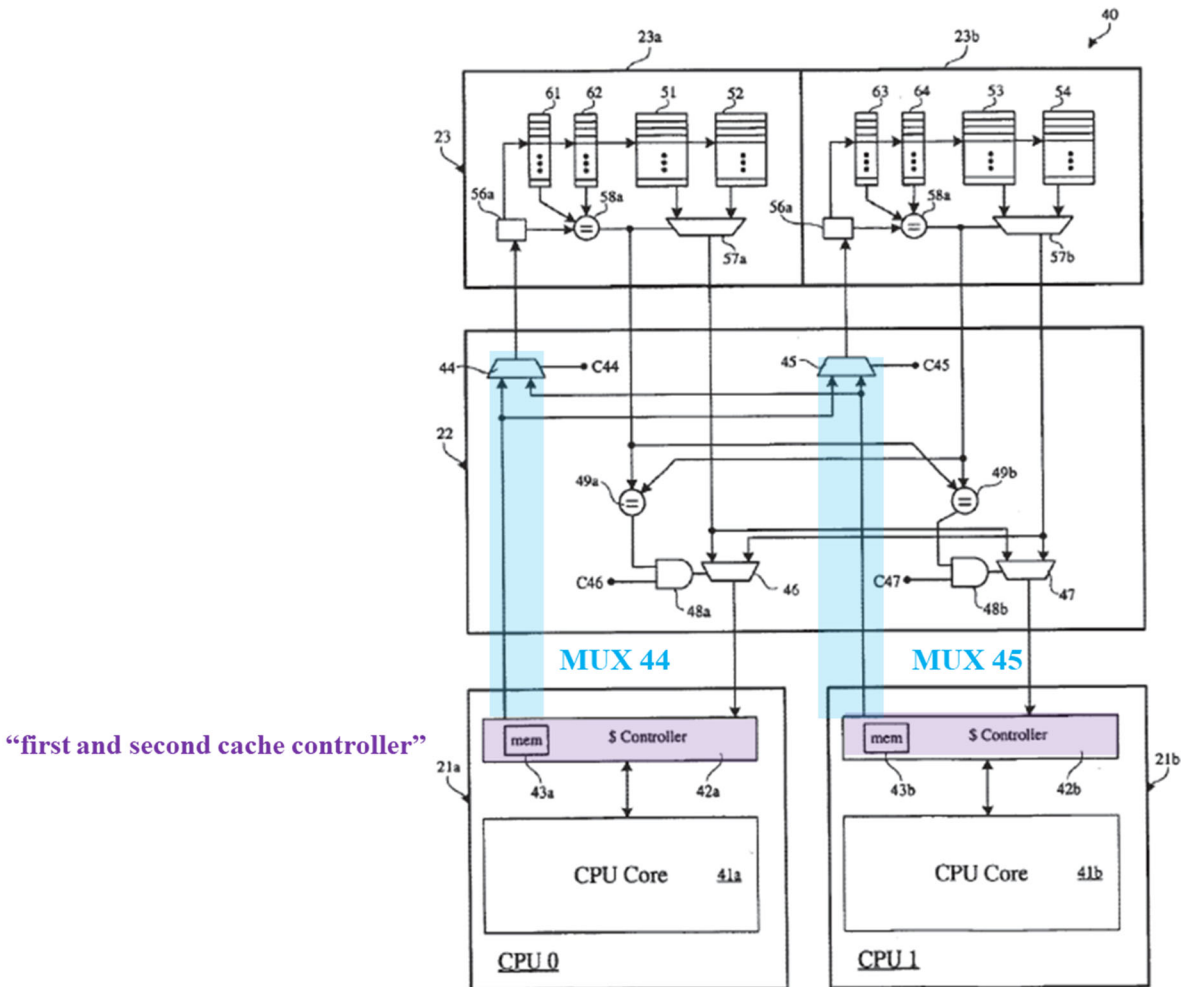


FIG. 4

Cherabuddi explains that “[d]ata arrays 51-52 read out the selected cache line to MUX 57a, and tag arrays 61-62 read out corresponding tag fields to comparator 58a. Comparator 58a compares the tag fields with the tag address received from address converter 56a, and selects which data (if any) MUX 57a forwards to MUX 46.” Ex-1002, 7:34-39. “Similarly, data arrays 53-54 read out the selected cache line to MUX 57b, and tag arrays 63-64 read out corresponding tag fields to comparator 58b. Comparator 58b compares the tag field with the tag address

received from address converter 56b, and selects which data (if any). MUX 57b forwards to MUX 46.” Ex-1002, 7:39-45.

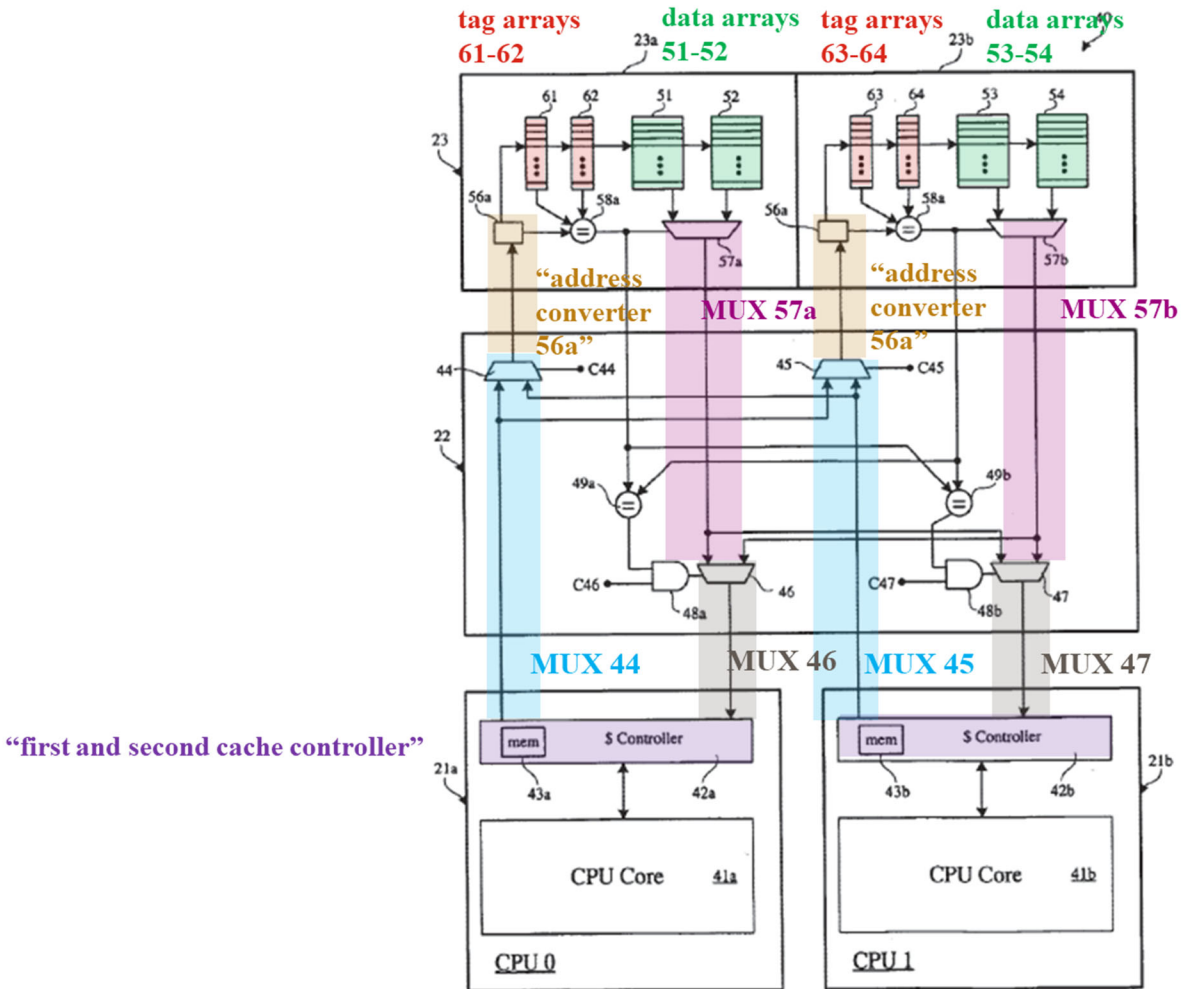


FIG. 4

Ex-1002, Fig. 4 (annotated).

Cherabuddi explains that “MUX 46 selectively returns data to CPU 21a from either cache memory partition 23a or 23b in response to AND gate 48a, and MUX 47 selectively returns data to CPU 21b from either cache memory partition 23a or 23b in response to AND gate 48.” Ex-1002, 6:14-18. Figure 4 also shows that MUX

46 and 47 are connected to cache controllers 42a and 42b. Here, CPUs 21a and 21b, via cache controllers 42 and 42b (“*first and second cache controllers*”) can selectively provide data and information to and receive data and information from data arrays 51-54 and tag arrays 61-64 within cache memory partitions 23a and 23b (“*at least one allocable cache memory block is capable of being used by the first and second cache controllers*”). Ex-1005, ¶ 98.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 96.

- v. ***[1.5] a resource allocation controller coupled to determine an accessing cache memory controller selected from the group consisting of the first and second cache memory controllers, whereby the accessing cache memory controller is allowed to access the allocable cache memory block,***
 - a. **“a resource allocation controller coupled to determine an accessing cache memory controller selecting from the group consisting of the first and second cache memory controllers . . .”**

The “*resource allocation controller*” of the ’264 Patent “determines which cache memory controller has access to the dynamically allocable cache memory block.” Ex-1001, Abstract; Ex-1005, ¶ 99. Cache access circuit 22 of Cherabuddi is a “*resource allocation controller*” because cache access circuit 22 controls accessibility of the cache memory controllers to the dynamically allocable cache memory block. Ex-1002, 3:42-46. Indeed, Cherabuddi explains that “cache access circuit 22 allows the resources of cache memory 23 to be **dynamically allocated**

between the first and second CPUs 21a-21b according to each CPU's processing requirements to more efficiently utilize cache resources.” Ex-1002, 3:42-46; Ex-1005, ¶ 100.

Each CPU includes “a CPU core 41 and a cache controller 42.” Ex-1002, 5:1-2. “Each cache controller 42” (“*group consisting of the first and second cache memory controllers*”) “transfers address and data between its associated CPU core 41 and cache access circuit 22.” Ex-1002, 5:2-6. Cherabuddi teaches that a cache access circuit 22 “selectively couples the first and/or second CPUs 21a-21b to the first and/or second cache memory partitions 23a-23b.” Ex-1002, 3:40-42. “[C]ache access circuit 22 allows the resources of cache memory 23 to be dynamically allocated between the first and second CPUs 21a-21b,” containing the first and second cache memory controllers, “according to each CPU's processing requirements to more efficiently utilize cache resources.” Ex-1002, 3:42-46. In the 1T state (M=0), the system kernel sets one of the CPUs 21 as active and sets the other CPU 21 as inactive, including the cache memory controllers therein. Ex-1002, 3:63-65. Here, cache access circuit couples the active CPU, containing a cache memory controller (“*access cache memory controller*”), to both the first and second cache memory partitions 23a-23b, allowing the active CPU to use all sources of cache memory 23. Ex-1002, 4:7-10 (“[C]ache access circuit 22 couples the first CPU 21a to both the first and second cache memory partitions 23a-23b to allow the

first CPU 21a to use all sources of cache memory 23.”). In other words, the cache access circuit (“*resource allocation controller*”) selectively couples the active CPU—either CPU 21a or 21b containing cache memory controllers (“*group consisting of the first and second cache memory controllers*”)—to the first and second cache memory partitions 23a-23b to allow the active CPU to use all sources of cache memory 23 (“*a resource allocation controller coupled to determine an accessing cache memory controller*”). Ex-1002, 4:7-10; Ex-1005, ¶ 101.

b. “...whereby the accessing cache memory controller is allowed to access the allocable cache memory block”

In the 1T state (M=0), cache access circuit 22 couples the active CPU 21 to both the first and second cache memory partitions 23a-23b to allow the active CPU 21 to use all sources of cache memory 23. Ex-1002, 4:7-10. CPU 21a or 21b, containing cache memory controller 42a and 42b (“*the accessing cache memory controller*”), respectively, is set as active to allow the active CPU “to use all resources of cache memory 23,” thereby accessing data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively (“*allowed to access the allocable cache memory block*”). Ex-1002, 4:7-10; Ex-1005, ¶ 101.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 99.

- vi. ***[1.6] wherein the cache memory blocks are usable by the cache controllers to store data and instructions fetched from a random-access memory.***

Cherabuddi teaches that “[e]ach cache controller 42” (“*the cache controllers*”) “transfers address and data between its associated CPU core 41 and cache access circuit 22.” Ex-1002, 5:2-6. For the same reasons discussed for limitation [1.3], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*the cache memory blocks.*” Ex-1005, ¶¶ 93, 94 (citing Ex-1002, 4:66-5:14, 5:15-49).

In the 2T state (M=1), cache access circuit couples CPUs 21a and 21b (containing cache controller 42a and 42b (“*the cache controllers*”)) to cache memory partition 23a and 23b, respectively (containing the data arrays 51-54 and tag arrays 61-64 (“*cache memory blocks*”), for exclusive access thereto. Ex-1002, 4:23-27. Cherabuddi explains that data from CPUs 21a and 21b to cache memory partition 23a and 23b in the 2T state is accomplished as follows: “C44=0 forces MUX 44 to provide an address or data from CPU 21a to cache memory partition 23a, C45=0 forces MUX 45 to provide an address or data from CPU 21b to cache memory partition 23b.” Ex-1002, 6:34-37. Figure 4 shows that MUX 44 and 45 are connected to cache controller 42a and cache controller 42b, which “transfer[] address and data between [their] associated CPU core 41 and cache access circuit 22.” Ex-1002, 5:2-5.

