

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMERICAN AIRLINES, INC. AND SOUTHWEST AIRLINES CO.,

Petitioners

v.

THE REGENTS OF THE UNIVERSITY OF CALIFORNIA,

Patent Owner.

Inter Partes Review No. IPR2025-01511

U.S. Patent No. 7,712,080 B2

Issue Date: May 4, 2010

Title: SYSTEMS AND METHODS FOR PARALLEL DISTRIBUTED PROGRAMMING

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 7,712,080 B2**

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

I.	Mandatory Notices Under 37 C.F.R. § 42.8.....	2
A.	Real Party in Interest Under § 42.8(b)(1).....	2
B.	Related Matters Under § 42.8(b)(2)	3
C.	Lead and Backup Counsel Under § 42.8(b)(3)	3
D.	Service Information.....	5
II.	Fee Payment.....	5
III.	Grounds for Standing.....	5
IV.	Brief Overview of the '080 Patent.....	5
V.	Prosecution History of the '080 Patent.....	8
VI.	Priority Date.....	11
VII.	Claim Construction.....	12
VIII.	Person of Ordinary Skill in the Art (POSITA).....	12
IX.	Precise Relief Requested and Grounds Raised.....	13
X.	Brief Description of the Applied Prior Art Reference Including Descriptions of “MESSENGERS”	14
A.	Fukuda '98 (Ex.1008)	14
B.	Fukuda '01 (Ex.1009)	19
XI.	The Challenged Claims are Unpatentable	22
A.	Ground 1: Claims 1-18 are anticipated by Fukuda '98 (Ex.1008).....	22
1.	Claim 1	23

- a. [1pre] A method of developing a distributed parallel computing program, comprising steps of: 23
- b. [1a] establishing at least one distributed shared variable, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically distributed across multiple memories; 23
- c. [1b] developing at least one distributed sequential computing program to access the at least one distributed shared variable;.....26
- d. [1c] transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one intermediate condition occurs within the at least one distributed sequential computing program.....29
- e. [1d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations, ...34
- f. [1e] wherein the at least one intermediate condition comprising

	one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.....	35
2.	Claim 2: The method of claim 1, wherein each of the at least one distributed sequential computing program includes at least one self-migrating thread, capable of migrating among at least one processor and capable of accessing the at least one distributed shared variable.....	38
3.	Claim 3: The method of claim 1, wherein the at least one distributed shared variable is configured to be located among a plurality of memories.....	40
4.	Claim 4: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple processors.....	41
5.	Claim 5: The method of claim 1, wherein the distributed parallel computing program is configured to operate using multiple threads.	41
6.	Claim 6: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple nodes.	42
7.	Claim 7: The method of claim 1, further comprising the step of configuring the at least one distributed sequential computing program to include at least one mobile agent.	43

8. Claim 8: The method of claim 7, wherein the at least one mobile agent is implemented using self-migrating threads.....44
9. Claim 9.....44
 - a. [9pre]: A distributed parallel computing system, having at least one memory area and at least one processor, comprising:.....44
 - b. [9a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area;.....45
 - c. [9b] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable, and.....46
 - d. [9c] configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program, ...46
 - e. [9d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned

	child distributed sequential computing program to perform parallel processing and/or operations, ...	47
f.	[9e] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.....	47
10.	Claim 10: The system of claim 9, wherein the at least one distributed sequential computing program comprises at least one mobile agent.	47
11.	Claim 11: The system of claim 10, wherein the at least one mobile agent is implemented using self-migrating threads.....	48
12.	Claim 12: The system of claim 10, wherein the at least one mobile agent is an explicit- navigation mobile agent.....	48
13.	Claim 13: The system of claim 10, wherein the at least one mobile agent is an implicit- navigation mobile agent.....	49
14.	Claim 14: The system of claim 10, wherein the at least one mobile agent is configured to move from one processor to another processor.	50
15.	Claim 15: The system of claim 10, wherein the at least one mobile agent operates within a distributed shared memory system.....	51
16.	Claim 16: The system of claim 9, wherein the at least one processor is located across a physical network.	51

17.	Claim 17: The method of claim 1, wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.....	51
18.	Claim 18: The method of claim 9, wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.....	52
XII.	Conclusion	52

TABLE OF AUTHORITIES

	Page(s)
CASES	
<i>Guangdong Alison Hi-Tech Co. v. Int'l Trade Comm'n</i> , 936 F.3d 1353 (Fed. Cir. 2019)	22
<i>In re GPAC Inc.</i> , 57 F.3d 1573 (Fed. Cir. 1995)	12
<i>Phillips v. AWH Corp.</i> , 415 F.3d 1303 (Fed. Cir. 2005)	12
<i>Sotera Wireless, Inc. v. Masimo Corp.</i> , IPR2020-01019, Paper 12 (P.T.A.B. Dec. 1, 2020)	13
STATUTES	
35 U.S.C. § 102	passim
OTHER AUTHORITIES	
37 C.F.R. § 42.8	2
37 C.F.R. § 42.100(b)	12
37 C.F.R. § 42.103(a)	5
37 C.F.R. § 42.15(a)	5
83 Fed. Reg. 51,340 (Oct. 11, 2018)	12

LIST OF EXHIBITS

Exhibit No.	Description of Document
1001	U.S. Patent No. 7,712,080 B2 (the '080 Patent) to Lei Pan, Lubomir R. Bic, and Michael B. Dillencourt (filed May 21, 2004; issued May 10, 2010).
1002	File history for the '080 Patent.
1003	Declaration of Dr. Michael Shamos regarding the invalidity of U.S. Patent No. 7,712,080 B2.
1004	CV for Dr. Michael Shamos.
1005	Lei Pan, Lubomir F. Bic, Michael B. Dillencourt, <i>Distributed Sequential Computing Using Mobile Code: Moving Computation to Data</i> , International Conference on Parallel Processing, pp. 77-84, September 3-7, 2001, available at https://ieeexplore.ieee.org/document/952049 ("Pan").
1006	Naoya Suzuki, Munehiro Fukuda, Lubomir F. Bic, <i>Self-Migrating Threads for Multi-Agent Applications</i> , ICWC 99; IEEE Computer Society International Workshop on Cluster Computing, December 2-3, 1999, available at https://ieeexplore.ieee.org/document/810828 ("Suzuki").
1007	Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, Fehmina Merchant, <i>Message versus Messengers in Distributed Programming</i> , Proceedings of 17th International Conference on Distributed Computing Systems, May 27-30, 1997, available at https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1c575e92687dbc06acec75ae66a7a2f9174cb31c ("Fukuda '97")
1008	Munehiro Fukuda, Lubomir B. Bic, Michael B. Dillencourt, Fehmina Merchant, <i>Distributed Coordination with Messengers</i> , Science of Computer Programming 31 (July 1998) 291-311, available at https://www.sciencedirect.com/science/article/pii/S0167642397000245 ("Fukuda '98")..

Exhibit No.	Description of Document
1009	Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, Fehmina Merchant, <i>Messengers: Distributed Programming Using Mobile Agents</i> , Transactions of the Society for Design and Process Science, Vol. 5, No. 4, pp. 95-112, December 2001, available at https://scispace.com/papers/messengers-distributed-programming-using-mobile-agents-50ktvx5clr (“Fukuda ’01”).
1010	Amended Complaint, <i>Intellectual Ventures I LLC v. American Airlines, Inc.</i> , No. 4:24-cv-980-ALM (E.D. Tex. Nov. 2, 2024) (Dkt. No. 84).
1011	Amended Complaint, <i>Intellectual Ventures I LLC v. Southwest Airlines Co.</i> , No. 7:24-cv-277-ALM (W.D. Tex. Nov. 2, 2024) (Dkt. No. 56).
1012	IV’s Infringement Contentions re the ’080 patent in <i>Intellectual Ventures I LLC v. American Airlines, Inc.</i> , No. 4:24-cv-980-ALM (E.D. Tex. Nov. 2, 2024).
1013	IV’s Infringement Contentions re the ’080 patent in <i>Intellectual Ventures I LLC v. Southwest Airlines Co.</i> , No. 7:24-cv-277-ALM (W.D. Tex. Nov. 2, 2024).

In May 2025, Intellectual Ventures I LLC (“IV”) targeted Petitioners American Airlines, Inc. (“American”) and Southwest Airlines Co. (“Southwest”) (collectively “Petitioners”) with accusations of infringing U.S. Patent No. 7,257,080 B2 (the ’080 Patent) (Ex.1001). IV is **not** the owner of the ’080 patent. As stated on the face of the ’080 patent, the Regents of the University of California is the owner of the ’080 patent. Ex.1001. IV contends that it “is the exclusive licensee under the ’080 Patent, and holds all substantial rights therein, including the right to grant licenses, to exclude others, and to enforce and recover past damages for infringement of the ’080 Patent.” Ex.1011 at ¶ 37; Ex.1010 at ¶ 37.

Petitioners respectfully request that the Board now test the ’080 Patent through *inter partes* review (IPR) of claims 1-18 (the Challenged Claims), constituting all claims of the ’080 Patent. The ’080 Patent relates to the creation of a parallel distributed program from a computing distributed sequential program that spawns a fetching distributed sequential program that run concurrently to perform parallel or pipelined computing. The program operates in a space having at least one distributed shared variable located across the multiple memories and multiple processors. Ex.1001 (2:11-16; 5:57-6:65); Ex.1003 ¶ 28. The ’080 patent describes the MESSENGERS system as an example of the alleged invention. *Id.* (8:26-10:17); Ex.1003 ¶¶ 34-35; 47.

In July 1998 (more than one year prior to the earliest filing date of the '080 patent—May 2003), two of the three named inventors of the '080 patent published a paper entitled *Distributed Coordination with MESSENGERS* which describes the same MESSENGERS system that is described in the '080 patent and which also discloses each limitation of each of the claims (1-18) of the '080 patent. Ex.1008 (“Fukuda '98); Ex.1003, ¶¶ 46, 93, 123-240.

