

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Mark Henrik Sandstrom
U.S. Patent No.: 11,036,556 Attorney Docket No.: 42342-0121IP1
Issue Date: June 15, 2021
Appl. Serial No.: 17/195,174
Filing Date: March 8, 2021
Title: CONCURRENT PROGRAM EXECUTION OPTIMIZATION

Mail Stop Patent Board

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

**PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT
NO. 11,036,556 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42**

TABLE OF CONTENTS

I.	REQUIREMENTS FOR IPR UNDER 37 C.F.R. §42.104.....	1
A.	Standing.....	1
B.	Challenge and Relief Requested	1
II.	SUMMARY OF THE '556 PATENT.....	2
A.	Brief Description	2
B.	Prosecution History	5
III.	LEVEL OF ORDINARY SKILL	6
IV.	CLAIM CONSTRUCTION UNDER 37 C.F.R. §§ 42.104(b)(3)	7
V.	THE CHALLENGED CLAIMS ARE UNPATENTABLE.....	7
A.	[Ground 1A]: Kupferschmidt in view of Tuan (Claims 1, 2, and 4-8).7	
1.	Overview of Kupferschmidt	7
2.	Overview of Tuan	10
3.	Obviousness Based on Kupferschmidt-Tuan Combination.....	11
(a)	Overview of the Kupferschmidt-Tuan Combination.....	11
(b)	Mapping of the Kupferschmidt-Tuan Combination to Claims 1, 2, and 4-8.....	18
B.	[Ground 1B]: Kupferschmidt in view of Tuan and Brent (Claims 1-8)	63
1.	Overview of Brent.....	63
2.	Obviousness Based on Kupferschmidt-Tuan-Brent Combination.....	64
(a)	Overview of the Kupferschmidt-Tuan-Brent Combination	64
(b)	Mapping of the Kupferschmidt-Tuan-Brent Combination to Claims 1-8.....	66
C.	[Ground 1C]: Kupferschmidt in view of Tuan and Sandstrom-501 (Claim 4).....	67
1.	Overview of Sandstrom-501	67

2.	Obviousness Based on Kupferschmidt-Tuan-Sandstrom-501 Combination.....	69
D.	[Ground 1D]: Kupferschmidt in view of Tuan, Brent, and Sandstrom-501 (Claim 4).....	71
VI.	PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION.....	72
A.	§ 314(a) - <i>Fintiv</i>	72
B.	§ 325(d) – <i>Advanced Bionics</i>	74
VII.	MANDATORY NOTICES UNDER 37 C.F.R § 42.8(a)(1).....	76
A.	Real Party-In-Interest Under 37 C.F.R. § 42.8(b)(1)	76
B.	Related Matters Under 37 C.F.R. § 42.8(b)(2)	76
C.	Lead And Back-Up Counsel Under 37 C.F.R. § 42.8(b)(3)	76
D.	Service Information.....	77
VIII.	PAYMENT OF FEES – 37 C.F.R. § 42.103.....	77
IX.	CONCLUSION.....	78

EXHIBITS

MS-1001	U.S. Patent No. 11,036,556 (“the ’556 Patent”)
MS-1002	Excerpts of the File History of the ’556 Patent
MS-1003	Expert Declaration of Jeffrey Chase, Ph.D.
MS-1004	U.S. Pat. App. Pub. No. 2010/0333099 (“Kupferschmidt”)
MS-1005	Vu Manh Tuan, “A Study on a Multitasking Environment for Dynamically Reconfigurable Processors,” Doctoral Dissertation, School of Science for Open and Environmental Systems Graduate School of Science and Technology, Keio University (2009) (“Tuan”)
MS-1006	U.S. Pat. App. Pub. No. 2010/0131955 (“Brent”)
MS-1007	Declaration of June Ann Munford
MS-1008	[RESERVED]
MS-1009	U.S. Patent App. Pub. No. 2014/0245262 (“Hill”)
MS-1010	U.S. Patent App. Pub. No. 2012/0079501 (“Sandstrom-501”)
MS-1011	Excerpts of Phillip A. Laplante, DICTIONARY OF COMPUTER SCIENCE, ENGINEERING, AND TECHNOLOGY, CRC Press (2001)
MS-1012	Excerpts of Alan Freedman, COMPUTER DESKTOP ENCYCLOPEDIA, 9 th Ed. (2001)
MS-1013	Excerpts of McGraw-Hill Dictionary of Scientific and Technical Terms, Sixth Edition (2003)
MS-1014	Excerpts of Random House Concise Dictionary of Science and Computers (2004)