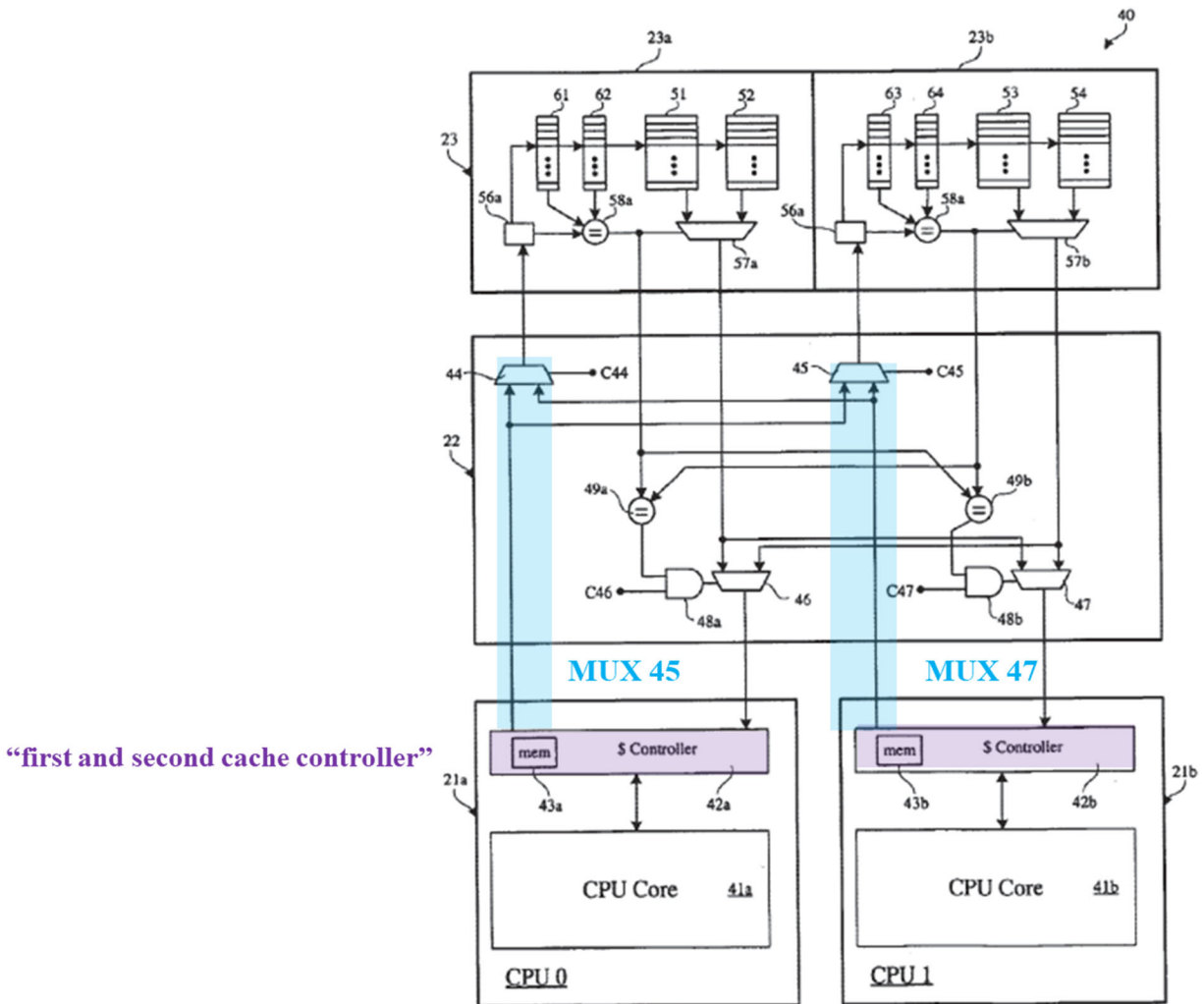


FIG. 4

Ex-1002, Fig. 4 (annotated). Here, CPUs 21a and 21b, via cache controllers 42 and 42b (“the cache controllers”) can selectively provide data and information to data arrays 51-54 and tag arrays 61-64 within cache memory partitions 23a and 23b (“wherein the cache memory blocks are usable by the cache controllers”). Ex-1005, ¶ 102. Cherabuddi explains that “[t]ag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively” (Ex-1002, 5:39-41), and “[d]ata arrays 51-54 each include a plurality

of cache lines for storing data retrieved from main memory 12” (Ex-1002, 5:29-30), which “may be any suitable random access memory (RAM) such as, for example, DRAM” (“to store data and instructions fetched from a random-access memory”) (Ex-1002, 3:17-18). A POSITA would understand that cache memory 23 is a unified cache that holds both instructions and data (often singularly referred to as “data” because instructions are data. Ex-1005, ¶¶ 102, 106. Further, Cherabuddi teaches that MPU 40 shown in Figure 4 is part of a system of main memory that the cache memory blocks store data and instructions fetched from main memory, DRAM. Ex-1002, 3:17-22, 3:26-27 (“Fig. 2 shows an MPU 20 that is an embodiment of MPU 11 of Fig. 1.”); 4:66-67 (“Fig. 4 shows an MPU 40 that is an embodiment of MPU 20.”); Ex-1005, ¶¶ 103-05.

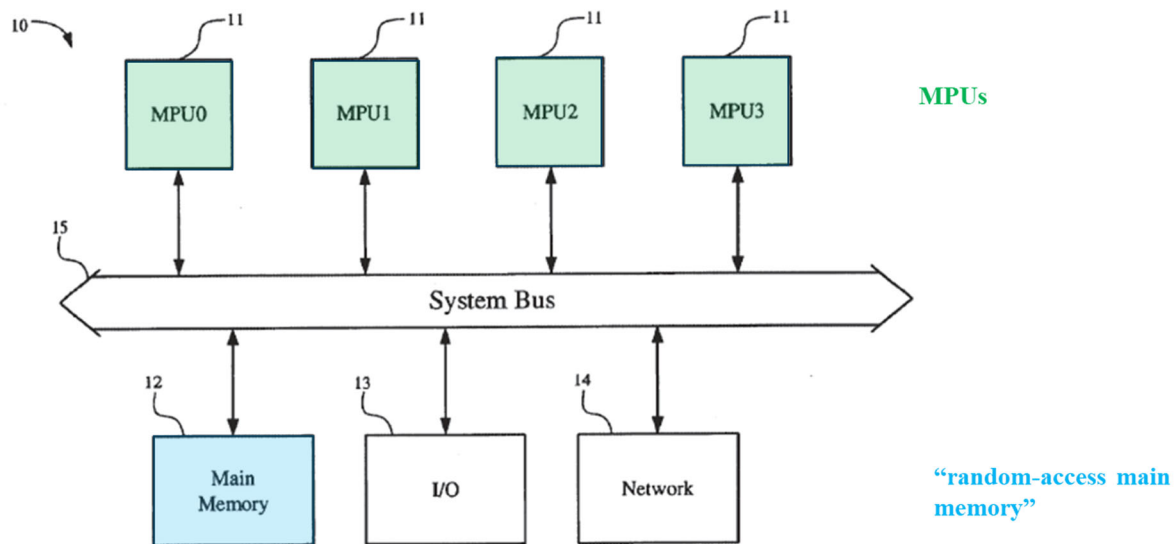


FIG. 1

Ex.1002, Fig. 1 (annotated).

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 102.

2. Claim 2

- i. *[2.0] The processor integrated circuit of claim 1, further comprising a plurality of first level cache systems, wherein the first processor fetches instructions and accesses data from the first cache controller through a first first level cache system, and wherein the second processor fetches instructions and accesses data from the second cache controller through a second first level cache system.*

Cherabuddi teaches “a plurality of first level cache systems.” Figure 4 depicts an MPU containing “CPUs 21a-21b, cache access circuit 22, and cache memory partitions 23a-23b.” Ex-1002, 4:66-5:1. Each CPU includes both a CPU core and a cache controller. Ex-1002, 5:1-2. Each CPU core includes “**L1 cache memory**” (“a plurality of first level cache systems”). Ex-1002, 5:10-13. Indeed, Figure 4 shows that CPUs 21a-21b respectively contain CPU core 41a-41b, which themselves contain an L1 (first-level) cache memory, not expressly depicted in Figure 4

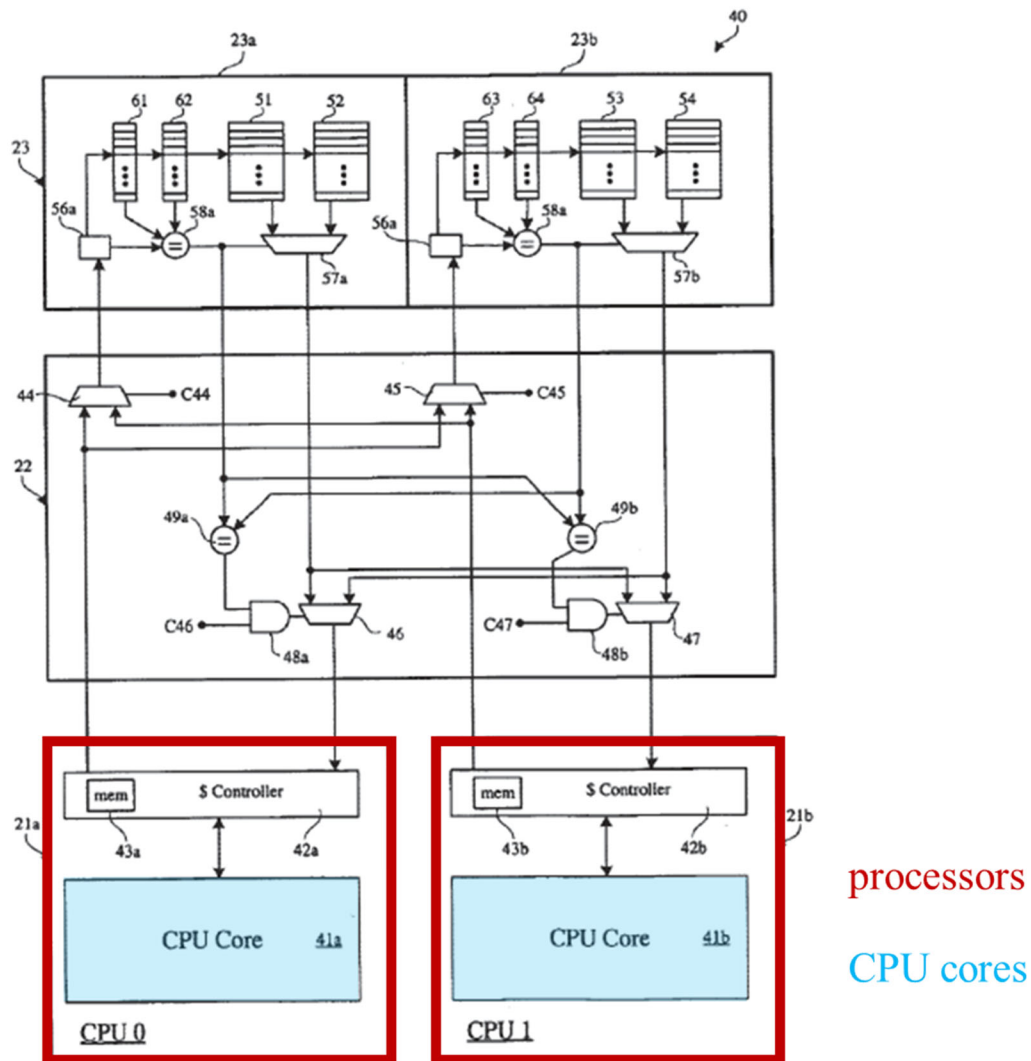
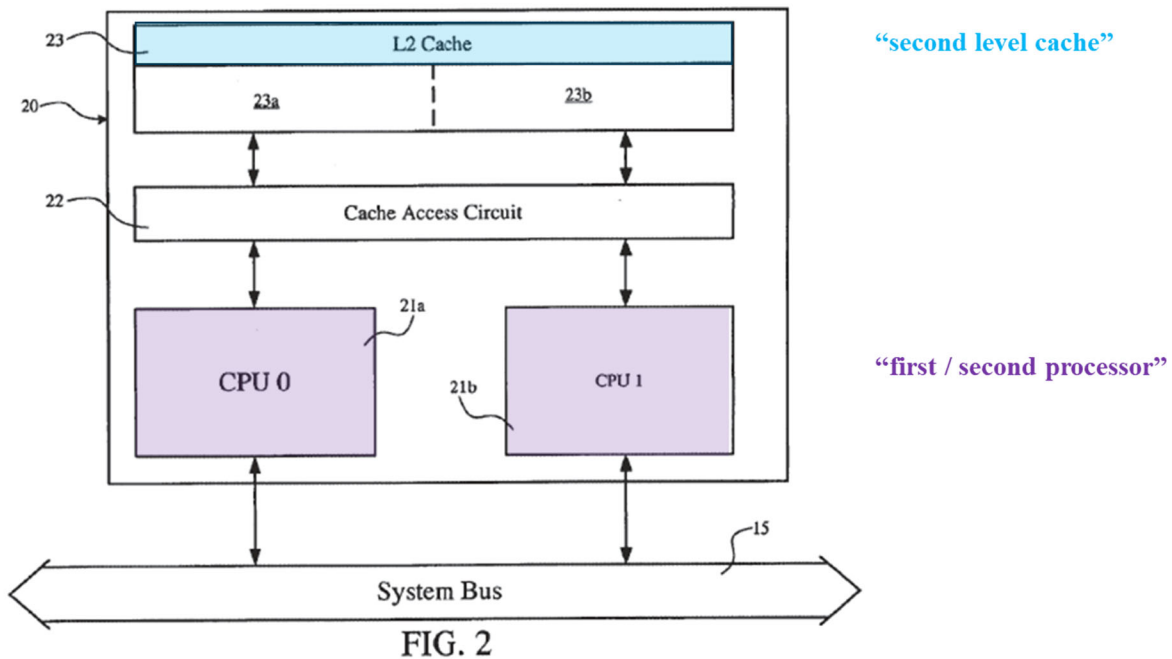


FIG. 4

Ex-1002, Fig. 4 (annotated). A POSITA would understand that the L1 cache memory in each CPU core constitutes “*a plurality of first level cache systems*,” which behaves like small hardware databases that can store and retrieve data, because Cherabuddi discloses “L2 cache” in Figure 2. Ex-1005, ¶¶ 25, 111, 114-17.



Ex-1002, Fig. 2 (annotated), Ex-1005, ¶ 116.

Cherabuddi teaches “wherein the first processor fetches instructions and accesses data from the first cache controller through a first first level cache system” L1 cache memory is utilized as a database to store and retrieve data and instructions (“accesses data”). Ex-1005, ¶¶ 72. Here, CPU 21a (“the first processor”) retrieves instructions and accesses data (“fetches instructions and access data”) utilizing cache controller 42a (“from the first cache controller”), from the L1 cache memory (“through a first first level cache system”). Ex-1002, 5:2-5 (“Each cache controller 42 . . . transfers address and data between its associated CPU core 41 and cache access circuit 22.”); Ex-1002, ¶ 117.

Cherabuddi also teaches “wherein the second processor fetches instructions and accesses data from the second cache controller through a second first level

cache system” via the CPU 21b (“*the second processor*”), cache controller 42b (“*the second cache controller*”), and L1 cache memory within CPU core 41b (“*a second first level cache system*”), for the same reasons in the preceding paragraph. Ex-1005, ¶ 117.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 118.