The Examiner did not consider the July 1998 *Distributed Coordination with MESSENGERS* paper, let alone whether that reference disclosed the claim limitations. Claims 1-18 of the '080 patent are anticipated by Fukuda '98 (Ex.1008) under 35 U.S.C. § 102 and are therefore invalid.

Petitioners respectfully request the Board to institute IPR of the '080 Patent and cancel the Challenged Claims.

I. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8

A. Real Party in Interest Under § 42.8(b)(1)

American is a real party in interest to this petition. American is a wholly owned subsidiary of American Airlines Group, Inc., which is not a real party in interest to this petition. No unnamed entity is funding, controlling, or otherwise has an opportunity to control or direct this petition or American's participation in any resulting IPR.

Southwest is also a real party in interest to this petition. No unnamed entity is funding, controlling, or otherwise has an opportunity to control or direct this petition or Southwest's participation in any resulting IPR.

B. Related Matters Under § 42.8(b)(2)

To the best of Petitioners' knowledge, the '080 Patent is or has been involved in these matters:

Name	No.	Court	Filed	Status
<i>Intellectual Ventures I LLC et al. v. American Airlines, Inc.</i>	4:24-cv-980	EDTX	Nov. 2, 2024	Active
<i>Intellectual Ventures I LLC et al. v. Southwest Airlines Co.</i>	7:24-cv-277	WDTX	Nov. 2, 2024	Active

C. Lead and Backup Counsel Under § 42.8(b)(3)

Petitioners designate the following counsel:

Lead Counsel	Backup Counsel
John B. Campbell (Reg. No. 54,665) jcampbell@McKoolSmith.com McKool SMITH, P.C. 303 Colorado Street, Suite 2100 Austin, TX 78701 Tel: (512) 692-8730 Fax: (512) 692-8744	Alan Block (Reg No. 35,450) ablock@mckoolsmith.com McKool Smith, P.C. 300 South Grand Avenue, Suite 2900 Los Angeles, California 90071 Tel: (213) 694-1200 Fax: (213) 694-1234 Casey Shomaker (Reg. No. 77,998) cshomaker@mckoolsmith.com McKool Smith, P.C.

Petition for *Inter Partes* Review of
U.S. Patent No. 7,712,080 B2

Lead Counsel	Backup Counsel
	300 Crescent Court, Suite 1200 Dallas, TX 75201 Tel: (214) 978-4218 Fax: (214) 978-4044
	Keith D. Harden (Reg. No. 74,472) kharden@munckwilson.com Munck Wilson Mandala, LLP 2000 McKinney Ave., Ste. 1900 Dallas, Texas 75201 Tel: (972) 628-3600 Fax: (972) 628-3616
	S. Wallace Dunwoody (Texas Bar No. 24040838 - admission <i>pro hac vice</i> to be requested) wdunwoody@munckwilson.com Munck Wilson Mandala, LLP 2000 McKinney Ave., Ste. 1900 Dallas, Texas 75201 Tel: (972) 628-3600 Fax: (972) 628-3616
	Michael C. Wilson (Texas Bar No. 21704590 - admission <i>pro hac vice</i> to be requested) mwilson@munckwilson.com Munck Wilson Mandala, LLP 2000 McKinney Ave., Ste. 1900 Dallas, Texas 75201 Tel: (972) 628-3600 Fax: (972) 628-3616

D. Service Information

USPTO records show the attorneys having power of attorney over the '080 Patent are Orrick, Herrington & Sutcliff LLP. This petition is thus being served by Federal Express to the correspondence address for the '080 Patent: Orrick, Herrington & Sutcliff, Attn: Kenneth S. Roberts, 4 Park Plaza, Suite 1600, Irvine, California 92614-2558. Petitioners consent to electronic service at AA_Intellectual_Ventures@mckoolsmith.com.

II. FEE PAYMENT

The Office is allowed to charge the fees specified by 37 C.F.R. §§ 42.103(a) and 42.15(a) to Deposit Account No. 50-5723.

III. GROUNDS FOR STANDING

Petitioners certify that the '080 Patent is available for IPR and that Petitioners are not barred or estopped from requesting an IPR challenging the '080 Patent's claims on the grounds identified in this petition. Petitioners filed this petition within one year of service of the amended complaints against Petitioners.

IV. BRIEF OVERVIEW OF THE '080 PATENT

The '080 patent addresses software applications that operate on multiple processors that use multiple memories, which are referred to as "parallel distributed programs." Ex.1001 (1:25-30); Ex.1003 ¶ 28.

In one aspect of the invention, “a parallel distributed program includes at least one distributed shared variable located across the multiple memories and one or more distributed programs configured to operate across multiple processors.” *Id.* (2:11-16). In another aspect of the invention, “the one or more distributed programs include one or more self-migrating threads configured to migrate from one processor to another.” Ex.1001 (2:16-19).

More specifically, the '080 patent relates to “navigational programming,” which includes distributed sequential computing (“DSC”). DSC “allows a programmer to develop threads that efficiently access multiple memories spanning across multiple processors without having to implement the complicated, tedious, time consuming, and error-prone low-level tasks of message handling in distributed programming.” Ex.1001 (3:60-64).

Another aspect of “navigational programming” is distributed parallel computing (“DPC”). DPC is computing using multiple concurrent DSC programs. Ex.1001 (5:57-59); Ex.1003 ¶ 30. In one example provided in the '080 patent, a DSC program “injects” or spawns a separate DSC program that performs “fetching.” *Id.* (6:27-60); Ex.1003 ¶ 31. The '080 patent states that the computing DSC program and the fetching DSC program “together make a DPC program”:

The computing DSC program and the fetching DSC program together make a DPC program. The two DSC threads of the DPC program run concurrently to perform parallel or pipelined

computing. In this particular example, the computations and the fetching are pipelined.

Id. (6:61-65); Ex.1003 ¶ 32.

The '080 patent further states that “[n]avigational programming may be facilitated by mobile agent systems. Such a mobile agent has the ability to halt its execution, encapsulate the values of its variables, move to another node, restore the state, and continue executing. This ability is often referred to as strong mobility.” Ex.1001 (8:20-25); Ex.1003 ¶ 33. An example of such a mobile agent system is MESSENGERS, “in which applications are developed as collections of mobile agents, which may be referred to as Messengers.” *Id.* (8:29-31); Ex.1003 ¶ 34. The '080 patent provides a description of MESSENGERS at 8:31-10:16, which, among other things, includes a description of the “two types of variables in MESSENGERS: agent variables and node variables.” *Id.* (9:1-2).

The '080 patent further describes the use of the “inject()” statement to spawn another Messenger: “[a] Messenger can spawn another Messenger using a statement, such as inject()” Ex.1001 (9:16-17) (*see also* 8:39-43: “Messengers may be injected by a program, which is part of the MESSENGERS system, or by another Messenger into any of the daemon nodes, and they may start creating new logical nodes and links on the current or any other daemons.”); Ex.1003 ¶ 35.

Injecting new Messengers is a task for the daemon: “There are four tasks for a daemon. First, to accept signals, such as UNIX signals, to inject Messengers. These signals are sent by the program minject or another messenger.” Ex.1001 (8:50-53); Ex.1003 ¶ 35.

V. PROSECUTION HISTORY OF THE '080 PATENT

Applicants Lei Pan, Lubomir F.Bic, and Michael B. Dillencourt filed the application for the '080 patent on May 21, 2004 (S/N 10/850,842), claiming priority to a provisional application (No. 60/472,612) filed on May 21, 2003.

The Examiner issued a first office action on May 21, 2007 (Ex.1002 at 161) to which the Applicants responded with a Rule 132 Declaration (*Id.* at 154).

The Examiner issued a second office action on October 18, 2007, rejecting claims 1-7 and 9-25 as being unpatentable over Pan, *Distributed Sequential Computing Using Mobile Code: Moving Computation to Data* (“Pan”) (Ex.1005) in view of Suzuki, *Self-Migrating Threads for Multi-Agent Applications* (“Suzuki”) (Ex.1006). Ex.1002 at 126-131. The Examiner further rejected claim 8 as being unpatentable over Pan (Ex.1005) in view of Suzuki (Ex.1006) and further in view of Fukuda, *Message versus Messengers in Distributed Programming* (“Fukuda '97”) (Ex.1007). Ex.1002 at 131-132.

In their response, Applicants amended claim 1 to include the limitations of claims 7 and 8, which they cancelled, and similarly amended claim 15. Ex.1002 at 111-112, 115.

The Examiner issued a third office action on July 22, 2008, rejecting claims 1-6, 9-10, and 15-22 as being unpatentable over Pan (Ex.1005) in view of Suzuki (Ex.1006) and further in view of Fukuda (Ex.1007). Ex.1002 at 100-105.

In their response, Applicants made no amendments to the claims, but argued that “Suzuki discloses a computing paradigm comprising a set of autonomous self-migrating threads that operates in a sequential sense and not in the parallel sense as claimed by Applicants.” Ex.1002 at 89 (emphasis in original). Thus, as Applicants argued, “[n]othing in Suzuki discloses a parallel system. One of ordinary skill in the art would readily recognize that parallel processing enables many calculations to be carried out simultaneously, i.e., without terminating and resuming thread processes.” *Id.* at 90.

The Examiner issued a fourth office action on April 2, 2009, rejecting claims 1-6, 9-10, and 15-22 as being unpatentable over Pan (Ex.1005) in view of Suzuki (Ex.1006) and further in view of Fukuda (Ex.1007). Ex.1002 at 75-80. Responding to Applicants’ arguments regarding Suzuki, the Examiner maintained that Suzuki meets the claim limitation of “a parallel distributed application,” because “such parallel distributed application can be reasonabl[y] interpreted as any type[] [of]

application[s] that exist in parallel and distributed forms including the parallel distributed programs or threads or nodes as disclosed by Suzuki (see example, p.3, right column, section ‘Thread Navigation’, ‘The thread propagates along links and/or jumps directly to nodes using its member function hop()’; also see Fig.3 at page 4).” Ex.1002 at 75.

In response, the Applicants amended independent claims 1 and 15 to describe the “parallel program” as a “parallel computing program” and to add a wherein clause: “wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one child distributed sequential computing program to perform parallel processing and/or operations.” Ex.1002 at 55, 56-57.

Applicants then contended that “[n]either Pan, Suzuki or Fukuda, alone or in combination, teach or suggest the claimed invention as claimed in independent claims 1 and 15,” stating that:

Pan, Suzuki and Fukuda simply do not teach or suggest transforming a distributed sequential computing (DSC) program into a distributed parallel computing program (DPC) program or a DSC program configured to transform into a DPC program where the DPC concurrently uses the DSC and child DSC to perform parallel and/or pipeline processing. Although Pan may describe the concept of DSC programs, which is computing using a single locus of computation over distributed data (Application [0024]), Pan does not disclose or suggest transforming the DSC into a DPC by spawning a child DSC or a DSC configured to transform into a DPC by spawning a child DSC. Similarly, Suzuki describes the use

of a collection of self-migrating threads. Although, as Applicants acknowledge at paragraph 0024 of the application, a self-migrating thread may be utilized to perform DSC, a description of the use of a collection of self-migrating threads does not disclose or suggest transforming a DSC into a DPC by spawning a child DSC or DSC configured to transform into a DPC by spawning a child DSC.

. . . Moreover, in light of Pan's discussion of implementation of DSCs using MESSENGERS, the fact that Fukuda describes replication does not teach or suggest transforming a DSC into a DPC by spawning a child DSC wherein the DPC concurrently uses the DSC and child DSC to perform parallel processing and/or operations, but rather only suggests to one skilled in the art the replication of a MESSENGER within an individual DSC program.

Ex.1002 at 59-60.

After the Examiner agreed to the Applicants' proposed amendments to further define the terms "distributed shared variable" and "intermediate condition," the '080 patent issued. Ex.1002 at 38-40.

VI. PRIORITY DATE

For purposes of this IPR, Petitioners assume the earliest effective filing date of the '080 Patent is May 21, 2003, which is the date on which applicants filed Provisional Application No. 60/472,612 ('612 provisional) to which the '080 Patent claims priority. Ex.1001 (Cover). Thus, this petition applies pre-AIA 35 U.S.C. §102 in showing unpatentability. Petitioners' prior art references qualify as prior art. *See infra* Section IX.

VII. CLAIM CONSTRUCTION

Petitioners apply the *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005), claim construction standard. 83 Fed. Reg. 51,340, 51,340-44 (Oct. 11, 2018). Petitioners do not believe explicit construction is needed to resolve this petition as the analysis herein establishes that the challenged claims are obvious under any reasonable interpretation under the *Phillips* standard, and thus Petitioners submit that the Board need not resolve any express constructions. Still, the district court case is in its early stages, having yet to commence claim construction and the parties have not identified any disputed claim terms.¹ Petitioners reserve the right to revisit constructions proposed in district court and, in accordance with 37 C.F.R. § 42.100(b), request leave to submit the district court’s claim construction order so the Board may consider it.

VIII. PERSON OF ORDINARY SKILL IN THE ART (POSITA)

Based on the relevant factors, *In re GPAC Inc.*, 57 F.3d 1573, 1579 (Fed. Cir. 1995), a POSITA “would have had a bachelor’s degree in computer science, computer engineering, electrical engineering, or related field, and at least two years of experience in computer networking and parallel computing, or a person with a master’s degree in one of the foregoing and at least one year of experience in the

¹ Patent Owner very broadly reads the claims asserted in the litigation against commonly and widely used open-source software known as Apache Spark. *See* Exs. 1012 and 1013.

aforementioned fields. Ex.1003 ¶ 40. Additional education could substitute for professional experience, and vice-versa. *Id.*

IX. PRECISE RELIEF REQUESTED AND GROUNDS RAISED

Petitioners request cancellation of claims 1-18 based on the following ground:

Ground	Claims	Basis for Challenge
1	1-18	Anticipated by Fukuda '98 under §102 (Ex.1008).

Petitioners stipulate that, if trial is instituted, Petitioners will not pursue in the District Court Litigation any ground raised or that could have been reasonably raised in an IPR. *Sotera Wireless, Inc. v. Masimo Corp.*, IPR2020-01019, Paper 12 at 18 (P.T.A.B. Dec. 1, 2020).

Petitioners also submit the Declaration of Dr. Michael Shamos (Ex.1003), a qualified expert, in support of the anticipation ground. *See* Ex.1003 ¶¶ 3-13; Ex.1004.

Ground 1 is not cumulative of the art of record, including the art of record considered by the Examiner, as Fukuda '98 was not before the Examiner during prosecution.

During prosecution, Applicants argued that “Pan, Suzuki and Fukuda simply do not teach or suggest transforming a distributed sequential computing (DSC)

program into a distributed parallel computing program (DPC) program or a DSC program configured to transform into a DPC program where the DPC concurrently uses the DSC and child DSC to perform parallel and/or pipeline processing.” Ex.1002 at 59; Ex.1003 ¶ 89-92. As a result, Applicant’s sole point of novelty over the prior art was transforming a distributed sequential computing (DSC) program into a distributed parallel computing program (DPC) program or a DSC program configured to transform into a DPC program where the DPC concurrently uses the DSC and child DSC to perform parallel and/or pipeline processing. *Id.* Ground 1 addresses this purported deficiency. Ex.1003 ¶ 92.

**X. BRIEF DESCRIPTION OF THE APPLIED PRIOR ART
REFERENCE INCLUDING DESCRIPTIONS OF “MESSENGERS”**

A. Fukuda ’98 (Ex.1008)

Fukuda ’98 is a publication entitled “Distributed Coordination with Messengers.” Fukuda ’98 was published in July, 1998 in Science of Computer Programming, at pages 291-311 of Volume 31, Issues 2-3. Thus, Fukuda ’98 qualifies as pre-AIA prior art. 35 U.S.C. §§ 102(a), 102(b). Two of the authors of Fukuda ’98 are also named inventors of the ’080 patent—Lubomir F. Bic and Michael B. Dillencourt. When examining the ’080 patent application, the Examiner did not have Fukuda ’98.

Fukuda ’98 describes a programming paradigm called MESSENGERS:

MESSENGERS² is a paradigm for the programming of distributed systems. It is based on the principles of autonomous messages, called Messengers, which carry their own behavior in the form of a program. This enables them to navigate freely in the underlying computational network, communicate with one another, and invoke compiled node-resident C functions in the nodes they visit. Hence, a distributed application is viewed as a collection of C functions whose invocation and interoperation is orchestrated by Messengers.

Ex.1008 at 291; Ex.1003 ¶ 61.

The system described in Fukuda '98 is based on an underlying shared state space through which processes, called Messengers, communicate. Ex.1008 at 292; Ex.1003 ¶ 63. As described in Fukuda '98, the underlying state space contains both functions and data. *Id.*; Ex.1003 ¶ 63. Messengers may read or write data and dynamically incorporate functions into their behavior. *Id.*; Ex.1003 ¶ 64. The shared space is structured by being mapped onto a logical network. *Id.* Every data item or function in the shared state space resides at some node in this network. *Id.*; Ex.1003 ¶ 63.

Messengers have state information, consisting of local variables and control data and migrate to nodes containing data that they wish to read, write, or modify, and migrate to nodes containing functions that they wish to invoke. Ex.1008 at 292; Ex.1003 ¶ 64.

² Fukuda '98 uses the term "MESSENGERS" in all caps to denote the entire system, and uses the term "Messengers" with initial caps only to denote the individual processes. In this Petition, Petitioner shall use the same nomenclature. Ex.1008 at 292 n 1.

More specifically, a Messenger is “a self-contained object, consisting of a program and current state, including a program counter and local variables.” Ex.1008 at 293; Ex.1003 ¶ 65. “The program, which is a combination of interpreted MESSENGERS code and native C code, describes the Messenger’s behavior, which allows it to navigate autonomously through the underlying communication network and carry out computation in the nodes it visits.” *Id.*; Ex.1003 ¶ 65.

The system disclosed in Fukuda ’98 is “implemented as a collection of daemons instantiated on all physical nodes participating in the distributed computation. The daemon’s task is to continuously receive Messengers injected from the user interface or arriving from other daemons, to interpret the Messengers’ behaviors described by their scripts, and to send them on to their next destinations as appropriate.” Ex.1008 at 294; Ex.1003 ¶ 66.

The network used by Messengers is a logical network comprised of both logical and physical nodes. Ex.1008 at 294; Ex.1003 ¶ 67. The logical network is “an application-specific computation network created on top of the daemon network. Multiple logical network nodes may be created on the same daemon network nodes, thus running on the same physical node, and they may be interconnected by logical links into an arbitrary topology.” *Id.*; Ex.1003 ¶ 67.

Within the system, there are three types of variables: messenger, node, and network variables. Ex.1008 at 301; Ex.1003 ¶ 68. “Messenger variables are private to and carried by each Messenger as it propagates through the logical computational network. Node variables are resident in nodes and shared by all Messengers currently running on the same logical node. Network variables are predefined at each logical node and give each Messenger access to the network information local to the current node.” *Id.*; Ex.1003 ¶ 68.

In addition to terminating and executing a navigational statement, which causes the Messenger to move to another node, Messengers can also invoke a compiled C function using the *func()* or *exec()* statements. Ex.1008 at 295; Ex.1003 ¶ 69.