3. Claim 3

- i. ***[3.0] The processor integrated circuit of claim 1, wherein the cache memory blocks further comprise cache tag memory.***

As discussed in limitation [1.3], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*the cache memory blocks.*” Ex-1005, ¶¶ 93, 94 (citing Ex-1002, 4:66-5:14, 5:15-49). Figure 4 depicts “cache memory partitions 23a-23b,” which include “two data RAM arrays 51-52 **having corresponding searchable tag arrays 61-62**, respectively, while cache memory partition 23b includes two data RAM arrays 53-54 **having corresponding searchable tag arrays 63-64**, respectively.” Ex-1002, 5:15-19.

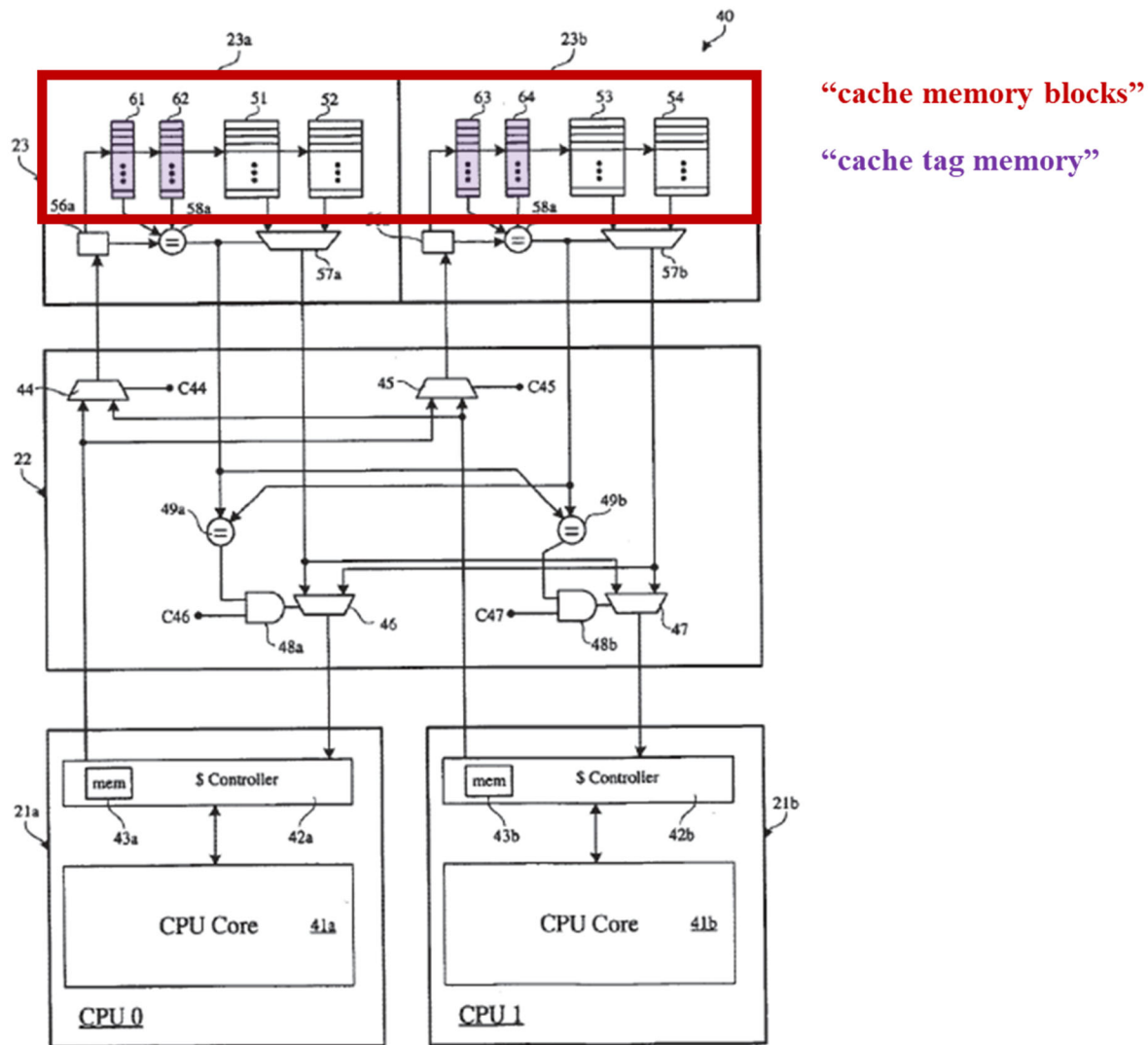


FIG. 4

Ex-1002, Fig. 4 (annotated); Ex-1005, ¶ 122 Cherabuddi explains that “[t]ag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively” (“*cache tag memory*”). Ex-1002, 5:39-41; Ex-1005, ¶ 124.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 125.

B. Ground 2: Claims 4-5 and 8-10 Are Obvious Based on Cherabuddi and Dean

1. Motivation to Combine

Claim elements [4.0] and [5.4] would have been obvious based on the teachings of Cherabuddi and Dean. Ex-1005, ¶¶ 213, 218 In particular, the teachings of Dean incorporated into the teachings of Cherabuddi would have rendered obvious “*wherein each cache controller is provided with cache hit rate monitoring apparatus*” of claim [4.0] and “*monitoring past cache performance associated with processors and partitions*” of claim [5.4].

To start, both Cherabuddi and Dean are analogous art to the ’264 Patent. Cherabuddi and Dean are both directed to a similar field of endeavor of the ’264 Patent—dynamically allocating cache memory in a multi-processor system. Ex-1001, (54), Abstract; Ex-1002, (54) Abstract, [0002]; Ex-1003, Abstract; Ex-1005, ¶ 189.

Cherabuddi and Dean are also both reasonably pertinent to a problem described by the ’264 Patent—“provid[ing] sufficient cache to support high hit rates for all or most threads if maximum processor performance is to be attained.” Ex-1001, 2:29-32. Cherabuddi recognizes this problem. Ex-1002, 1:23-25 (“Conversely, when there is a cache miss, the requested data must be retrieved from main memory, and thus CPU performance approaches the speed of main memory. Thus, increased performance may be achieved by maximizing the percentage of

cache hits during operation.”). And so does Dean. Ex-1003, 1:20-22 (“Accordingly, improvement in memory access times has been one of the major areas of research for increasing the speed of the computer.”); Ex-1005, ¶ 189.

Cherabuddi explains that when a CPU requests data from cache memory, this is a “cache hit,” and CPU performance operates at the faster speed of cache memory, but when requested data is located in main memory, and not cache memory, this is referred to as a “cache miss,” and CPU performance operates at the slower speed of main memory. Ex-1002, 1:20-25. Cherabuddi’s disclosure is consistent with the disclosure of the ’264 Patent that “[m]emory references that are found in cache are called ‘hits’ in the cache, while references not found in cache are called cache ‘misses.’” Ex-1001, 2:14-16. A POSITA would be motivated to maximize cache hits and minimize cache misses so that CPU performance can operate at the faster speed of cache memory. Ex-1005, ¶ 31.

Cherabuddi also suggests partitioning cache memory dynamically allocated between two or more processors on an MPU based on “a desired system configuration or to the processing needs of the processors.” Ex-1002, 1:61-66. In particular, a cache memory is partitioned between two or more processors and treats each cache memory partition as a dedicated cache memory. Ex-1002, 2:8-12. In this scenario, the entire cache memory can dynamically allocate cache resources between multiple processors to maximize cache resources while allowing each

processor and corresponding allocated cache memory partition to operate independently. Ex-1002, 2:39-44. Cherabuddi's invention also allows for any processor to access the entire cache memory in case another processor is found to be defective, maximizing cache resources. Ex-1002, 2:44-49; Ex-1005, ¶ 191.

A POSITA would be motivated to modify the teachings of Cherabuddi to include Dean's teachings of utilizing a cache miss percentage calculated with the expected or desired cache hit ratio for several reasons. Cherabuddi observes that “increased performance may be achieved by maximizing the percentage of cache hits during operation.” Ex-1002, 1:26-27. Dean teaches that “[t]he system and method preferably determine[] which process could benefit the most if additional cache space was allocated to the process or could benefit a computer system the most if additional cache space was allocated to the process.” Ex-1003, 3:58-62. Dean also teaches that “[t]he reallocation can take place periodically, thereby maximizing the performance of the system and handling unforeseen demands by processes.” Ex-1003, 3:64-66. Dean explains that “a relational cache miss percentage is used to reallocate the ways, but other system metrics may also be used.” Ex-1003, 3:9-11. Accordingly, a POSITA would be motivated to improve system performance by dynamically reallocating cache memory as taught in Cherabuddi by monitoring cache hit rates as taught in Dean. Ex-1005, ¶ 191.

In that same vein, Cherabuddi also describes the benefit of dynamically allocating cache resources between multiple processors, which maximizes cache resources, similarly increasing performance. Ex-1002, 2:40-46 (“The ability to dynamically allocate cache resources between multiple processors advantageously allows the entire cache memory to be used, irrespective of whether one or both processors are currently active, thereby maximizing cache resources while allowing for both one-thread and two-thread execution modes. In addition, the present invention may be used to maximize cache resources when one of the on-board processors is defective.”). Likewise, Dean observes that computer system innovation is motivated by the demand for consumers to have faster and more powerful computers. Ex-1003, 1:11-13. A contributing inhibitor is “the speed by which data can be accessed from memory.” Ex-1003, 1:14-17. Dean explains that a multiprocessor may be delayed in processing speed by accessing data from a slow main memory, but cache memory (a small amount of very fast and expensive memory that is used to store a copy of frequently accessed information) can be combined with main memory to increase memory access while remaining a relatively lower cost. Ex-1003, 1:27-30. A POSITA would recognize that the cache miss percentage, monitoring the expected or desired cache hit ratio, as taught by Dean, would be another metric used in dynamically apportioning cache partitions among the processors in Cherabuddi. Ex-1005, ¶ 191.

A POSITA would have been motivated to combine utilizing a cache miss percentage calculated with the expected or desired cache hit ratio in Dean to dynamically allocate cache resources between multiple processors to maximize cache resources in Cherabuddi. Ex-1005, ¶ 200. A POSITA would have reasonably expected to succeed because the multi-processor system of Cherabuddi teaches that cache memory is “dynamically allocated” **“in response to specific needs of associated CPUs 21a-21b”** to **“maximize cache performance** by ensuring that both cache memory partitions 23a-23b are utilized” (Ex-1002, 4:38-43), and cache memory in the system of Dean is allocated “to the different processes” and “then dynamically reallocated **based on performance,**” most preferably, using “a relational cache miss percentage, but other system metrics may also be used” (Ex-1003, 3:6-11); Ex-1005, ¶ 201. Such processes can be **“processors,** tasks, or threads” that “can access the cache during any cycle.” Ex-1003, 3:4-6. The cache reallocation in Dean occurs to maximize performance based on the needs of the system. Ex-1003, 2:66-3:1 (“The preferred embodiments of the present invention provide a performance based system and method for dynamic allocation of a unified multiport cache.”); Ex-1003, 10:35-40 (“Preferably the process that has its ways assigned to process A would be the process that has the lowest cache miss percentage. **This should provide the best increase in performance,** as after

reallocation process A should have a smaller cache miss percentage and the other process should have a slightly higher cache miss percentage.”); Ex-1005, ¶ 197.

For the above reasons, it would have been obvious to a POSITA to include a known element (Dean’s teachings relating to a cache miss percentage based on the expected or desired cache hit ratio) into another mechanism (Cherabuddi’s teachings of a partitionable cache) that yields predictable results (dynamically partitioning a cache based on Dean’s miss-rate metrics). Ex-1005, ¶ 200. Similarly, for the above reasons, this combination would have been obvious to a POSITA because it is nothing more than the arrangement of known element performing their same function (Cherabuddi’s partitionable cache and Dean’s expected or desired cache hit ratio) with predictable results. Ex-1005, ¶ 200. Finally, for the above reasons, a POSITA would have found it obvious to try any metric for measuring cache performance (such as Dean’s expected or desired cache hit ratio) as a gauge allocating Cherabuddi’s partitionable cache between the first and second cache controllers. Ex-1005, ¶ 200.

2. Claim 4

- i. ***[4.0] The processor integrated circuit of claim 1, wherein each cache controller is provided with cache hit rate monitoring apparatus.***

A POSITA would have been motivated to combine Cherabuddi and Dean for the reasons described above. Section.VII.B.1. Also, during prosecution, the

Examiner already found Dean to suggest the additional element added by claim 4, and Patentee did not dispute that. Ex-1007, 294.

As discussed in limitation [1.0], Cherabuddi teaches “*a processor integrated circuit.*”

It would have been obvious to include Dean’s teachings relating to a relational cache miss percentage based on the expected or desired cache hit ratio in the system of Cherabuddi. Ex-1005, ¶ 210. Cherabuddi already teaches partitioning cache memory dynamically allocated between two or more processors on an MPU based on “a desired system configuration or to the processing needs of the processors.” Ex-1002, 1:61-66. Cherabuddi explains CPUs 21a-21b a cache controller 42 (“*wherein each cache controller*”). Ex-1002, 4:66-5:2. As discussed for limitation [1.4], CPUs 21a and 21b, via cache controllers 42a and 42b can selectively provide data and information to and receive data and information from the cache lines of data arrays 51-54 within cache memory partitions 23a and 23b. Cherabuddi explains that “[d]ata arrays 51-54 each include a plurality of cache lines for storing data retrieved from main memory 12,” and “[t]ag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively.” Ex-1002, 5:29-30, 5:39-40.

Dean suggests a functionally similar apparatus to Cherabuddi, including a multiport cache “that allows multiple single-cycle look ups through a multiport tag

and multiple single-cycle cache accesses from a multiport cache.” Ex-1003, 3:1-4. As a result, this multiport cache includes “multiple processes, **which could be processors**, tasks, or threads” that can access the cache during any cycle. Ex-1003, 3:4-6. Dean explains that “[t]ag 150 comprises a tag memory 600, a tag allocation register 151, and N hit/miss indications 190. The N hit/miss indications 190 are coupled to the cache 130 and serve to inform the cache or process of a hit. Tag allocation register 151 has space reserved to track system metrics 191, elapsed cycles 192, a cache evaluation period 193, a cache miss percentage 194, and a cache reallocation factor 195.” Ex-1003, 4:24-30. Dean also explains that cache miss percentage 194 is a preferred “factor used to determine if the cache should be reallocated.” Ex-1003, 4:48-53. As a result of the cache miss percentage 194 held in tag 150 of Dean, a POSITA would understand that in the combination of Dean and Cherabuddi, the cache-miss hardware would be held with the cache’s tag arrays, for example, Cherabuddi’s tag arrays 61-64. Ex-1005, ¶ 210. Dean explains that “[t]he cache miss percentage 195 can be calculated by multiplying the **expected or desired cache hit ratio** for each process by the cache evaluation period and adding up the results.” Ex-1003, 9:33-38. A POSITA would have understood that the cache miss rate is merely the inverse of the cache hit rate, and can be calculated as $\text{miss_rate} = 1 - \text{hit_rate}$. Ex-1005, ¶ 209. Accordingly, a POSITA would have realized that Cherabuddi’s cache controllers 43a-43b would have access to and

would be able to use the cache-miss hardware; each cache controller is provided with cache hit rate monitoring apparatus. (“wherein each cache controller is provided with cache hit rate monitoring apparatus”). Ex-1005, ¶ 210.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 213.