The *exec()* statement is as follows: “*exec(file = filename [; in = argument] [; out = argument;] [nowait]*”).” Ex.1008 at 301. Fukuda ’98 describes *exec()* statements as spawning an executable C program that operates concurrently with the invoking Messenger:

The parameter *filename* is a Unix path name that specifies a file containing an executable C program. This must contain a *main()* function, which is spawned as a separate Unix process, i.e., operating concurrently with the invoking Messenger. The arguments specified by *in* are passed to the new process at the time of creation and results are returned to the invoking Messenger via the *out* argument. If *nowait* is specified, the process may continue

³ “Nowait” is optional. Ex.1008 at 301.

operating even after the parent Messenger has left the node (or has terminated). With this option, no results are returned to the invoking Messenger.

Id.

The *func()* statement is as follows: “*func(file = filename; name = func_name* [*; in = argument*] [*; out = argument*]).” Ex.1008 at 302; Ex.1003 ¶ 70. Fukuda '98 describes the *func()* statement as being similar to the *exec()* statement, except that it contains executable C functions rather than a complete program with a *main()* function, as with the *exec()* statements:

Similar to *exec*, the parameter *filename* is a Unix path name specifying a file. This must contain one or more executable C functions (but not necessarily a complete program with a *main()* function). The function specified by the parameter *func_name* is invoked and the arguments specified by *in* are passed to it. Unlike with *exec*, the function executes as part of the current Messenger, that it, as part of the MESSENGERS interpreter process. The invoking Messenger is blocked until the function terminates and returns its results to the Messenger script via the *out* argument.

Id.; Ex.1003 ¶ 70.

Fukuda '98 further describes how, in MESSENGERS, new processes are created: “In MESSENGERS, [process creation] is done by injecting a new Messenger.” Ex.1008 at 307-308; Ex.1003 ¶ 71.

Fukuda '98 describes an exemplary biological simulation using Messengers that “illustrates how a distributed simulation can be structured as a collection of functions, residing on different nodes of a network, where the invocation of these

functions and the exchange of data among them is orchestrated by Messengers.” Ex.1008 at 303; Ex.1003 ¶ 72. In the example, Fukuda ’98 describes creating (“injecting”) a Messenger: “The corresponding Messenger is injected into the lung node, where it first invokes the *new_lung_cone* function (line 4) to compute the new styrene concentration using *inhaled_sty* and *c_veinal* (initially zero).” *Id.* at 304; EX1003 at ¶ 72.

Fukuda ’98 describes a benefit of Messengers as being their capability of “self-migration,” “i.e., autonomous navigation through both the logical and the physical networks.” Ex.1008 at 306; Ex.1003 ¶ 73. Thus, “[a]n important implication of the MESSENGERS paradigm is that the individual entities move not only within the simulated space but actually migrate among processors in the network,” which occurs “when an entity crosses the boundary of a logical region” and “[t]he corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*; Ex.1003 ¶ 73.

B. Fukuda ’01 (Ex.1009)

Fukuda ’01 is a publication entitled “Messengers: Distributed Programming Using Mobile Agents.” Fukuda ’01 was published in December 2001 in Transactions of the Society for Design and Process Science, Vol. 5, No. 4, pp. 95-112. Thus, Fukuda ’01 qualifies as pre-AIA prior art. 35 U.S.C. §§ 102(a), 102(b).

Two of the named authors of Fukuda '01 are also named inventors of the '080 patent—Lubomir F. Bic and Michael B. Dillencourt. When examining the '080 patent application, the Examiner did not have Fukuda '01.

Fukuda '01 describes distributed applications, which it defines as “collections of multiple concurrent activities executing on a loosely coupled network of computer nodes.” Ex.1009 at 95; Ex.1003 ¶ 77. Fukuda '01 describes the difference between explicit concurrency and implicit concurrency in distributed applications: “The majority of distributed applications are developed using explicit concurrency, that is, using language where the programmer is required to divide the problem into separate activities and specify their dependencies. This is accomplished using various constructs, such as *fork*, *cobegin*, *parfor*, or concurrent process declarations. Implicit concurrency relies on the compiler to extract the potentially parallel activities and to provide both the mapping and the coordination primitives.” *Id.*; Ex.1003 ¶ 77.

Like Fukuda '98, Fukuda '01 describes a programming paradigm called MESSENGERS:

In this paper we present and explore a new programming paradigm for distributed systems, which differs from those mentioned above in its basic philosophy. The main concept is mobile agents, which are self-contained autonomous programs that have their own identity and behavior. Mobile agents can navigate through the underlying computational network, perform various tasks at the nodes they visit, and coordinate their activities with other agents

participating in the distributed computation. We have developed a mobile agent system, called MESSENGERS, which we will use to illustrate concepts underlying general purpose distributed programming using mobile agents.

Ex.1009 at 96; Ex.1003 ¶ 78.

Fukuda '01 describes the three levels of networks in MESSENGERS: the physical network, the daemon network, and the logical network (Ex.1009 at 96-97), which Fukuda '01 depicts in its Figure 1:

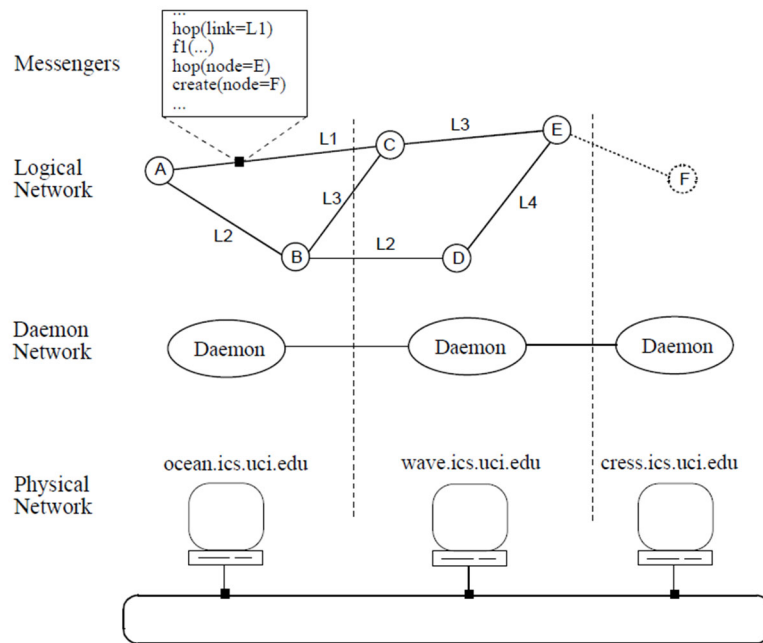


Fig. 1 Network Layers.

Ex.1009 at 97; Ex.1003 ¶ 79.

Fukuda '01 describes the principles of operation and the programming of the MESSENGER system. Ex.1009 at 96-99; Ex.1003 ¶ 80. In describing the programming for Messengers, Fukuda '01 describes the “[t]ask

control/coordination statements,” including the “inject” statement, which “initiates (injects) the execution of one or more instances of another Messenger. The parameters specify the Messenger script to be injected and the initial parameters to be passed to the newly created Messenger. In addition the starting destination(s) may be specified.” *Id.* at 99; Ex.1003 at ¶ 80.

XI. THE CHALLENGED CLAIMS ARE UNPATENTABLE

As Fukuda '98 discloses each of the limitations of claims 1-18 of the '080 patent, Fukuda '98 anticipates each claim of the '080 patent, thereby rendering it invalid under § 102. Ex.1003 ¶ 46, 93, 123-240.

A. Ground 1: Claims 1-18 are anticipated by Fukuda '98 (Ex.1008)

Claims 1-18 of the '080 patent are invalid as being anticipated by Fukuda '98, *i.e.*, every limitation of claims 1-18 is expressly or inherently disclosed by Fukuda '98 (Ex.1008). 35 U.S.C. § 102; *Guangdong Alison Hi-Tech Co. v. Int'l Trade Comm'n*, 936 F.3d 1353, 1363-1364 (Fed. Cir. 2019) (citing *SRI Int'l, Inc. v. Internet Sec. Sys., Inc.*, 511 F.3d 1186, 1192 (Fed. Cir. 2008)) (“A patent claim is invalid as anticipated only if each and every element of the claim is expressly or inherently disclosed in a single prior art reference.”)

1. Claim 1

- a. [1pre] A method of developing a distributed parallel computing program, comprising steps of:**

Whether or not limiting, Fukuda '98 discloses claim [1pre].

As discussed below with respect to claim [1c], Fukuda '98 discloses claim [1pre]'s method of developing a distributed parallel computing program.

- b. [1a] establishing at least one distributed shared variable, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically distributed across multiple memories;**

The '080 patent describes “distributed shared variables”:

In one aspect of navigational programming, a distributed shared variable (“DSV”) may be used. DSVs and self-migration make shared variable programming possible beyond shared memory that provides single-address memory space. A DSV is logically a single variable that includes several variables that may be physically distributed across multiple memories. An example of a DSV is a global array.

Ex.1001 (3:33-39); Ex.1003 ¶ 125.

With respect to MESSENGERS, the '080 patent describes “node variables”:

There are two types of variables in MESSENGERS: agent variables and node variables. An agent variable is private to a particular Messenger and travels with that Messenger as it migrates through the logical network. A node variable is stationary and is accessible by all Messengers currently at the logical node to which the variable belongs. Hence agent variables can be used to carry data between nodes, while node variables can be used for inter-agent communication.

Ex.1001 (9:1-9); Ex.1003 ¶ 126.

During prosecution of the '080 patent, the Examiner, three times, found that Pan (Ex.1005), which describes MESSENGERS, disclosed the “distributed shared variables” of claim [1a] at p. 77, right column, second paragraph, “distributed sequential computing” and in the sentence “The data is distributed in essence . . . the code follows the data (an agent-based approach).” Ex.1002 at 126; 101; and 76. Applicants did not object or otherwise respond to this finding, choosing instead to address other limitations of the claim. *Id.* at 115-117; 89-91; and 58-61.

Fukuda '98 discloses the “distributed shared variable” of claim [1a]. Consistent with the disclosures in the '080 specification and the distributed data of Pan, relied on by the Examiner, Fukuda '98 discloses “node variables,” which are examples of “distributed shared variables”: “Node variables are resident in nodes and shared by all Messengers currently running on the same logical node.” Ex.1008 at 301; Ex.1003 ¶ 127.

Fukuda '98 further describes how the Messengers interact with node variables:

To accomplish their mission, Messengers must generally interact with one another in both time and space. Spatial interaction is accomplished via shared node variables defined in every logical node. Every Messenger script contains a node variables declaration that uses the same style and conventions as C. This provides the necessary mapping between the Messenger code and the actual node-resident variables. Using the declared names, a Messenger can read and write any of the node variables of the node in which it currently resides. Consequently, any collection of Messengers can

interact with each other by hopping to a common node and reading/writing agreed-upon node variables.

Ex.1008 at 302; *see also* 304-305 (describing exemplary node variables “*c_veinal*” and “*n_msgs*”); Ex.1003 ¶ 127.

Fukuda '98 describes an example of a distributed shared variable, *i.e.*, *virtual time*, which is “a global time line consistent over all physical nodes,” that is, “a globally shared variable that simulates the passage of time.” Ex.1008 at 302; Ex.1003 ¶ 128.

Fukuda '98 further discloses that nodes (and therefore node variables) in MESSENGERS are physically distributed across multiple memories. Fukuda '98 discloses that the MESSENGERS system involves a physical network (a LAN or WAN) (Ex.1008 at 294), and discloses that Messengers (which interact with node variables at nodes) “move not only within the simulated space but actually migrate among processors in the network.” Ex.1008 at 306; Ex.1003 ¶ 129. A POSITA would have understood that a LAN (local area network) and a WAN (wide area network) each require physically distinct, multiple memories and processors and therefore Fukuda '98 necessarily disclosed that the node variables are physically distributed across multiple memories. Ex.1003 ¶ 129. This is confirmed by Fukuda '01, which depicts the three networks involved in MESSENGERS (as described in

Fukuda '98 (Ex.1008 at 294) and as described in the '080 patent (Ex.1001 (8:31-39)), in its Figure 1:

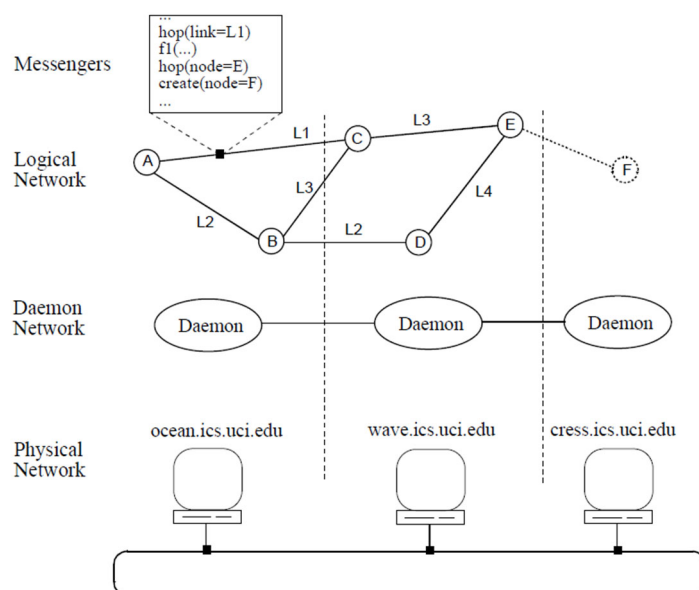


Fig. 1 Network Layers.

Ex.1009 at 97.

Thus, Fukuda '98 (Ex.1008) discloses claim [1a]. Ex.1003 ¶ 131.

- c. [1b] developing at least one distributed sequential computing program to access the at least one distributed shared variable;**

The '080 patent describes “distributed sequential programming” as “computing using a single locus of computation over distributed data, possibly stored in a DSV. A self-migrating thread may be utilized to perform DSC.” Ex.1001 (3:50-54); Ex.1003 ¶ 132. The '080 patent describes an example of a

sequential program that performs block-fashion matrix multiplication. Ex.1001 (6:16-40); Ex.1003 ¶ 132.

During prosecution of the '080 patent, the Examiner, three times, found that Pan (Ex.1005) disclosed the “distributed sequential computing programs” of claim [1b] at p. 77, right column, first paragraph, “distributed the sequential algorithm” and the reference to “re-programming.” Ex.1002 at 126; 101; and 76. Applicants did not object or otherwise respond to this finding, choosing instead to address other limitations of the claim. *Id.* at 115-117; 89-91; and 58-61.

Fukuda '98 discloses “developing at least one distributed sequential computing program to access the at least one distributed shared variable” of claim [1b]. Ex.1008 at 303; Ex.1003 ¶ 133. Fukuda '98 describes a “coordination example” of a biological simulation of the circulatory system: “We illustrate how a distributed simulation can be structured as a collection of functions, residing on different nodes of a network, where the invocation of these functions and the exchange of data among them is orchestrated by Messengers.” Ex.1008 at 303; Ex.1003 ¶ 133.

Fukuda '98 describes this distributed sequential computing program as follows:

Our implementation uses MESSENGERS as a control language to coordinate the operation and interaction of compiled node-resident C functions, which carry out the actual computations of the model.

The basic approach is to map each organ onto a separate node, which contains the necessary sets of constants and differential equations as C functions. The toxin-carrying fluids, such as blood, are implemented as waves of consecutive Messengers, which cycle through the organism along the predefined paths, thus mimicking the actual flow through the body over time. As they pass through the organs, they trigger the execution of the appropriate functions to compute the new concentrations and other values for the current simulated time increment.

Ex.1008 at 304; Ex.1003 ¶ 134.

Fukuda '98 describes the Messenger script for styrene concentration at 304-305 and shows this script in Figure 4 (Ex.1003 ¶¶ 135-136):

```
(1) styrene_sim(n_iter, inhaled_sty) {  
(2)   c_veinal == 0;  
(3)   for (i = 0; i < n_iter; i++) {  
(4)     func(name="new_lung_conc"; in=c_veinal, inhaled_sty; out = c_art);  
(5)     n_msgrs = number_of_organ;  
(6)     hop(link="arterial");  
(7)     func(name="new_organ_conc"; in=c_art; out=c_ven_o);  
(8)     hop(link="veinal");  
(9)     c_veinal += c_ven_o;  
(10)    if (--n_msgrs > 0)  
(11)      exit;  
(12)  }  
(13) }
```

Fig. 4. Messenger script for styrene inhalation.

Thus, Fukuda '98 (Ex.1008) discloses claim [1b]. Ex.1003 ¶ 137.

- d. **[1c] transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one intermediate condition occurs within the at least one distributed sequential computing program**

The '080 patent describes “transforming the at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program when at least one intermediate condition occurs within the at least one distributed sequential computing program.”

The '080 patent describes the sequential program that performs block-fashion matrix multiplication, including describing how, in a distributed environment, “we may use a processor to post-fetch the computed C sub-matrix and pre-fetch the next pair of A and B sub-matrices,” as shown in the DSC program below:

```
(1) for i = 0 to p - 1
(2)   for j = 0 to p - 1
(2.1)     inject (WR (i, j))
(2.2)     waitEvent (IOb(i,j))
(3)       Cij = Ai Bj
(3.1)     hop (!μ)
(4)   end for
(5) end for
```

Ex.1001 (6:33-40); Ex.1003 ¶ 139.

According to the '080 patent, in the above program:

WR() is a separate DSC program that performs fetching. It is “injected,” or spawned, by the computing thread, and it hops to the other node to perform fetching, after which it “signals” an event to indicate that the fetching is done. The computing thread will (2.2) wait for the event to make sure that the data is correctly post- and pre-fetched, (3) compute the sub-matrix multiplication, and (3.1) hop to the other node to perform the computation for the next loop.

Ex.1001 (6:43-51); Ex.1003 ¶ 140.

Note that the DSC program in the '080 patent “injects” “WR()” using an “inject()” statement at line (2.1). Ex.1001 (6:36); Ex.1003 ¶ 141.

Thus, the '080 patent describes how “[t]he computing DSC program and the fetching DSC program together make a DPC program. The two DSC threads of the DPC program run concurrently to perform parallel or pipelined computing. In this particular example, the computations and the fetching are pipelined.” Ex.1001 (6:61-65); Ex.1003 ¶ 141.

Similarly, the '080 patent further discusses how, in MESSENGERS, a “Messenger can spawn another Messenger using a statement, such as inject(). Synchronization among Messengers uses ‘events,’ and statements, such as signalEvent() and waitEvent().” Ex.1001 (9:16-19; *see also* 8:38-43 and 8:50-52); Ex.1003 ¶ 142.

With respect to the portion of claim [1c] pertaining to “at least one intermediate condition occurs within the at least one distributed sequential computing program,” the ’080 patent describes each box of Figures 3(a), 4(a), and 5(a) as representing a computation, wherein “[a] box marked with R means the computation represented by the box produces intermediate result that will be required by the following computation on the next node, whereas a box marked with NR means the computation on the next node is independent of the computation represented by the box.” Ex.1001 (7:47-52); Ex.1003 ¶ 143. In other words, the intermediate condition is the receipt of the intermediate result from part of the computation on the current node:

[T]he computation on the next node is scheduled as soon as the dependency condition allows it. That is, since in the DSC the computation on the next node only depends on the intermediate result from part of the computation on the current node, as shown in FIG. 3(a), the thread can clone itself as soon as the computation of R is done, and have the clone hop to the next node carrying the intermediate result and continue the computation at the same time when it continues the computation of NR on the current node, as shown in FIG. 3(b).