3. Claim 5

- i. [5.0] *A method of dynamically allocating cache on a multiple-processor integrated circuit, where the multiple processor integrated circuit is used in a partitionable multiple-processor system and comprises:*

Figure 1 shows a multi-processor system:

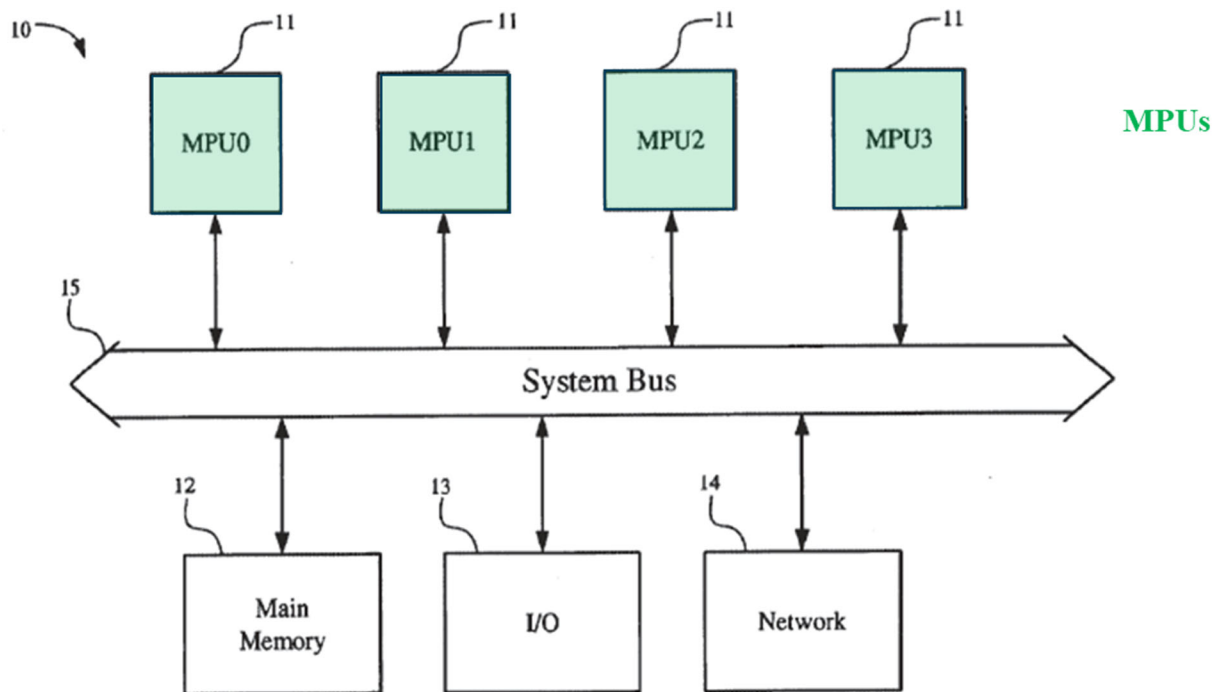
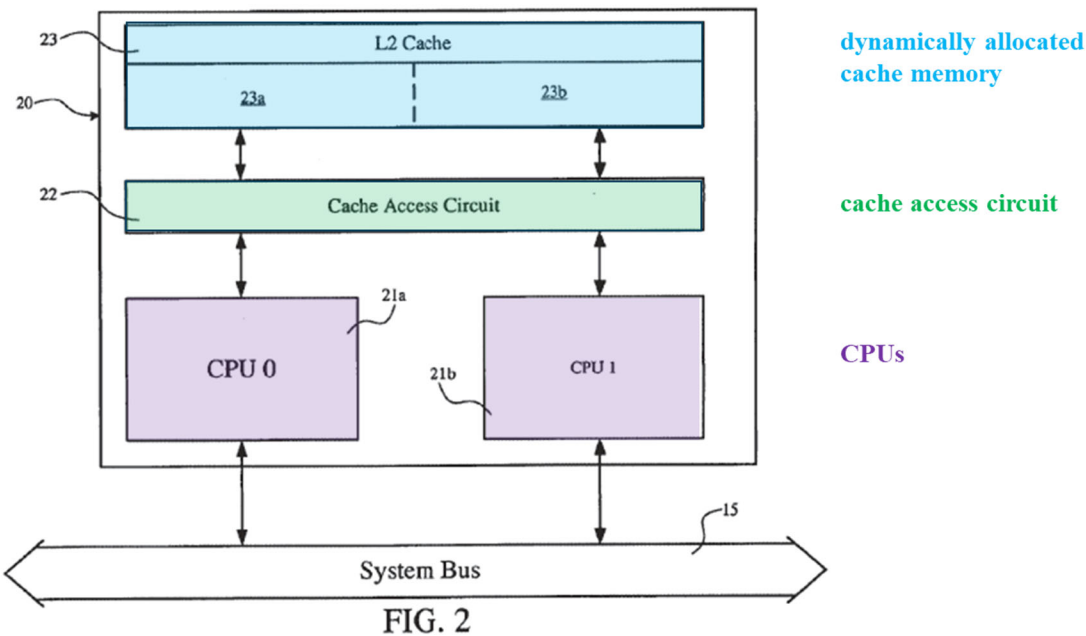


FIG. 1

Ex-1002, Fig. 1 (annotated); Ex-1005, ¶ 128. Figure 2 shows MPU 20 “that is one embodiment of MPU 11 of Fig. 1,” which includes “first and second processors such

as [CPUs] 21a-21b, a cache access circuit 22, and a dynamically allocated cache memory 23” (*“the multiple processor integrated circuit”*). Ex-1002, 3:26-30. Cherabuddi explains that “cache access circuit 22 allows the resources of cache memory 23 to be dynamically allocated between the first and second CPUs 21a-21b according to each CPU’s processing requirements to more efficiently utilize cache resources,” which demonstrates a *“the multiple processor integrated circuit”* being “used in a partitionable multiple processor system.” Ex-1002, 3:43-45; Ex-1005, ¶ 133.



Ex-1002, Fig. 2 (annotated).

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 128.

ii. ***[5.1] a plurality of processors each coupled to receive instructions from a first level cache associated therewith,***

MPU 40 contains CPUs 21a-21b (“*a plurality of processors*”). Ex-1002, 4:66-67. Cherabuddi explains that each CPU contains a CPU core, which has several well-known elements, including “**L1 cache memory, instruction units, fetch and decode units, execution units**, register files, writer cache(s), and so on.” Ex-1002, 5:11-13. A POSITA would understand that the L1 cache memory is a “*first level cache*,” which stores data and instructions for each CPU. Ex-1005, ¶¶ 72, 135. Figure 4 shows that each CPU (“*a plurality of processors*”) contains an independent CPU core, with an associated L1 cache memory therein, and each processor is therefore coupled to receive instructions from a “*first level cache*.” Ex-1005, ¶ 137. A POSITA would have known that processors receive instructions from their first-level cache to operate (“*a plurality of processors each coupled to receive instructions from a first level cache associated therewith*”). Ex-1005, ¶ 139.

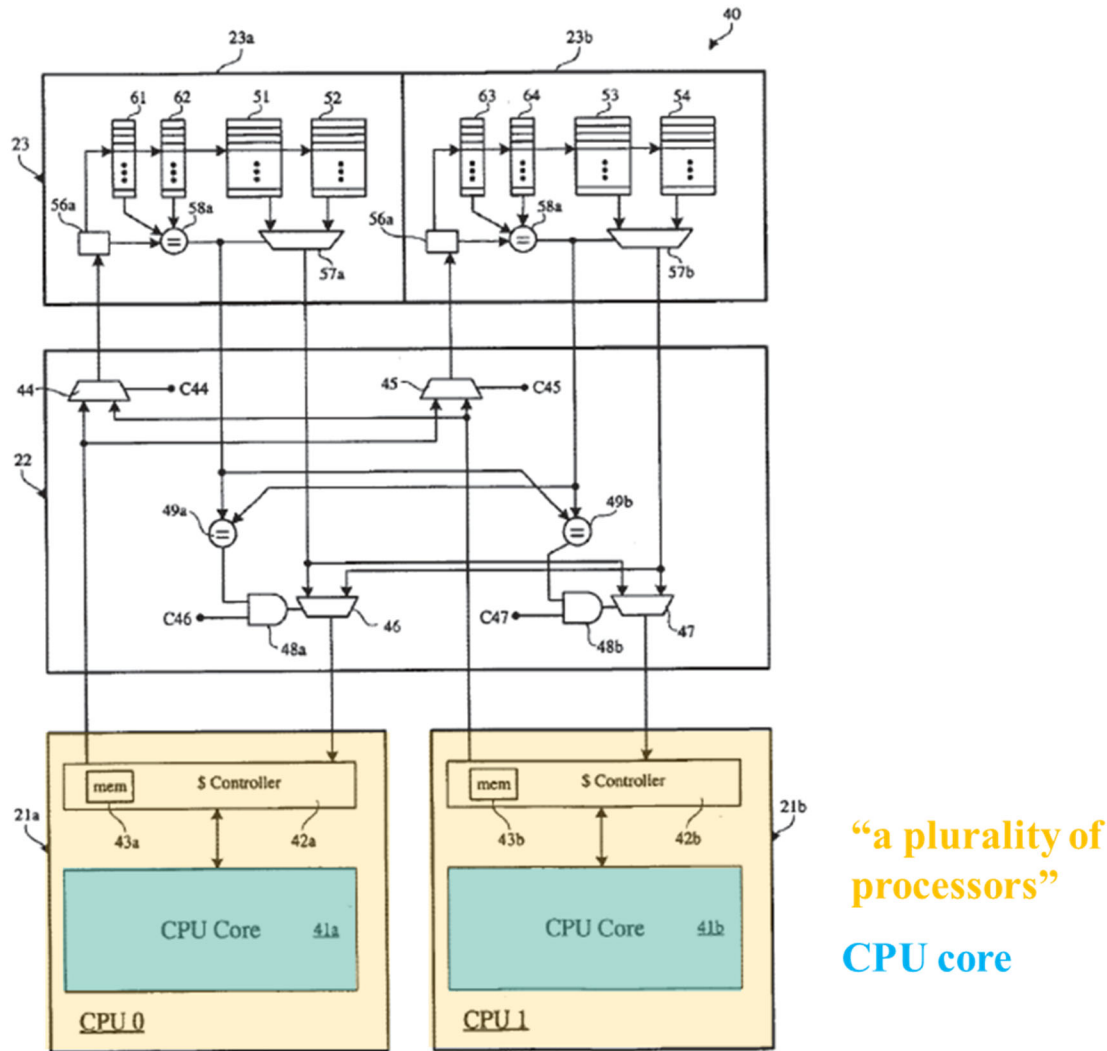


FIG. 4

Ex-1002, Fig. 4 (annotated).

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 135.

- iii. *[5.2] a plurality of allocable upper level cache memory blocks, interconnect apparatus for transmitting cache misses at each first level cache to upper level cache memory blocks assigned thereto, and*

Cherabuddi explains that each cache memory partition includes well-known address converters 56a-56b that convert a main memory address received from cache

access circuit 22 into a cache address that is used to concurrently address both the tag arrays 61-64 and data arrays 51-54.” Ex-1002, 5:19-28. “Data arrays 51-54 each include a plurality of cache lines for storing data received from main memory 12.” Ex-1002, 5:29-30, 31-32. “Tag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively.” Ex-1002, 5:39-41. The ’264 Patent describes that cache memory blocks 198 “incorporate[] data memory 200” and “tag memory 202.” Ex-1001, 4:37-39. Therefore, a POSITA would understand that data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*cache memory blocks*” as they include the entire tag/data set for each cache memory partition. Ex-1005, ¶ 141. Data arrays 51-54 and tag arrays 61-64 are a higher level cache memory and are therefore “*upper level cache memory blocks*” because L1 cache memory (“*first level cache*”) is initially checked for the desired cache block, and if it is not there, data arrays 51-54 and tag arrays 61-64 are checked for the desired data. Ex-1005, ¶¶ 140, 143. Further, cache access circuit 22 allows cache memory 23 (containing data arrays 51-54 and tag arrays 61-64) to be dynamically allocated between CPUs 21a-21b (“*a plurality of allocable upper level cache memory blocks*”). Ex-1002, 3:43-45; Ex-1005, ¶ 60.

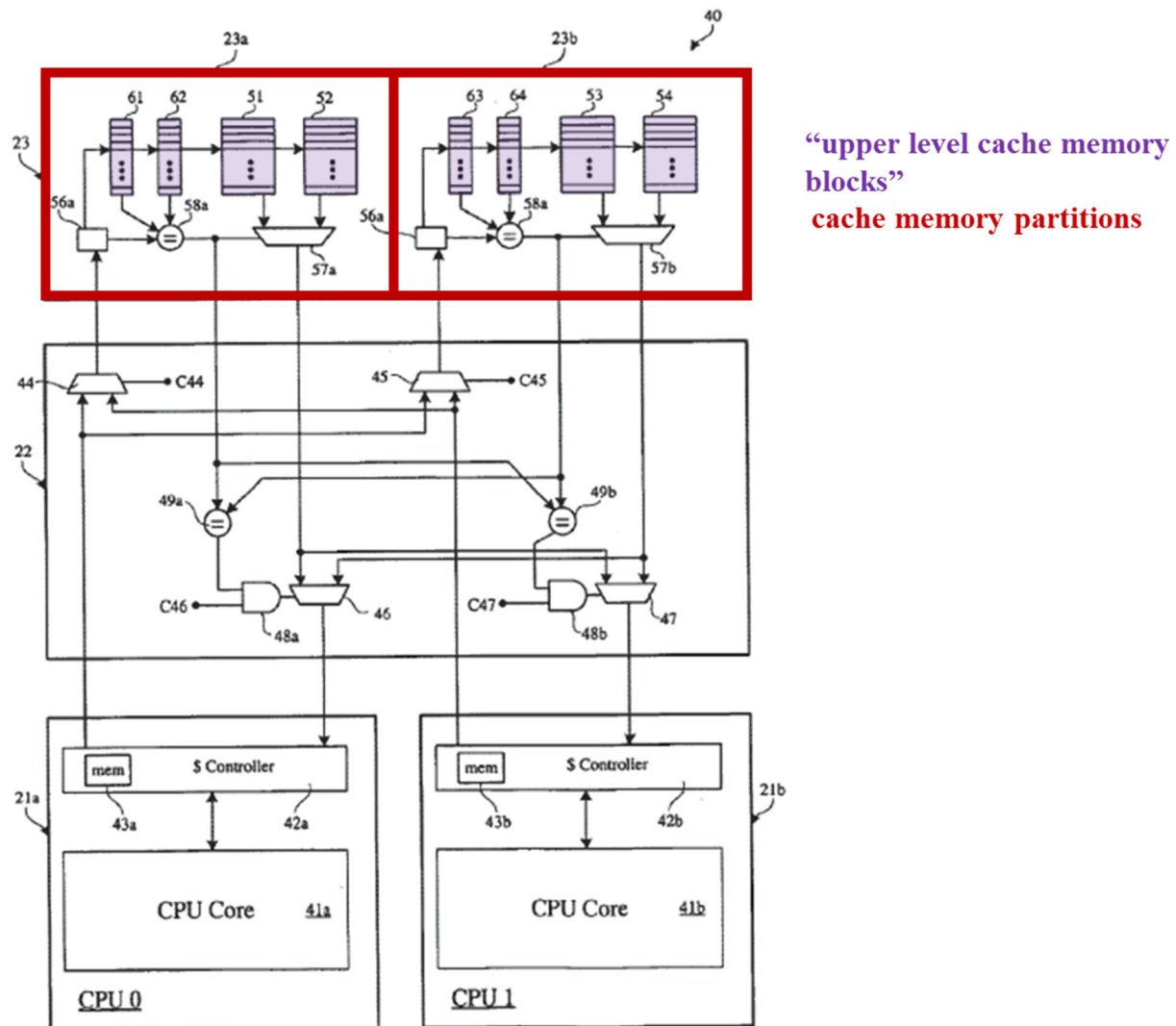


FIG. 4

Ex-1002, Fig. 4 (annotated).

The “*interconnect apparatus*,” is depicted in Figure 4, which is shown based on the wires from cache access controller 22, connecting the first cache controller 42a and the second cache controller 42b to the cache memory partitions 23a-23b.

Ex-1002, Fig. 4; Ex-1005, ¶ 145.

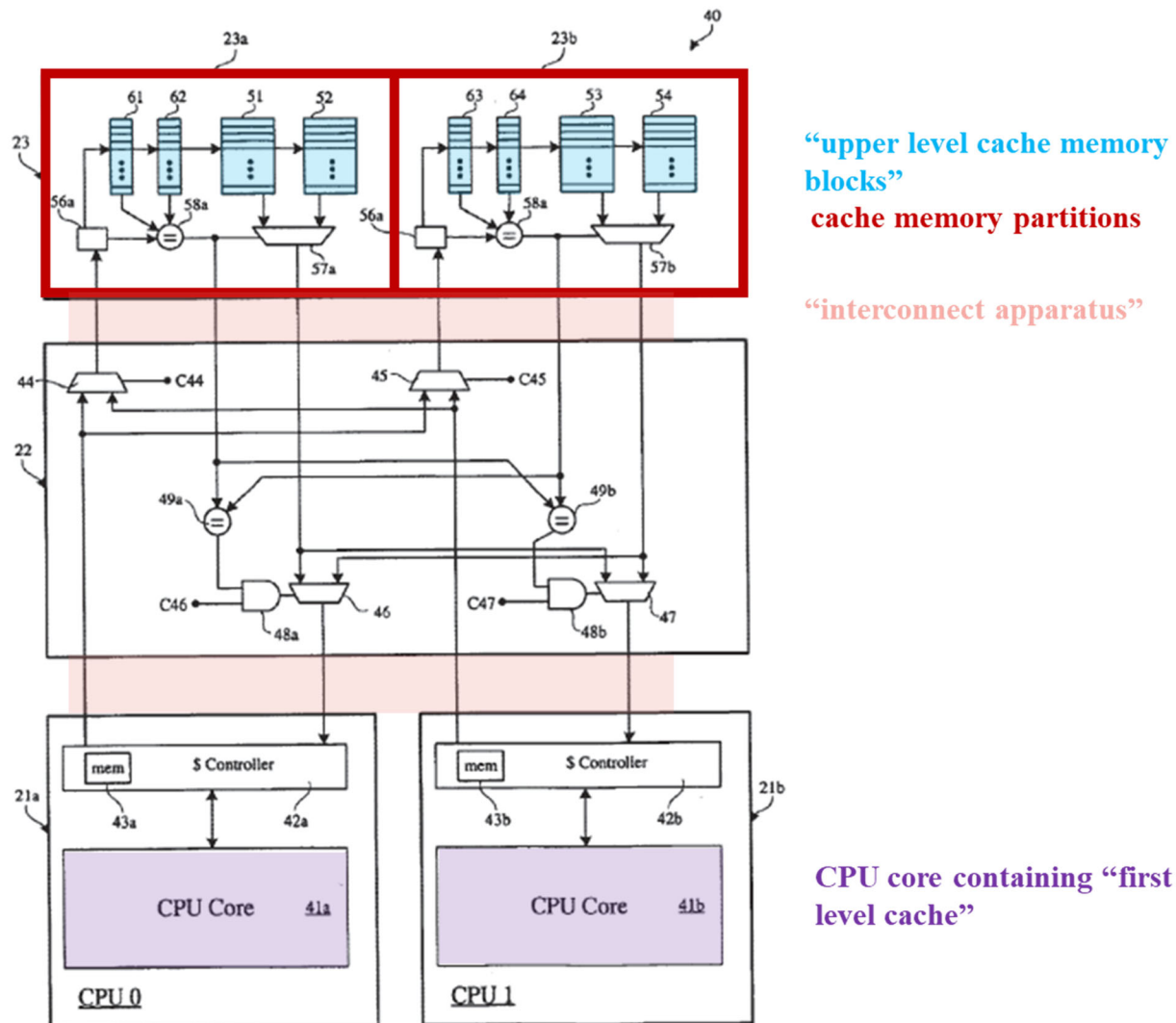


FIG. 4

Ex-1002, Fig. 4 (annotated).

Each CPU contains a CPU core, which itself has “L1 cache memory.”

Ex-1002, 5:11-13. A POSITA would understand that the L1 cache memory is a “*first level cache*.” Ex-1005, ¶¶ 25, 144. Indeed, Dr. Jacob explains that a cache stores and retrieves data in the form of a cache block (some number of contiguous bytes from memory) tagged with the memory address of the data. Ex-1005, ¶¶ 25, 144.

The address is used to search within the cache to determine whether the desired block is located within the cache, and if the values match, the desired block is located within the cache, referred to as a cache hit, but if the values do not match, the desired block is not located within the cache, referred to as a cache miss (“*cache misses*”). Ex-1005, ¶¶ 26, 144; Ex-1002, 1:20-25 (“When data requested by the CPUs is in the cache memory, there is a cache hit . . . [W]hen there is a cache miss, the requested data must be retrieved from main memory.”). In Cherabuddi, a POSITA would recognize that the L1 cache memory of each CPU core (“*each first level cache*”) would be initially checked for the desired block, and if it is not there (resulting in a cache miss), the cache miss would be communicated by each cache controller (via the wires connected to the cache access circuit) (“*interconnect apparatus for transmitting cache misses at each first level cache to . . .*”) to data arrays 51-54 and tag arrays 61-64 (“*. . . upper level cache memory blocks assigned thereto*”). Ex-1005, ¶¶ 29, 147.

Cherabuddi explains how “*upper level cache memory blocks*” are assigned to “*each first level cache.*” Each cache controller “transfers address and data between its associated CPU core 41 and cache access circuit 22.” Ex-1002, 5:2-6. As discussed for limitation [1.4], CPUs 21a and 21b, via cache controllers 42 and 42b can selectively provide data and information to and receive data and information from data arrays 51-54 and tag arrays 61-64 within cache memory partitions 23a and

23b. *See* § VIII.A.1.iv.b. For example, Cherabuddi explains that while MPU 20 is in the 2T state, CPU 21a requests data from cache memory partition 23a—particularly, the cache lines of data arrays 51-52—by providing a main memory address to address converter 56a via MUX 44. Ex-1002, 6:42-44; *see also* Ex-1002, 6:60-61 (“CPU 21b may simultaneously request data from cache memory partition 23b in a similar manner.”). Cherabuddi teaches that “[a]ddress converter 56a converts the main memory address to a cache address that includes a tag address and a cache address.” Ex-1002, 6:44-46. Cherabuddi further teaches that “[t]he cache index is used to select a cache line in data arrays 51-52 and associated tag arrays 61-62.” Ex-1002, 6:46-48. “If there is data stored at the selected cache line in data arrays 51 and/or 52, the data is read out to MUX 57a.” Ex-1002, 6:48-49. Here, in the 2T state, data arrays 51-52 are accessible only by CPU 21a, containing L1 cache memory within CPU core 41a. Ex-1002, 6:42-44 (“*interconnect apparatus for transmitting cache misses at each first level cache to upper level cache memory blocks assigned thereto*”); Ex-1005, ¶ 147. Similarly, data arrays 53-54 are accessible only by CPU 21b, containing L1 cache memory within CPU core 41a. Ex-1002, 6:60-61 (“CPU 21b may simultaneously request data from cache memory partition 23b in a similar manner.”); Ex-1005, ¶ 147.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 145.

iv. ***[5.3] allocation apparatus for assigning upper level cache memory blocks to processors;***

Cherabuddi teaches a cache access circuit 22, which “couples the first and/or second CPUs 21-a-21b to the first and/or second cache memory partitions 23a-23b.” Ex-1002, 3:40-42. As discussed for limitation [5.2], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively (located within cache memory 23), constitute “*upper level cache memory blocks.*” Ex-1005, ¶¶ 141, 147. Cherabuddi teaches that cache access circuit 22 (“*allocation apparatus*”) “selectively couples” CPU 21a-21b to cache memory partitions 23a-23b. Ex-1002, 3:40-42. Cache access circuit 22 (“*allocation apparatus*”) dynamically allocates cache memory 23 (data arrays 51-54 and tag arrays 51-54 (“*upper level cache memory blocks*”) between CPUs 21a-21b (“*processors*”), and is therefore responsible for “*assigning upper level cache memory blocks to processors.*” Ex-1002, 3:39-45; Ex-1005, ¶ 150.

Dynamic allocation, as described in Cherabuddi, is accomplished by “MUXes 44-45,” which “selectively provide address information from CPUs 21a-21b to cache memory partitions 23a-23b, respectively, and MUXes 46-47 selectively provide data from cache memory partitions 23a-23b to CPUs 21a-21b, respectively.” Ex-1002, 5:56-61. Cherabuddi further explains how data and addresses are routed from CPUs to cache memory partitions: “Specifically, MUX 44 selectively provides address and data information to cache memory partition 23a from either CPU 21a or

CPU 21b in response to C44, and MUX 45 selectively provides address and data information to cache memory partition 23b from either CPU 21a or CPU 21b in response to C45.” Ex-1002, 6:9-14. Cherabuddi also explains how data and addresses are returned from cache memory partitions to CPUs: “MUX 46 selectively returns data to CPU 21a from either cache memory partition 23a or 23b in response to AND gate 48a, and MUX 47 selectively returns data to CPU 21b from either cache memory partition 23a or 23b in response to AND gate 48b.” Ex-1002, 6:14-18.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 148.

v. ***[5.4] the method comprising the steps of: monitoring past cache performance associated with processors and partitions;***

A POSITA would have been motivated to combine Cherabuddi and Dean for the reasons described above. Section. VII.B.I. Also, during prosecution the Examiner already found Dean to suggest the additional element added by limitation [5.4], and Patentee did not dispute that. Ex-1007, 297.

It would have been obvious to a POSITA to include Dean’s teachings relating to tracking cache miss percentage over a period of time and then determining whether the cache should be reallocated in the system of Cherabuddi for the reasons explained below. In particular, Cherabuddi already teaches that “increased performance may be achieved by maximizing the percentage of cache hits during operation.” Ex-1002, 1:26-27. Cherabuddi describes “dynamically allocating

resources of cache memory 23 **in response to specific needs of associated CPUs 21a-21b.**” Ex-1002, 4:38-39. This dynamic allocation may be done by determining the tasks of computer programs that may be allocated among the processors. Ex-1002, 4:4-7, 4:13-17. For example, in the 2T state (M=1), the kernel sets both CPUs 21a-21b to the active state, and assigns an execution thread to each CPU. Ex-1002, 4:17-20. Then, “if the kernel determines that only one of CPUs 21a-21b is necessary for a particular instruction code sequence, the kernel may transition MPU 20 to the 1T state (M=0), flushing the second cache memory partition 23b and then assigning execution of the instruction code sequence to the active CPU 21a.” Ex-1002, 4:31-36.

A POSITA would recognize that tracking cache miss percentage over a period of time and then determining whether the cache should be reallocated, in Dean, would merely be an additional step utilized in dynamically allocating cache resources between multiple processors in Cherabuddi. Dean explains that the tag allocation register tracks system metrics, including “elapsed cycles 192, a cache evaluation period 193, a cache miss percentage 194, and a cache reallocation factor 195.” Ex-1003, 4:27-34. The tag allocation register also tracks other system metrics, including “overall system performance.” Ex-1003, 4:38-40. Prior overall system performance is monitored over a period of time as “[e]lapsed cycles 192 is used to determine the number of cycles that have elapsed since the last evaluation system

performance.” Ex-1003, 4:40-42. Dean also discusses the cache evaluation period to determine “when the system’s performance should be re-evaluated.” Ex-1003, 4:42-44. Dean explains that “[i]f the elapsed cycles 192 is larger than (or equal to) cache evaluation period 193, then the system performance should be evaluated and the ways of the cache reassigned to the processes.” Ex-1003, 4:44-47. Also, “[t]ag allocation controller 161 evaluates system 100’s performance and then reallocates ways of the cache to the processes that would be best suited to have more cache memory. Most preferably, this is performed through reference to the misses per process, with the process having the highest number of misses receiving the most allocation of the cache. Ex-1003, 7:58-64; Ex-1005, ¶ 220.

Past cache performance is also monitored via the predetermined cache reallocation factor, which “is a number that indicates how much greater the misses for one process must be versus another process for the method to allocate cache ways to the first process.” Ex-1003, 10:4-7. Dean also tracks the total number of cache misses for each processor. Ex-1003, 4:34-38 (“Each variable would contain the number of misses for each process. For example, if process 110 missed 100 times, then ‘100’ would be stored in one of the variables in system metrics 101.”). The total misses for all processes are compared to the cache miss percentage (a factor used to determine if the cache should be allocated) to determine if there are too many misses. Ex-1003, 4:48-53. A POSITA would understand that Dean uses a cache-

miss percentage to monitor past cache performance associated with the processors and partitions, and uses this information to allocate storage in the cache to the different processors (*“monitoring past cache performance associated with processors and partitions”*). Ex-1005, ¶¶ 221, 223, 226.