Id.; Ex.1003 ¶ 143.

Fukuda ’98 discloses claim 1[c]. Fukuda ’98 discloses that “a Messenger script may include statements to dynamically link and invoke C functions, precompiled and executed in native-mode on the current node. This permits the

behavior of each Messenger, and hence the functionality of the entire application, to be composed dynamically.” Ex.1008 at 301; Ex.1003 ¶ 144.

As taught by Fukuda '98, two statements provide an interface to executable C functions via the Unix file system. Ex.1008 at 301; Ex.1003 ¶ 145. One of these statements, *exec()*, contains a *main()* function “which is spawned as a separate Unix process, i.e., operating concurrently with the invoking Messenger.” *Id.* In the *exec()* function, shown below, “[t]he arguments specified by *in* are passed to the new process at the time of creation and results are returned to the invoking Messenger via the *out* argument”:

exec(file = filename [; in = argument] [; out = argument;] [nowait]⁴).

Ex.1008 at 301; Ex.1003 ¶ 145.

The distributed simulation example in Fukuda '98 demonstrates claim [1c]. As Fukuda '98 describes, the “[t]he corresponding Messenger is *injected* into the lung node.” Ex.1008 at 304 (emphasis added); Ex.1003 ¶ 146. Consistent with the '080 specification, “injecting” is the same as “spawning.” Ex.1001 (6:44-45); Ex.1003 ¶ 146. Therefore, “injecting” the Messenger into the lung node is “spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program,” *i.e.*, the distributed simulation

⁴ “Nowait” is optional. Ex.1008 at 301.

is a distributed sequential computing program and the Messenger that is “injected” that runs the script shown in Figure 4 for the styrene inhalation is also a distributed sequential computing program⁵, which, together, transforms the distributed simulation to a distributed parallel computing program. The intermediate condition occurs when, within the distributed simulation, each cycle begins, as described by Fukuda ’98. Ex.1008 at 304-305; Ex.1003 ¶ 146.

Thus, consistent with the ’080 patent specification, Fukuda ’98 discloses the transformation of at least one distributed sequential computing program into at least one distributed parallel computing program by spawning at least one child distributed sequential computing program from the at least one distributed sequential computing program (*i.e.*, using, for example, the *exec()* statement which contains a *main()* function (Ex.1008 at 301) or using the “inject” statement (Ex.1008 at 304)⁶) when at least one intermediate condition occurs within the at least one distributed sequential computing program (*e.g.*, the arguments specified by *in*, as disclosed in Fukuda ’98 (Ex.1008 at 301) or the parameters to be passed

⁵ As disclosed by Fukuda ’98, Messengers are programs: “MESSENGERS is a paradigm for the programming of distributed systems. It is based on the principles of autonomous messages, called Messengers, which carry their own behavior in the form of a **program**.” Ex.1008 at 291 (emphasis added); Ex.1003 ¶ 147.

⁶ A POSITA would have confirmed, from Ex.1009 at 99, that the “inject” statement in MESSENGERS “initiates (injects) the execution of one or more instances of another Messenger,” as discussed in Section X.B., *supra*.

to a newly created Messenger when invoking the “inject” statement (Ex.1008 at 304)⁷); Ex.1003 ¶ 148.

Thus, Fukuda '98 (Ex.1008) discloses claim [1c]. Ex.1003 ¶ 149.

- e. **[1d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,**

As discussed, the '080 patent describes how, when the separate DSC program, WR() that performs fetching is “injected” or “spawned,” “[t]he computing DSC program and the fetching DSC program together make a DPC program. The two DSC threads of the DPC program run *concurrently* to perform parallel or pipelined computing. In this particular example, the computations and the fetching are pipelined.” Ex.1001 (6:61-65) (emphasis added); Ex.1003 ¶ 151.

Similarly, Fukuda '98 discloses that *exec()*, contains a *main()* function, “which is spawned as a separate Unix process, i.e., operating *concurrently* with the invoking Messenger.” Ex.1008 at 301 (emphasis added); Ex.1003 ¶ 152.

⁷ A POSITA would have confirmed, from Ex.1009 at 99, that, when invoking the “inject” statement in MESSENGERS, parameters are passed to the newly created Messenger: “[t]he parameters specify the Messenger script to be injected and the initial parameters to be passed to the newly created Messenger,” as discussed in Section X.B., *supra*.

Fukuda '98 further discloses the distributed simulation of where Messengers are running concurrently, as shown in Figure 3 (e.g., the messengers for C_ven_f, C_ven_m, C_ven_r, and C_ven_l are shown to be running concurrently):

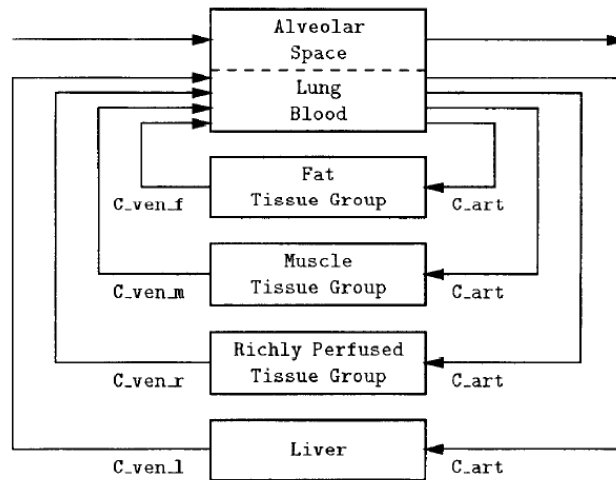


Fig. 3. A toxicology model.

Ex.1008 at 304; Ex.1003 ¶ 153.

Thus, Fukuda '98 (Ex.1008) discloses claim [1d]. Ex.1003 ¶ 154.

- f. [1e] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.

The '080 patent discloses “wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.”

As discussed above, the '080 patent describes each box of Figures 3(a), 4(a), and 5(a) as representing a computation, wherein “[a] box marked with R means the computation represented by the box produces intermediate result that will be

required by the following computation on the next node, whereas a box marked with NR means the computation on the next node is independent of the computation represented by the box.” Ex.1001 (7:47-52); Ex.1003 ¶ 155. In other words, the intermediate condition is the receipt of the intermediate result from part of the computation on the current node:

[T]he computation on the next node is scheduled as soon as the dependency condition allows it. That is, since in the DSC the computation on the next node only depends on the intermediate result from part of the computation on the current node, as shown in FIG. 3(a), the thread can clone itself as soon as the computation of R is done, and have the clone hop to the next node carrying the intermediate result and continue the computation at the same time when it continues the computation of NR on the current node, as shown in FIG. 3(b).

Ex.1001 (7:54-63); Ex.1003 ¶ 155.

Fukuda '98 discloses claim 1[e]. As taught by Fukuda '98, two statements provide an interface to executable C functions via the Unix file system. Ex.1008 at 301; Ex.1003 ¶ 156. One of these statements, *exec()*, contains a *main()* function, “which is spawned as a separate Unix process, i.e., operating concurrently with the invoking Messenger.” *Id.*; Ex.1003 ¶ 156.

In the *exec()* function, shown below, “[t]he arguments specified by *in* are passed to the new process at the time of creation and results are returned to the invoking Messenger via the *out* argument.”

exec(file = filename [; in = argument] [; out = argument;] [nowait]⁸).

Ex.1008 at 301; Ex.1003 ¶ 157.

Further, the distributed simulation example in Fukuda '98 demonstrates claim [1e]. Ex.1003 ¶ 158. As Fukuda '98 describes, the “[t]he corresponding Messenger is *injected* into the lung node.” Ex.1008 at 304 (emphasis added); Ex.1003 ¶ 158. The intermediate condition occurs when, within the distributed simulation, each cycle begins, as described by Fukuda '98, and ends when the *nmsgsr*s variable decrements. Ex.1008 at 304-305; Ex.1003 ¶ 158.

Thus, consistent with the '080 patent specification, Fukuda '98 discloses the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation (*e.g.*, using, for example, the *exec()* statement which contains a *main()* function) when at least one intermediate condition also comprises an intermediate result (*e.g.*, the arguments specified by *out*). Ex.1003 ¶ 159. This also occurs when invoking the “inject” statement, as shown in the example in Fukuda '98, where the injected Messenger returns to the lung node after adding the *c_ven_o* value to the cumulative node variable *c_veinal* and the shared variable *n_msgsr*s in the lung node decrements. Ex.1008 at 304-305; Ex.1003 ¶ 160.

⁸ “Nowait” is optional. Ex.1008 at 301.

Thus, Fukuda '98 (Ex.1008) discloses claim [1e]. Ex.1003 ¶ 160.

Fukuda '98 therefore anticipates claim 1. Ex.1003 ¶ 161.

- 2. Claim 2: The method of claim 1, wherein each of the at least one distributed sequential computing program includes at least one self-migrating thread, capable of migrating among at least one processor and capable of accessing the at least one distributed shared variable.**

The '080 patent describes “self-migrating threads or tasks” as “threads or tasks that can migrate from one processor to another,”⁹ which “may be in the form of mobile agents with strong mobility.” Ex.1001 (3:16-19); Ex.1003 ¶ 163.