Accordingly, a POSITA would have realized that Cherabuddi would have monitored cache miss percentage over a period of time of CPUs 21a-21b and cache memory partitions 23a-23b as an additional element in dynamically allocating cache memory 23 between CPUs 21a-21b according to each CPU’s processing requirements. Ex-1002, 3:42-46; Ex-1005, ¶ 219.

Accordingly, Cherabuddi and Dean teach this limitation. Ex-1005, ¶ 218.

vi. [5.5] determining desired processor to partition and upper level cache block allocations to processors; and

As discussed for limitation [5.2], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively (located within cache memory 23), constitute *“upper level cache memory blocks.”* Ex-1005, ¶¶ 141, 147. Cherabuddi explains that “[c]ache access circuit 22 selectively couples the first and/or second CPUs 21a-21b to the first and/or second cache memory partitions 23a-23b.” Ex-1002, 3:40-42. Cherabuddi also explains that “cache access circuit 22 allows the resources of cache memory 23 [(containing data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively)] to be dynamically allocated between the first and second CPUs 21a-21b according to each CPU’s processing requirements

to more efficiently utilize cache resources” (“*upper level cache block allocations*”) Ex-1002, 3:42-45. In the 2T state, “during execution of the computer program(s),” cache access circuit couples the first CPU 21a to the first cache memory partition 23a for exclusive use and couples the second CPU 21b to the second cache memory partition 23b for exclusive use. Ex-1002, 4:13-14, 4:23-27. “If the kernel determines that only one of CPUs 21a-21b is necessary for a particular instruction code sequence,” MPU 20 is transitioned to the 1T state, and “both cache memory partitions 23a-23b are allocated to the active CPU.” Ex-1002, 4:38-44. In other words, when only one CPU is activated, both cache partitions are allocated to that CPU for its use, and the decision to use two cores splits the cache in two such that the upper level cache blocks are allocated to both processors (“*determining desired processor to partition and upper level cache block allocations to processors*”). Ex-1005, ¶ 154.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 151.

- vii. ***[5.6] repartitioning the system, the step of repartitioning the system including allocation of upper level cache blocks to processors of at least one of the multiple processor integrated circuits,***

Cherabuddi teaches that CPUs operate in two modes where the kernel assigns tasks between CPUs of each MPU. Ex-1002, 3:62-63. In the 1T state (M=0), the kernel assigns tasks to only one of the CPUs (21a or 21b) while the other CPU remains idle. Ex-1002, 3:63-67. Here, for example, the cache access circuit 22

couples the active CPU 21a to both the first and second cache memory partitions 23a-23b to use all resources of cache memory 23. Ex-1002, 4:7-10. While in the 1T state, the kernel can repartition and allocate cache resources to a single CPU “according to each CPU’s processing requirements to more efficiently utilize cache resources” (“*repartitioning the system, the step of repartitioning the system*”). Ex-1002, 3:45-46; Ex-1005, ¶ 155.

As discussed for limitation [5.2], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively (located within cache memory 23), constitute “*upper level cache blocks.*” Ex-1005, ¶¶ 141, 147. Cherabuddi also teaches transitioning between the 1T and 2T states. Ex-1002, 3:61-62. If the kernel determines that certain tasks may be executed in parallel, the kernel may transition MPU 20 to the 2T state (M=1). Ex-1002, 4:13-17. Cherabuddi explains that cache memory 23 is dynamically allocated based on the needs of CPUs 21a-21b to maximize cache performance and ensure that both cache memory partitions 23a-23b are utilized. Ex-1002, 4:38-42. Cherabuddi also explains that in the 1T state, both cache memory partitions 23a–23b are allocated to the active CPU, and in the 2T State, each cache memory partition 23a and 23b is allocated only to its corresponding CPUs 21a and 21b, respectively” (“*allocation of upper level cache blocks to processors*”). Ex-1002, 4:43-47; Ex-1005, ¶ 155. As discussed in limitation [5.0], CPUs 21a and 21b are a part of the same “*multiple processor integrated circuits.*”

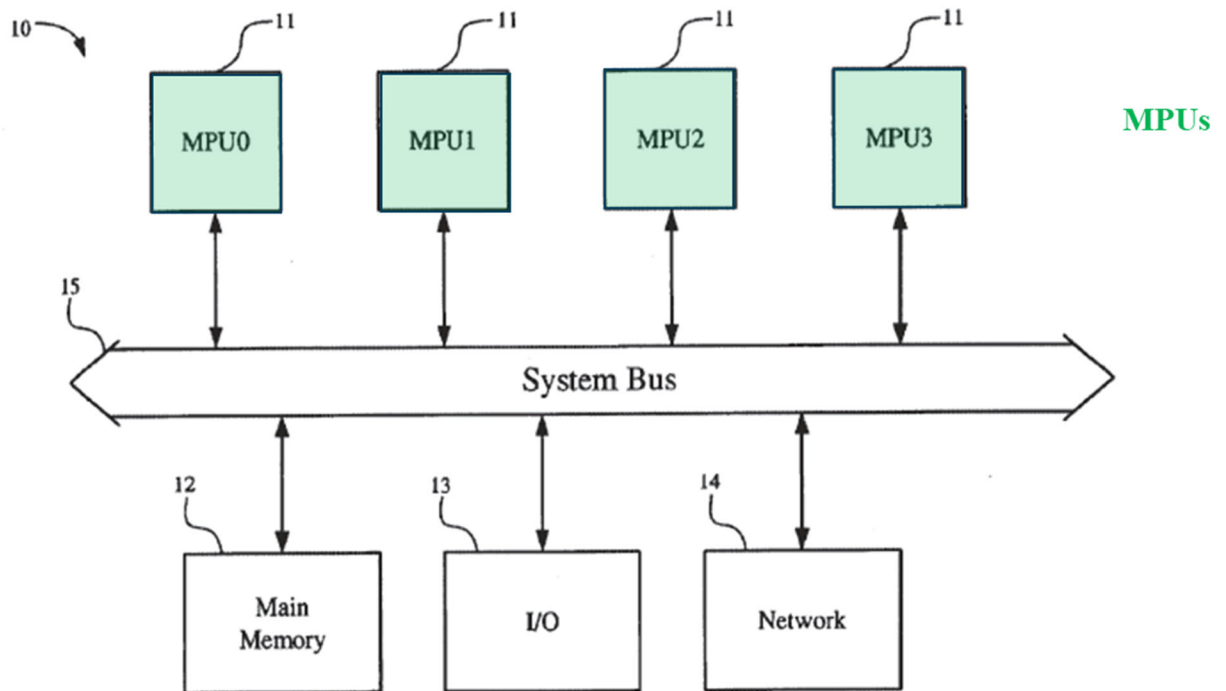


FIG. 1

Ex-1002, Fig. 1 (annotated). Figure 2 shows MPU 20 “that is one embodiment of MPU 11 of Fig. 1,” including CPUs 21a-21b:

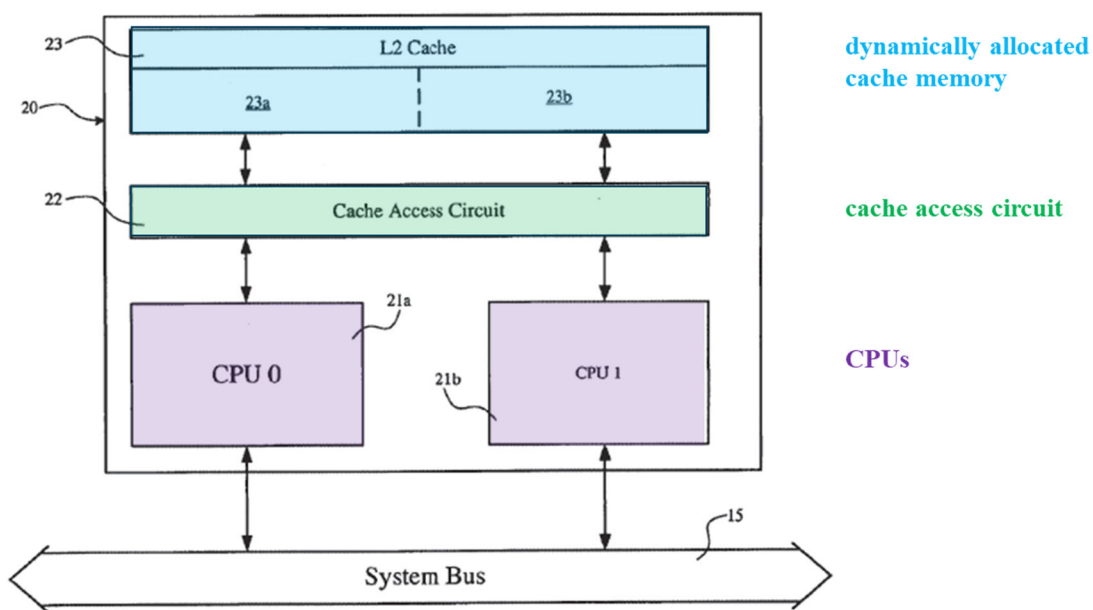


FIG. 2

Ex-1002, Fig. 2 (annotated).

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶¶ 155-56.

viii. [5.7] wherein the upper level cache blocks are usable to store data and instructions fetched from a random-access main memory.

Figure 1 depicts system 10, which includes “four MPUs 11 connected to each other and to a main memory 12, an input/output (I/O) device 13, and a network 14 via a system bus 15.” Ex-1002, 3:14-17.

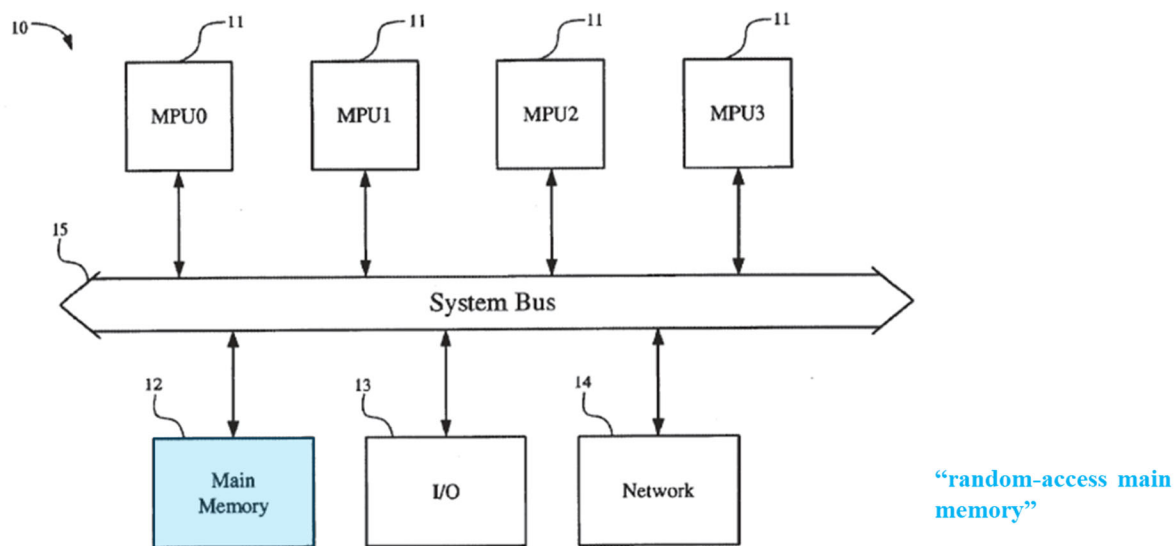


FIG. 1

Ex-1002, Fig.1 (annotated). Cherabuddi explains that main memory 12, depicted above, “is shared by MPUs 11, and may be any suitable random access memory (RAM) such as, for example, DRAM” (“random-access main memory”). Ex-1002, 3:19-20; Ex-1005, ¶ 158. Cherabuddi further explains that MPUs include cache memory to store data in anticipation of future use by the CPUs. Ex-1002, 1:16-17. As discussed for limitation [5.2], data arrays 51-52 and 53-54 coupled with tag

arrays 61-62 and 63-64, respectively, constitute “*upper level cache memory blocks.*” Ex-1005, ¶ 141. Indeed, data arrays 51-54 and tag arrays 61-64 store data and instructions fetched from another memory level, such as a main memory or another level of cache. Ex-1005, ¶ 161. “Tag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively” (Ex-1002, 5:39-41), and “[d]ata arrays 51-54 each include a plurality of cache lines for storing data retrieved from main memory 12” (Ex-1002, 5:29-30), which “may be any suitable random access memory (RAM) such as, for example, DRAM” (“*upper level cache memory blocks are usable to store data and instructions fetched from a random-access main memory*”) (Ex-1002, 3:17-18). Ex-1005, ¶ 159.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 162.

4. Claim 8

- i. ***[8.0] The method of claim 5, wherein the multiple processor integrated circuit further comprises a plurality of non-allocable cache memory blocks.***

As discussed *supra*, Cherabuddi teaches “*wherein the multiple processor integrated circuit*” of claim 5. As discussed for limitation [1.3], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*a plurality of . . . cache memory blocks.*” Ex-1005, ¶¶ 93, 94. When MPU 20 is in the 2T state (M=1), “[t]he cache access circuit 22 couples the first CPU 21a to the first

cache memory partition [containing data arrays 51-52 and tag arrays 61-62] for exclusive use thereto and couples the second CPU 21b to the second cache memory partition 23b [containing data arrays 53-54 and tag arrays 63-64] for exclusive access thereto.” Ex-1002, 4:23-27; Ex-1005, ¶ 170. In this state, the “*cache memory blocks*,” are assigned to one CPU and are non-allocable to the other CPU (the other CPU cannot access those blocks) (“*a plurality of non-allocable cache memory blocks*”). Ex-1005, ¶ 171.

To the extent one may assert that Cherabuddi does not teach this element, Cherabuddi also teaches the L1 cache memory, located within the CPU core of each CPU, is private to each processor and not shared between CPUs (“*a plurality of . . . cache memory blocks*”). Ex-1002, 5:10-14. A POSITA would understand that non-allocable means that each L1 cache memory is private to its CPU core and is therefore non-allocable to other CPUs. Ex-1005, ¶¶ 168-69.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 166.

5. Claim 9

- i. ***[9.0] The method of claim 5, wherein the interconnect apparatus further comprises a plurality of upper level cache controllers, and where each upper level cache controller is capable of controlling operation of the allocable cache memory blocks as a writeback cache.***

Figure 4 depicts MPU 20, including “CPUs 21a-21b, cache access circuit 22, and cache memory partitions 23a-23b.” Ex-1002, 4:66-5:1. Figure 4 depicts the

“*interconnect apparatus*” as the cache access circuit 22, directly connected via wires to cache controllers 42 (“*a plurality of upper level cache controllers*”) and cache memory 23. Ex.1005, ¶¶ 175-76; Ex.1002, 5:2-6 (“Each cache controller 42 . . . transfers address and data between its associate CPU core 41 and cache access circuit 22.” (“*the interconnect apparatus further comprises a plurality of upper level cache controllers*”).

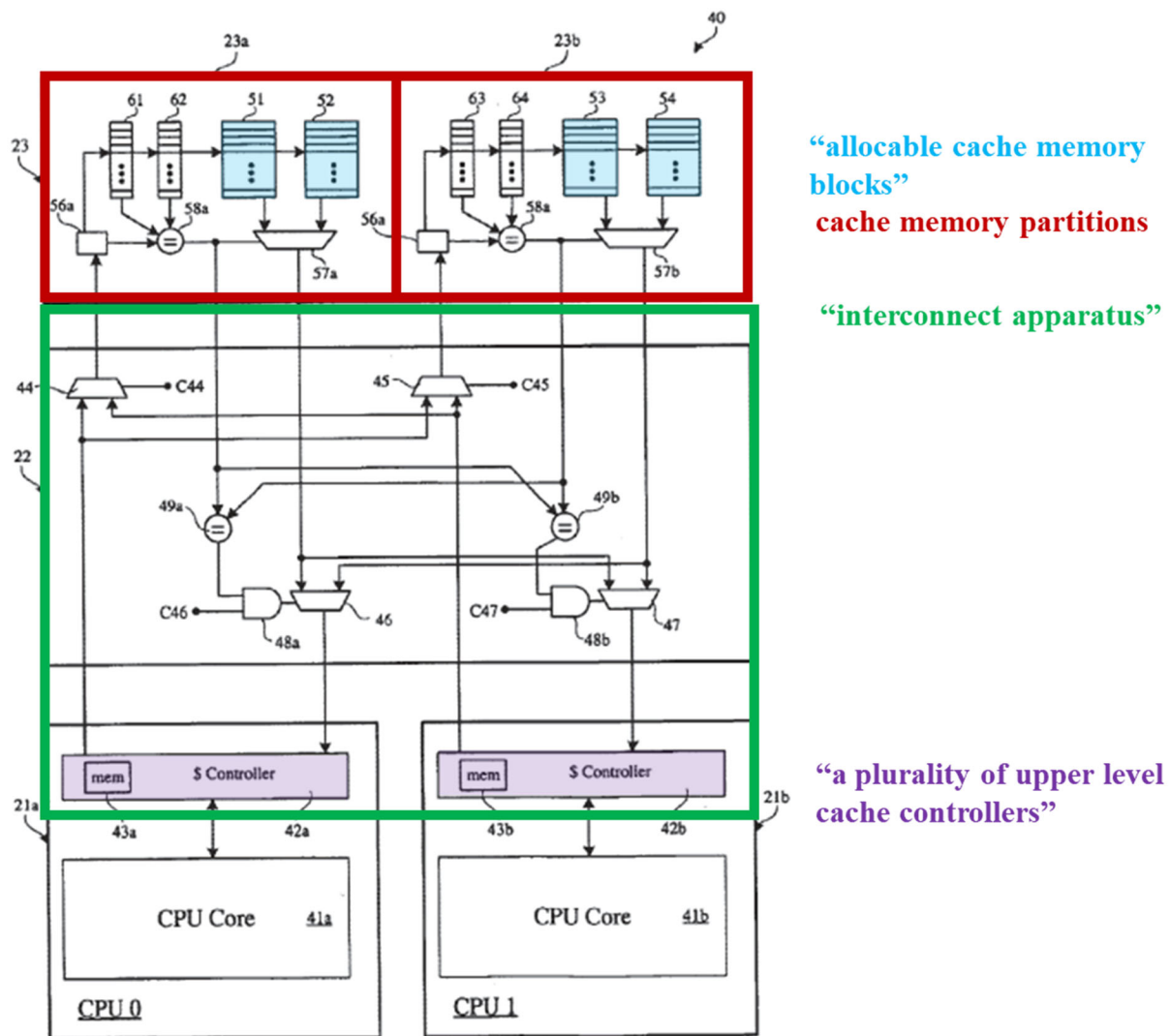


FIG. 4

Ex-1002, Fig. 4 (annotated).

As discussed for limitation [5.2], data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “*cache memory blocks*.” Ex-1005, ¶ 178. Further, when MPU is in the 1T state, both cache memory partitions 23a-23b (containing data arrays 51-54 and tag arrays 61-64) are allocated to the active CPU (“*allocable cache memory blocks*”). Ex-1002, 4:45-47. Cherabuddi also teaches that the kernel may transition MPU 20 to a 2T state (M=1) setting “both CPUs 21a-21b to the active state, and thereafter assign[ing] one execution thread to CPU 21a and another execution thread to CPU 21b in a well-known manner.” Ex-1002, 4:17-20. As a result of this configuration (M=1), “dirty data in cache memory partition 23b is written back to main memory 12 **using a well-known writeback operation**, thereby flushing cache memory partition 23b.” Ex-1002, 4:20-23. A POSITA would understand that “*a writeback cache*” would be a cache capable of performing a writeback operation. Ex-1005, ¶ 180. Indeed, a POSITA would understand each cache controller 42 (“*where each upper level cache controller*”), which “transfers address and data between its associated CPU core 41 and cache access circuit 22,” could also transfer dirty data from data arrays 51-54 and tag array 61-64 (“*is capable of controller operation of allocable cache memory blocks*”), located within cache memory partitions 23a-23b, to be written back to main memory (“*as a writeback cache*”). Ex.1002, 4:20-23, 5:2-6; Ex-1005, ¶ 180.

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 176.

6. Claim 10

- i. ***[10.0] The method of claim 5, wherein each upper level allocable cache block further comprises tag memory and cache data memory.***

As discussed in limitation [5.2], a POSITA would understand that data arrays 51-52 and 53-54 coupled with tag arrays 61-62 and 63-64, respectively, constitute “a plurality of allocable upper level cache memory,” which also teaches “each upper level allocable cache block” for the same reasons. Ex.-1005, ¶ 187. Further, Cherabuddi’s “[t]ag arrays 61-64 each include a plurality of lines for storing tag information for corresponding cache lines in data arrays 51-54, respectively.” Ex-1002, 5:39-41. And Cherabuddi’s “[d]ata arrays 51-54 each include a plurality of cache lines for storing data received from main memory 21.” Ex-1002, 5:29-30. Cherabuddi also teaches converters that convert a main memory address into a cache address that is used to concurrently address both the tag arrays and data arrays. Ex-1002, 5:19-28. From these teachings, a POSITA would have understood the tag arrays to be “tag memory” and the data arrays to be “cache data memory,” and, thus, have understood that “each upper level allocable cache block further comprises tag memory and cache data memory.” Ex-1002, 5:29-30; Ex-1005, ¶ 187.

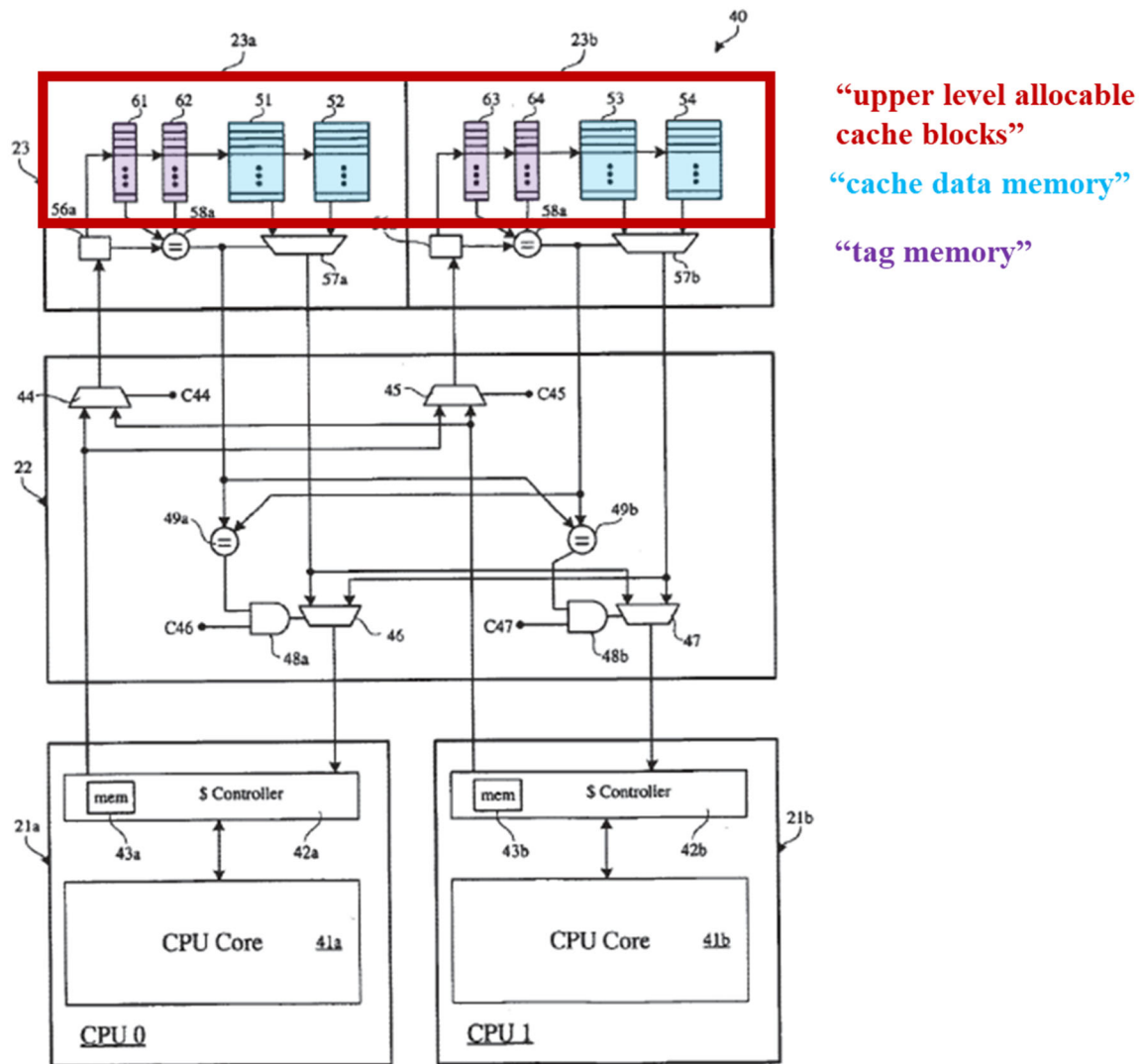


FIG. 4

Ex-1002, Fig. 4 (annotated);

Accordingly, Cherabuddi teaches this limitation. Ex-1005, ¶ 188.

C. Ground 3: Claim 11 Based on Cherabuddi and Noel**1. Claim 11**

- i. *[11.0] The method of claim 1, further comprising the steps of stopping execution of operating systems in each partition, and restarting execution of operating systems in each partition, and wherein the system is capable of being repartitioned without rebooting each operating system.*

Claim element [11.0] would have been obvious based on the teachings of Cherabuddi and Noel. Ex--1005, ¶ 260. In particular, Noel incorporated into the system of Cherabuddi teaches “*stopping execution of operating systems in each partition, and restarting execution of operating systems in each partition, and wherein the system is capable of being repartitioned without rebooting each operating system*” of claim [11.0].

To start, both Cherabuddi and Noel are analogous art to the '264 Patent for at least two reasons. First, Cherabuddi and Noel are both directed to a similar field of endeavor of the '264 Patent—dynamically allocating cache memory in a multi-processor system. Ex-1001, (54), Abstract; Ex-1002, (54) Abstract, 1:8-10; Ex-1004, (54), 1:6-11; Ex-1005, ¶ 250.

Cherabuddi and Noel are also both reasonably pertinent to a problem described by the '264 Patent to avoid system crashes or other issues arising in one partition that do not affect the other partitions—“Machines also exist that are capable of simultaneously running several copies of the same operating system with each

copy running independently in a separate partition. These machines are advantageous in that each partition may be dedicated to particular users and applications, and problems (including system crashes) that arise in one partition need not adversely affect operation in other partitions.” Ex-1001, 2:60-67. Cherabuddi recognizes the advantages of dynamically allocating cache resources to workaround defective processors. Ex-1002, 2:40-49 (“The ability to dynamically allocate cache resources between multiple processors advantageously allows the entire cache memory to be used, irrespective of whether one or both processors are currently active, thereby maximizing cache resources while allowing for both one-thread and two-thread execution modes. In addition, the present invention may be used to maximize cache resources when one of the on-board processors is defective. For example, if one processor is found to be defective during testing, it may be set as inactive, and the cache access circuit may allocate the entire cache memory to the other processor.”). Noel recognizes that in the prior art, a computing system could be partitioned with hardware to make a subset of the resources on a computer available for a particular application, but the resources were not permanently dedicated, which left “issues concerning performance improvements by means of load balancing of resources among partitions and resource availability.” Ex-1004, 2:7-13. Rather than partitioning a computing system and reserving a subset of system resources for a particular application, the invention taught by Noel discusses

“multiple instances of operating systems execute cooperatively in a single multiprocessor computer wherein all processors and resources are electrically connected together” such that the single physical machine, containing multiple physical processors and resources, “is adaptively subdivided by software into multiple partitions, each with the ability to run a distinct, copy, or instance, of an operating system.” Ex-1004, 4:29-36. Noel recognizes that the “resulting adaptively partitioned, multi-processing (APMP) system exhibits both scalability and high performance.” Ex-1004, 4:54-56; Ex-1005, ¶ 251.

Cherabuddi describes the benefit of dynamically allocating cache resources between multiple processors, maximizing and efficiently utilizing cache resources. Ex-1002, 2:40-49, 3:40-46. Noel similarly observed that it “would be advantageous to dynamically allocate cache to processors on a multiple processor integrated circuit, including on such integrated circuits that are parts of partitionable machines, so as to provide an amount of cache appropriate to each thread, or partition, executing on the system.” Ex-1004, 3:8-12. Noel also recognizes the desirability of having “a new computer system which provides improved flexibility, resource availability[,] and scalability.” Ex-1004, 4:23-25. A POSITA would therefore recognize that including dynamically turning on and off CPUs assigned to each partition and changing partitions without rebooting the system as taught by Noel

would facilitate the modulation of CPU activity and cache partition allocation, as described by Cherabuddi. Ex-1005, 252.