Fukuda '98 discloses that the Messengers described therein are “a self-contained object, consisting of a program and current state, including a program counter and local variables. The program, which is a combination of interpreted MESSENGERS code and native C code, describes the Messenger’s behavior, which allows it to navigate autonomously through the underlying communication network and carry out computation in the nodes it visits.” Ex.1008 at 293; Ex.1003 ¶ 164. Messengers are “autonomous messages,” “which carry their own behavior

⁹ Petitioner notes the inconsistency in claim 2, which requires that the “at least one self-migrating thread” be “capable of migrating among at least one processor,” and the definition in the '080 specification that self-migrating threads are “threads or tasks that can migrate from one processor to another.” Ex.1001 (3:16-18). The fact that Petitioner is addressing claim 2 herein should not be taken as an admission that claim is not indefinite, as it can be applied in instances where the self-migrating thread migrates among two or more processors. Petitioner reserves all rights to contend in the district court litigation that this inconsistency renders claim 2 indefinite.

in the form of a program” (*Id.* at 291) and are “capable of self-migration, i.e., autonomous navigation through both the logical and the physical networks” (*Id.* at 306). Ex.1003 ¶ 164.

Fukuda '98 further discloses that the self-migration capability of Messengers means that “the individual entities move not only within the simulated space but actually migrate among processors in the network.” Ex.1008 at 306; Ex.1003 ¶ 165. “This occurs when an entity crosses the boundary of a logical region. The corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*

Fukuda '98 further discloses that the Messengers are capable of accessing a distributed shared variable: “[n]ode variables are resident in nodes and shared by all Messengers currently running on the same logical node.” Ex.1008 at 301; Ex.1003 ¶ 166. “[A] Messenger can read and write any of the node variables of the node in which it currently resides. Consequently, any collection of Messengers can interact with each other by hopping to a common node and reading/writing agreed-upon node variables.” Ex.1008 at 302; Ex.1003 ¶ 166.

Fukuda '98 provides an example of a Messenger accessing distributed variables *c_veinal* and *n_msgrs*: “The circulation is accomplished using the two *hop* statements; the first (line 6) replicates the Messenger to all organ nodes, where

the corresponding *new-erg-cone* computes the new styrene concentration; the second *hop* statement (line 8) then takes the Messenger back to the lung node along the veinal links, where it adds its *c_ven_o* value (which represents the organ's contribution to the incremental change in the veinal styrene concentration) to the cumulative node variable *c_veinal* (line 9). To ensure that all Messengers have arrived at the lung node before the next cycle begins, the shared node variable *n_msgrs* in the lung node is used as a synchronization counter.” Ex.1008 at 304-305; *see also* 302 describing “*virtual time*” (*i.e.*, “a globally shared variable that simulates the passage of time”); Ex.1003 ¶ 167.

Thus, Fukuda '98 (Ex.1008) discloses claim 2. Ex.1003 ¶ 168.

3. Claim 3: The method of claim 1, wherein the at least one distributed shared variable is configured to be located among a plurality of memories.

As discussed with respect to claim [1b], *supra*, Fukuda '98 describes an example of a distributed shared variable, *i.e.*, *virtual time*, which is “a global time line consistent over all physical nodes” which is “a globally shared variable that simulates the passage of time.” Ex.1008 at 302; Ex.1003 ¶ 171.

As Messengers can migrate to nodes on different processors (as described at Ex.1008 at 306), the global variable “*virtual time*” must be located among a plurality of memories. Ex.1003 ¶ 172.

Thus, Fukuda '98 (Ex.1008) discloses claim 3. Ex.1003 ¶ 173.

4. Claim 4: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple processors.

As discussed with respect to claim 2, *supra*, Fukuda '98 discloses that the distributed parallel program is configured to operate across multiple processors: “the individual entities move not only within the simulated space but actually migrate among processors in the network.” Ex.1008 at 306; Ex.1003 ¶ 175. “This occurs when an entity crosses the boundary of a logical region. The corresponding Messenger performs a hop statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*; Ex.1003 ¶ 175.

Thus, Fukuda '98 (Ex.1008) discloses claim 4. Ex.1003 ¶ 176.

5. Claim 5: The method of claim 1, wherein the distributed parallel computing program is configured to operate using multiple threads.

Fukuda '98 discloses that the distributed parallel computing program is configured to operate using multiple Messengers (*i.e.*, threads). Ex.1003 ¶ 178.

Messengers are “autonomous messages,” “which carry their own behavior in the form of a program” (Ex. 1008 at 291) and are “capable of self-migration, *i.e.*, autonomous navigation through both the logical and the physical networks” (*Id.* at 306). Ex.1003 ¶ 179.

Fukuda '98 discloses multiple Messengers: “To accomplish their mission, Messengers must generally interact with one another in both time and space.

Spatial interaction is accomplished via shared node variables defined in every logical node. Every Messenger script contains a node variables declaration that uses the same style and conventions as C. This provides the necessary mapping between the Messenger code and the actual node-resident variables. Using the declared names, a Messenger can read and write any of the node variables of the node in which it currently resides. Consequently, any collection of Messengers can interact with each other by hopping to a common node and reading/writing agreed-upon node variables.” Ex.1008 at 302; Ex.1003 ¶ 180.

Fukuda '98 further discloses an example that uses multiple Messengers, as shown in Figure 3 and described with respect to Figure 4: “To ensure that all Messengers have arrived at the lung node before the next cycle begins, the shared node variable *n_msgrs* in the lung node is used as a synchronization counter. At the beginning of each cycle it is set to the number of organs (line 5) thus counting the number of Messenger replicas.” Ex.1008 at 304-305; Ex.1003 ¶ 181.

Thus, Fukuda '98 (Ex.1008) discloses claim 5. Ex.1003 ¶ 182.

6. Claim 6: The method of claim 1, wherein the distributed parallel computing program is configured to operate across multiple nodes.

Fukuda '98 discloses that the distributed parallel computing program is configured to operate across multiple nodes. Ex.1003 ¶ 184.

For example, Fukuda '98 discloses that “The program, which is a combination of interpreted MESSENGERS code and native C code, describes the Messenger’s behavior, which allows it to navigate autonomously through the underlying communication network and carry out computation in the *nodes* it visits.” Ex.1008 at 293 (emphasis added); *see also* 292: “Messengers are able to navigate freely and autonomously through the logical network. They migrate to nodes containing data that they wish to read, write, or modify, and to nodes containing functions that they wish to invoke.” (emphasis added); Ex.1003 ¶ 184.

Thus, Fukuda '98 (Ex.1008) discloses claim 6. Ex.1003 ¶ 185.

7. Claim 7: The method of claim 1, further comprising the step of configuring the at least one distributed sequential computing program to include at least one mobile agent.

Fukuda '98 discloses that the at least one distributed sequential computing program includes at least one mobile agent. Ex.1003 ¶ 187.

Fukuda '98 discloses that Messengers are self-contained autonomous programs that have their own identity and behavior: “Notably, autonomous navigation is used by a number of systems to create “mobile agents” (see Section 4). While their primary purpose is to roam the Internet, MESSENGERS uses navigation as a primary structuring tool, which, in conjunction with dynamic functional composition, yields a powerful coordination paradigm for general-purpose distributed computing.” Ex.1008 at 294; Ex.1003 ¶ 188.

The '080 patent confirms that Messengers are mobile agents: “An example of such a system is MESSENGERS, in which applications are developed as collections of *mobile agents, which may be referred to as Messengers.*” Ex.1001 (8:29-31); Ex.1003 ¶ 187.

Thus, Fukuda '98 (Ex.1008) discloses claim 7. Ex.1003 ¶ 189.

8. Claim 8: The method of claim 7, wherein the at least one mobile agent is implemented using self-migrating threads.

As discussed with respect to claim 2, *supra*, the Messengers (*i.e.*, mobile agents) are “autonomous messages,” “which carry their own behavior in the form of a program” (Ex. 1008 at 291) and are “capable of *self-migration*, *i.e.*, autonomous navigation through both the logical and the physical networks” (*Id.* at 306) (emphasis added). Ex.1003 ¶ 191. Further, as discussed in connection with claim 7, mobile agents are implemented using self-migrating threads. Ex.1003 ¶ 192.

Thus, Fukuda '98 (Ex.1008) discloses claim 8. Ex.1003 ¶ 193.

9. Claim 9

a. [9pre]: A distributed parallel computing system, having at least one memory area and at least one processor, comprising:

Whether or not limiting, Fukuda '98 discloses claim [9pre].

As discussed below with respect to claim [9c], Fukuda '98 discloses distributed parallel computing system. Ex.1003 ¶ 195.

As described in the Abstract of Fukuda '98, MESSENGERS is a distributed parallel computing system: “MESSENGERS is a paradigm for the programming of distributed systems.” EX1008. Abstract; Ex.1003 ¶ 195. Because the Messengers operate on different processors simultaneously, MESSENGERS is a distributed parallel computing system. Ex.1003 ¶ 195.

196. Fukuda '98 discloses at least one memory area: “In MESSENGERS. the programmer’s view is navigational, concerned with how to move through the logical network, and when to interact with the data or functions stored at a node.” *Id.*, 293; Ex.1003 ¶ 196. Data or functions cannot be stored at a node unless the node has a memory area. Ex.1003 ¶ 196. A function cannot be executed at a node unless the node has a processor. *Id.*

197. Further, “An important implication of the MESSENGERS paradigm is that the individual entities move not only within the simulated space but actually migrate among processors in the network.” *Id.*, 306; Ex.1003 ¶ 197.

- b. [9a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area;**

Fukuda '98 discloses the “distributed shared variable” of claim [9a]. Limitation [9a] is essentially the same as limitation [1a], but with the addition of references to “loading.” Ex,1003 ¶ 199. A variable cannot be “physically

distributed across multiple memories, as in [1a] unless it is “loaded” into those memories. *Id.*

Thus, Fukuda '98 (Ex.1008) discloses claim [9a]. Ex.1003 ¶ 200.