Specifically, it would have been obvious to include Noel's techniques relating to (1) dynamically turning on and off CPUs assigned to each cache partition and (2) changing the cache partitions without rebooting the system in the system of Cherabuddi for the reasons explained below. Ex-1004, 2:60-65, 4:49-55. In particular, Cherabuddi explains that Figure 1 depicts system 10, which includes operating software (also known as "system kernel" or "kernel") that assigns tasks between the CPUs of each MPU. Ex-1002, 3:46-50. The kernel allocates tasks between CPUs depending on the demand for particular applications. Ex-1002, 3:50-60. For example, the kernel will assign tasks to one CPU, while idling the other, for applications having a highly sequential code, and the kernel may assign different threads to both CPUs for simultaneous execution of applications that have parallel execution loops. Ex-1002, 3:52-56. Cherabuddi's disclosure is consistent with the disclosure of the '264 Patent that "[i]t would be advantageous to dynamically allocate cache to processors on a multiple processor integrated circuit, including on such integrated circuits that are parts of partitionable machines, so as to provide an amount of cache appropriate to each thread, or partition, executing on the system." Ex-1001, 3:8-12; Ex-1002, 1:62-66 ("[T]he resources of a partitioned cache memory are dynamically allocated between two or more processors on an MPU based on a

desired system configuration or to the processing needs of the processors.”); Ex-1005, ¶ 253.

A POSITA would have reasonably expected the combination of these teachings of Cherabuddi and Noel to succeed because they describe similar systems; they each describe a multiprocessor system in which the processors share a level of memory, partitioning that memory, and dedicating the processors to a memory partition. Ex-1005, ¶ 264; Ex-1002, 2:8-12 (“In one mode, both processors are set as active, and may simultaneously execute separate instruction threads. In this two-thread mode, the cache access circuit allows each processor to use a corresponding cache memory partitions as a dedicated cache.”); Ex-1004, 4:49-52 (“Resources, such as CPUs and memory, can be dynamically assigned to different partitions and used by instances of operating systems running within the machine by modifying the configuration.”). Also, a POSITA would reasonably expect to succeed because they are in similar fields, use similar techniques, and are compatible; therefore, their combination produces an expected result of (1) dynamically turning on and off CPUs assigned to each cache partition and (2) changing the cache partitions without rebooting. Ex. 1005, ¶ 264.

For the above reasons, it would have been obvious to a POSITA to include a known element (Noel’s teachings relating to dynamically turning on and off CPUs assigned to each partition and changing partitions without rebooting the system) into

another mechanism (Cherabuddi's teachings of a partitionable cache) that yield predictable results for several reasons. Ex-1005, ¶ 264.

A POSITA would have been motivated to combine Cherabuddi and Noel for the reasons described above. During prosecution, the Examiner already found Noel to teach the additional steps added by claim 11, and Patentee did not dispute that. Ex-1007, 297.

A POSITA would recognize that dynamically turning on and off CPUs assigned to each partition and changing partitions without rebooting the system, as taught by Noel, would facilitate the cache partition allocation, as described by Cherabuddi. Ex-1005, ¶ 265. Noel explains that efficient operation of applications in computing environment requires “fast, powerful[,] and flexible computing systems.” Ex-1004, 1:15-17. Noel recognizes that this motivation pervades other sought-after efficiency gains, including “expand[ing] the system work capacity and chang[ing] configurations **to achieve better utilization of resources without stopping execution of application programs on the system.**” Ex-1004, 1:23–27.

Noel teaches that the system has multiple instances of operating systems executing on it, that is, one copy of an operating system running in each partition—“The single physical machine with multiple physical processors and resources is adaptively subdivided by software into multiple partitions, each with the ability to run a distinct copy, or instance, of an operating system,” (“*operating systems in each*

partition”) Ex-1004, 4:33-36; Ex-1005, ¶¶ 250, 271. Noel explains, “Resources, such as CPUs and memory, can be dynamically assigned to different partitions and used by instances of operating systems running within the machine by modifying the configuration.” Ex-1004, 4:49-52. Indeed, “the partitioning of resources is performed by assigning resources within a configuration.” Ex-1004, 4:38-40.

Noel teaches that “*operating systems in each partition*” can be stopped and restarted. Ex-1005, ¶ 272. For instance, Noel explains that “[e]ach operating system instance must be able to shutdown, halt, or otherwise crash in a manner that control is passed to the console program. Conversely, each operating system instance must be able to reboot into an operational mode, independently of any other operating system instance.” Ex-1004, 9:22-27. Here, the operating system in each partition must be able to shutdown, halt, crash, or reboot each partition therefore teaching “*stopping execution of operating systems in each partition, and restarting execution of operating systems in each partition.*” Ex-1005, ¶¶ 272-74.

Noel teaches “*wherein the system is capable of being repartitioned without rebooting each operating system.*” Ex-1005, ¶ 273. Noel explains, “Resources, such as CPUs and memory, can be dynamically assigned to different partitions and used by instances of operating systems running within the machine by modifying the configuration. The partitions themselves can also be changed without rebooting the system.” Ex-1004, Abstract. Noel’s system can be repartitioned without requiring

a reboot. Ex-1005, ¶ 273. Noel also states that “moving an input/output processor from one partition to another may require both partitions to reboot. Changing the memory allocation of fragments, on the other hand, might be handled by an operating system instance without the need for a reboot.” Ex-1004, 22:22-27. Sometimes, repartitioning changes to the system might require rebooting each operating system involved, but in others, the system can be partitioned without rebooting each operating system (“*wherein the system is capable of being repartitioned without rebooting each operating system*”). Ex-1005, ¶ 273.

Cherabuddi and Noel teach this limitation. Ex-1005, ¶ 274.

VIII. DISCRETIONARY DENIAL IS NOT APPROPRIATE

A. The Board Should Not Deny Institution Under 35 U.S.C. § 325(d)

Denial under Section 325(d) is not warranted because the grounds advanced are neither cumulative nor redundant to the prosecution of the '264 Patent, as Cherabuddi, was not cited nor considered by the Examiner during prosecution of the '264 Patent. Thus, the combination of references in this Petition were not considered during prosecution. Further, the '264 Patent has not been previously challenged in an IPR. In addition, the Examiner did not have the benefit of Dr. Jacob's detailed expert testimony and the other evidence of record.

The Examiner only cited Dean for claims 4 and 5 and Noel for claim 11 during prosecution (for which the Examiner did not cite or consider Cherabuddi). And

critically, the Examiner found that Dean taught the additional elements of claims 4 and 5 and Noel taught the additional elements of claim 1111, and the Patentee never disputed those findings despite having two opportunities to do so (both after the Examiner's Non-Final Rejection and Final Rejection). Ex-1007, at 274-81, 316-19. Because the Patentee did not challenge the findings that Dean teaches the additional limitations of claims 4 and 5 and Noel teach the additional limitation of claim 11, the Examiner's consideration of Dean and Noel does not present a reason to deny the Petition under Section 325(d).

Thus, the Board should not exercise discretion to deny institution.

B. The Board Should Not Deny Institution Under 35 U.S.C. § 314(a)

The merits of Petitioner's arguments are compelling and the evidence in support is substantial. That "alone demonstrates that the PTAB should not discretionarily deny institution under *Fintiv*." Ex-1008 at 4-5. But in any event, the six *Fintiv* factors do not justify denying institution.

Factor 1 (stay) is **neutral**. SAP has not yet moved to stay pending this IPR.

Factor 2 (proximity to trial) is **neutral or weighs against denial**. In the originally filed litigation by Patent Owner Valtrus (*Valtrus I*), the trial date is not currently set (although jury selection is set for September 22, 2025). Ex-1009, 1. However, the Board should not rely on the current schedule in *Valtrus I* because Patent Owner Valtrus filed suit alone (without the other named Patent Owner, Key

Patent Innovations) and thus did not have Article III Standing. Ex-1010; Ex-1011. Petitioner filed a Motion to Dismiss for Lack of Standing in *Valtrus I*, reconsideration of which remains pending and casts serious doubt on whether *Valtrus I* will actually proceed along the current schedule. Indeed, in response to Petitioner's Motion to Dismiss for Lack of Standing, Patent Owners essentially concede that *Valtrus* alone lacked standing because Patent Owners together filed a second, duplicative lawsuit (*Valtrus II*) in an effort to rectify the Article III standing deficiencies in *Valtrus I*. Ex-1012; Ex-1013. This conduct confirms that Patent Owners themselves doubt that the trial schedule in *Valtrus I* will hold, further supporting that the Board should not rely on that schedule at all. Additionally, the other pending litigations involving the challenged patent (*Valtrus II* and a declaratory judgment action filed by Petitioner) do not have any trial date currently set, meaning there is no reliable trial date for the Board to consider in relation to this Petition.

Even if the Board assumes the *Valtrus I* trial date applies, this factor is neutral because there will be other claims not litigated at trial that are covered by this Petition. Therefore, a final written decision will issue first at least on the claims not litigated at trial but covered by this Petition.

Factor 3 (investment in parallel proceedings) **weighs against denial**. Importantly, claim 2 is not at issue in the related litigations. *Samsung Bioepis Co. Ltd v. Regeneron Pharmaceuticals, Inc.*, IPR2023-00739, Paper 9 at 57 (Oct. 20,

2023) (“[B]ecause certain of the challenged claims, . . . are not expressly at issue in the district court litigation, *Fintiv* factor 3 favors institution.”). The related litigations are in their early stages of discovery, motions to transfer and dismiss have been filed, and the investment by the parties has been relatively minimal. All litigations involving the challenged patent are at an early stage, with major deadlines several months away. Ex-1009. Patent Owners may argue that the parties have invested substantial resources on procedural motions, but the resources expended have been directed toward resolving procedural missteps perpetuated by Patent Owner Valtrus’s choice to file alone (without Article III standing) and Patent Owners’ later attempts to fix that with procedurally improper duplicative litigation. These procedural disputes (caused by Patent Owners’ actions) should not be held against Petitioner under this factor.

Factor 4 (overlap) **weighs against denial** because Petitioner hereby stipulates that, if the IPR is instituted, it will not pursue the IPR grounds in the district court. Thus, “[i]nstituting trial here serves overall system efficiency and integrity goals by not duplicating efforts and by resolving materially different patentability issues.” *Apple, Inc. v. SEVEN Networks, LLC*, IPR2020-00156, Paper 10 at 19 (June 15, 2020); *see also Sand Revolution II, LLC v. Continental Intermodal Group-Trucking LLC*, IPR2019-01393, Paper 24 at 12 (June 16, 2020). In addition, Petitioner challenges more claims than Patent Owners will ultimately assert at trial. Patent

Owner Valtrus agreed to narrow to 30 total asserted claims by April 30, 2025, and a maximum of six asserted claims per asserted patent. Ex-1009, 1. Thus, Patent Owners will be limited to asserting up to six of the nine asserted claims of the '264 Patent, merely a subset of the claims challenged in this Petition. *See, e.g., Apple*, No. IPR2020-00156, Paper 10, at 17, 21-22.

Factor 6 (other circumstances) **weigh against denial**. Here, the merits of the Petition are particularly compelling—for example, Cherabuddi, Dean, and Noel are strikingly similar disclosures to that of the '264 Patent, are directed at the same problem and proposes the same solutions as the '264 Patent, and teach nearly identical architecture, including several nearly identical figures. The evidence presented, if unrebutted at trial, would plainly lead to a conclusion that one or more claims are unpatentable by a preponderance of the evidence.

While Factor 5 (parties) may weigh slightly in favor of denial because the parties are the same as in district court, under the “holistic view” of the remaining factors and whether integrity of the system and efficiency is best served, institution is favored. *Samsung Elecs. Co. Ltd. v. Dynamics Inc.*, IPR2020-00505, Paper 11 at 15 (Aug. 12, 2020).

IX. CONCLUSION

Claims 1-5 and 8-11 are unpatentable for the reasons demonstrated above.

Dated: January 14, 2025

Respectfully submitted,

/ Louis L. Campbell /

Louis L. Campbell (Lead Counsel)

llcampbell@winston.com

USPTO Reg. No. 59,963

Winston & Strawn LLP

255 Shoreline Dr., Ste. 520

Redwood City, CA 94065

T: (650) 858-6500

Lead Counsel for Petitioner

CERTIFICATE OF COMPLIANCE

This Petition complies with the word count limits set forth in 37 C.F.R. § 42.24(a)(1)(i), because this Petition contains 13,946 words, excluding the parts of the Petition exempted by 37 C.F.R. § 42.24(a)(1), determined using the word count provided by Microsoft Word, which was used to prepare this Petition.

Dated: January 14, 2025

Respectfully submitted,

/ Louis L. Campbell /

Louis L. Campbell (Lead Counsel)

llcampbell@winston.com

USPTO Reg. No. 59,963

Winston & Strawn LLP

255 Shoreline Dr., Ste. 520

Redwood City, CA 94065

T: (650) 858-6500

Lead Counsel for Petitioner

CERTIFICATE OF SERVICE

Under 37 C.F.R. §§ 42.6(e) and 42.105(a), this is to certify that on January 14, 2025, I caused to be served a true and correct copy of the foregoing “**PETITION *INTER PARTES* REVIEW OF U.S. PATENT NO. 6,871,264**” and accompanying exhibits by Federal Express priority overnight mail on the following:

Botos Churchill IP Law LLP
430 Mountain Avenue
Suite 401
New Providence, NJ 07974

With a courtesy copy of this Petition and supporting material was also served on litigation counsel for Patent Owners via email:

Matt Berkowitz
mberkowitz@reichmanjorgensen.com

Patrick R. Colsher
pcolsher@reichmanjorgensen.com

Aaron Morris
amorris@reichmanjorgensen.com

Navid Cyrus Bayar
nbayar@reichmanjorgensen.com

Reichman Jorgensen Lehman & Feldberg LLP
100 Marine Parkway
Suite 300
Redwood Shores, CA 94065

Khue Van Hoang
khoang@reichmanjorgensen.com
Reichman Jorgensen Lehman & Feldberg LLP
400 Madison Avenue
Suite 14d
New York, NY 1001

Philip Ekle
peklem@reichmanjorgensen.com
Savannah Harman Carnes

scarnes@reichmanjorgensen.com

Reichman Jorgensen Lehman & Feldberg LLP
1909 K Street, NW
Suite 800
Washington, DC 20006

/ Louis L. Campbell /
Louis L. Campbell (Lead Counsel)
llcampbell@winston.com
USPTO Reg. No. 59,963
Winston & Strawn LLP
255 Shoreline Dr., Ste. 520
Redwood City, CA 94065
T: (650) 858-6500

Lead Counsel for Petitioner