- c. [9b] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable, and**

Limitation [9b] is essentially the same as limitation [1b], but with the addition of reference to a “processor.” Ex.1003 ¶ 201. A distributed shared variable (as in [1b]) cannot be accessed unless the distributed sequential computing program (as in [1b]) is configured to operate in a processor. *Id.*

Thus, Fukuda '98 (Ex.1008) discloses claim [9b]. Ex.1003 ¶ 202.

- d. [9c] configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program,**

Fukuda '98 discloses the “at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable” of claim [9c]. Ex.1003 ¶ 203.

Limitation [9c] is identical to limitation [1c], except that the phrase “transforming the at least one distributed sequential computing program” has been

replaced by “configured to transform.” Ex.1003 ¶ 203. This is simply a system form of method step [1c]. *Id.*

Thus, Fukuda ’98 (Ex.1008) discloses claim [9c]. Ex.1003 ¶ 204.

- e. **[9d] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,**

Limitation [9d] is literally identical to limitation [1d]. Ex.1003 ¶ 205.

Thus, Fukuda ’98 (Ex.1008) discloses claim [9d]. Ex.1003 ¶ 206.

- f. **[9e] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.**

Limitation [9e] is literally identical to limitation [1e]. Ex.1003 at ¶ 207

Thus, Fukuda ’98 (Ex.1008) discloses claim [9e]. Ex.1003 ¶ 208.

- 10. Claim 10: The system of claim 9, wherein the at least one distributed sequential computing program comprises at least one mobile agent.**

Claim 7 recites “configuring the at least one distributed sequential computing program to include at least one mobile agent,” which is described *supra*. Thus, Fukuda ’98 (Ex.1008) discloses claim 10. Ex.1003 ¶ 211.

11. Claim 11: The system of claim 10, wherein the at least one mobile agent is implemented using self-migrating threads.

As discussed with respect to claims 2 and 8, *supra*, the Messengers (*i.e.*, mobile agents) are “autonomous messages,” “which carry their own behavior in the form of a program” (Ex. 1008 at 291) and are “capable of self-migration, *i.e.*, autonomous navigation through both the logical and the physical networks” (*Id.* at 306); Ex.1003 ¶ 213.

Thus, Fukuda '98 (Ex.1008) discloses claim 11. Ex.1003 ¶ 214.

12. Claim 12: The system of claim 10, wherein the at least one mobile agent is an explicit-navigation mobile agent.

The '080 patent describes explicit-navigation mobile agents: “In another aspect of navigational programming, an explicit-navigation mobile agent system may be built on top of a DSM. A DSM system may be used to serve as communication media for the globally accessible data, or in other words, agent variables.” Ex.1001, 10:18-22; Ex.1003 ¶ 216. The '080 patent further states that “[s]elf-migrating threads may be in the form of mobile agents.” *Id.* (8:21-22); Ex.1003 ¶ 217.

Fukuda '98 discloses explicit-navigation mobile agents by disclosing that the source code that handles the explicit migration of threads is the hop() command: “**The *hop* Statement.** The *hop* statement permits a Messenger to move around the

logical network and is based directly on the navigational calculus.” Ex.1008 at 297; *see also* 298; 304-305 (Figure 4); Ex.1003 ¶ 218.

Thus, Fukuda ’98 (Ex.1008) discloses claim 12. Ex.1003 ¶ 219.

13. Claim 13: The system of claim 10, wherein the at least one mobile agent is an implicit-navigation mobile agent.

The ’080 Patent explains what an implicit-navigation mobile agent is: “These self-migrating threads may be programmed . . . implicitly, e.g., driven by data distribution. With this approach, a thread, M, may perform its computations on one processor, Processor 1, and if the thread M needs to access memory belonging to another processor, e.g., Processor 2, it may suspend its computations, migrate, or move, to the other processor, Processor 2, and then resume its computations.” Ex.1001, 3:20-28; Ex.1003 ¶ 221.

Further, “In another aspect of navigational programming, an implicit-navigation mobile agent system may be based on a DSM. In this system, a mechanism is included that decides for the agent when and where to migrate, and how the migration is achieved. FIG. 8 depicts a functional design of an implicit navigation agent system.” *Id.*, 10:36-41; Ex.1003 ¶ 222.

Fukuda ’98 discloses implicit-navigation mobile agents: “In MESSENGERS, multiple destination nodes may be specified. The specification

can be made *implicitly*, using the navigational calculus described in Section 2.3.”

Ex.1008 at 308 (emphasis added); Ex.1003 ¶ 223.

Thus, Fukuda '98 (Ex.1008) discloses claim 13. Ex.1003 ¶ 224.

14. Claim 14: The system of claim 10, wherein the at least one mobile agent is configured to move from one processor to another processor.

As discussed with respect to claims 2 and 4, *supra*, Fukuda '98 discloses that the distributed parallel program is configured to operate across multiple processors: “the individual entities move not only within the simulated space but actually migrate among processors in the network.” Ex.1008 at 306; Ex.1003 ¶ 226. “This occurs when an entity crosses the boundary of a logical region. The corresponding Messenger performs a *hop* statement, which causes it to automatically migrate into the processor responsible for the corresponding logical region.” *Id.*; Ex.1003 ¶ 226. The '080 patent discloses that a Messenger can be a mobile agent: “An example of such a system is MESSENGERS, in which applications are developed as collections of mobile agents, which may be referred to as Messengers.” Ex.1001, 8:29-31; Ex.1003 at ¶ 226.

Thus, Fukuda '98 (Ex.1008) discloses claim 14. Ex.1003 ¶ 227.

15. Claim 15: The system of claim 10, wherein the at least one mobile agent operates within a distributed shared memory system.

As described in Fukuda '98, MESSENGERS is a distributed shared memory system (at least because it uses “at least one distributed shared variable” per limitation 1[b]). Further, in MESSENGERS, “applications are developed as collections of mobile agents, which may be referred to as Messengers.” Ex.1001, 8:29-31; Ex.1003 ¶ 229.

Thus, Fukuda '98 (Ex.1008) discloses claim 15. Ex.1003 ¶ 230.

16. Claim 16: The system of claim 9, wherein the at least one processor is located across a physical network.

Fukuda '98 discloses that “[a] computation under MESSENGERS involves the dynamic construction of a logical network that is mapped onto the underlying physical network,” where the physical network is “the lowest level network (a LAN or WAN), which constitutes the underlying computational resource.” Ex.1008 at 293, 294; *see also* Ex.1009 at 98-99; Ex.1003 ¶ 232.

Thus, Fukuda '98 (Ex.1008) discloses claim 16. Ex.1003 ¶ 233.

17. Claim 17: The method of claim 1, wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.

As described in Fukuda '98, MESSENGERS is a distributed shared memory system (at least because it uses “at least one distributed shared variable” per

limitation 1[b]). Ex.1003 ¶ 235. MESSENGERS includes a distributed parallel computing program, as shown in connection with limitation 1[c]. *Id.*

Thus, Fukuda '98 (Ex.1008) discloses claim 17. Ex.1003 ¶ 236.

18. Claim 18: The method of claim 9, wherein the distributed parallel computing program is configured to operate across a distributed shared memory system.

See discussion for claim 17, *supra*.

Thus, Fukuda '98 (Ex.1008) discloses claim 17. Ex.1003 ¶ 239.

XII. CONCLUSION

Petitioners respectfully request institution of IPR and cancellation of the Challenged Claims based on the ground set forth and described above.

Dated: September 22, 2025

Respectfully submitted,

McKOOL SMITH, P.C.
ATTN: John B. Campbell
303 Colorado Street
Suite 2100
Austin, TX 78701
Tel: (512) 692-8730
Fax: (512) 692-8744

By: /s/ John B. Campbell
John B. Campbell
Reg. No. 54,665
Counsel for Petitioner

CERTIFICATE OF COMPLIANCE WITH WORD COUNT

Pursuant to 37 C.F.R. § 42.24(d), I certify that this petition complies with the type-volume limits of 37 C.F.R. § 42.24(a)(1)(i) because it contains 10,342 words, according to the word-processing system used to prepare this petition, excluding parts of this petition that are exempted by 37 C.F.R. § 42.24(a) (including the table of contents, a table of authorities, mandatory notices, a certificate of service or this certificate word count, appendix of exhibits, and claim listings).

Dated: September 22, 2025

McKOOL SMITH, P.C.
ATTN: John B. Campbell
303 Colorado Street
Suite 2100
Austin, TX 78701
Tel: (512) 692-8730
Fax: (512) 692-8744

By: /s/ John B. Campbell
John B. Campbell
Reg. No. 54,665

CERTIFICATE OF SERVICE

I hereby certify, pursuant to 37 C.F.R. Sections 42.6 and 42.106, that a complete copy of the attached **PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 7,712,080 B2**, including all exhibits (**Nos. 1001-1013**) and related documents, are being served by federal express on 22nd day of September, 2025, the same day as the filing of the above-identified document in the United States Patent and Trademark Office/Patent Trial and Appeal Board, upon Patent Owner by serving the correspondence address of record with the USPTO as follows:

Orrick, Herrington & Sutcliff,
Attn: Kenneth S. Roberts,
4 Park Plaza, Suite 1600,
Irvine, California 92614-2558

A courtesy copy was also sent via electronic mail to the Patent Owner's litigation counsel listed below:

Jonathan K. Waldrop
jwaldrop@kasowitz.com
Darcy L. Jones
djones@kasowitz.com
Marcus A. Barber
mbarber@kasowitz.com
John W. Downing
jdowning@kasowitz.com
Heather S. Kim
hkim@kasowitz.com
ThucMinh Nguyen
tnguyen@kasowitz.com
Jonathan H. Hicks
Jhicks@kasowitz.com
KASOWITZ BENSON TORRES LLP
333 Twin Dolphin Drive, Suite 200
Redwood Shores, California 94065

Dated: September 22, 2025

/s/ John B. Campbell

John B. Campbell

Reg. No. 54,665