MS-1015	Excerpts of A Dictionary of Computing, Oxford University Press, Sixth Edition (2008)
MS-1016	Excerpts of The Facts on File Dictionary of Computer Science, Revised Edition (2006)
MS-1017	U.S. Pat. No. 5,542,055 (“Amini”)
MS-1018	U.S. Pat. No. 4,647,123 (“Chin”)
MS-1019	Jessica Scarpati, Definition of “host” (in computing), https://www.techtarget.com/searchnetworking/definition/host (retrieved September 18, 2022)
MS-1020	U.S. Pat. App. Pub. No. 2008/0244126 (“Hundley”)
MS-1021	U.S. Pat. No. 8,055,872 (“Biles”)
MS-1022	U.S. Pat. App. Pub. No. 2005/0044344 (“Stevens”)
MS-1023	Muhammad Yasir, Introduction to FPGA Technology (May 12, 2011), <i>available at</i> https://www.fpgarelated.com/showarticle/17.php (retrieved September 27, 2022)
MS-1024	Microsoft Timeline Webpage: First Version of Windows Announced, <i>available at</i> https://news.microsoft.com/announcement/first-version-of-windows-announced/?return=https%3A%2F%2Fnews.microsoft.com%2Fabout%2F (retrieved September 28, 2022)
MS-1025	Dror G. Feitelson, Job Scheduling in Multiprogrammed Parallel Systems, IBM Research Report, Extended Version (Second Rev. August 1997)
MS-1026	U.S. Pat. No. 9,632,833 (“’833 Patent”)
MS-1027 – MS-1100	[RESERVED]

- MS-1101 Complaint for Patent Infringement, *ThroughPuter, Inc. v. Microsoft Corp.*, Case No. 2:22-cv-00492-BJR (April 13, 2022)
- MS-1102 First Amended Complaint for Patent Infringement, *ThroughPuter, Inc. v. Microsoft Corp.*, Consolidated Case Nos. 2:22-cv-00344-BJR and 2:22-cv-00492-BJR (June 3, 2022)
- MS-1103 Order Staying Case Pending *Inter Partes* Review, *ThroughPuter, Inc. v. Microsoft Corp.*, Case No. 2:22-cv-00344-BJR (August 18, 2022)
- MS-1104 Complaint for Patent Infringement, *ThroughPuter, Inc. v. Microsoft Corp.*, Case No. 1-21-cv-00397 (March, 31, 2021)

CLAIM LISTING

Element	Claim Language
[1.P]	A method performed in a data processing system, the method comprising:
[1.1]	receiving, by hardware logic and/or software logic, requests to perform different tasks on behalf of instances of a plurality of programs managed by a data processing system;
[1.2]	identifying, by the hardware logic and/or software logic for each of the instances, communication interdependencies between different processing stages of a set of processing stages of the respective instance;
[1.3]	based on conditions in the data processing system, dynamically varying, by the hardware logic and/or software logic, structures of field-programmable gate arrays used to process different tasks of the instances of the plurality of programs, the structures being dynamically varied by
[1.4]	identifying available field-programmable gate arrays of the data processing system that are available to process different processing stages of requesting instances of respective programs,
[1.5]	based at least on the conditions in the data processing system, identifying selected field-programmable gate arrays from the available field-programmable gate arrays to execute the different processing stages of the requesting instances of the respective programs,
[1.6]	configuring the selected field-programmable gate arrays to process a respective processing stage of a respective requesting instance, and
[1.7]	configuring certain selected field-programmable gate arrays to support communicating, by the task executing on the respective field-programmable gate array, final results to a requesting client over a network in the data processing system.

Element	Claim Language
[2]	The method of claim 1, further comprising configuring a portion of the selected field-programmable gate arrays to communicate intermediate results of the different tasks over the network to the certain selected field-programmable gate arrays, the certain selected field-programmable gate arrays using the intermediate results to produce the final results.
[3]	The method of claim 1, further comprising: maintaining availability information identifying availability of the field-programmable gate arrays; and modifying the availability information to indicate that a particular field-programmable gate array is in use.
[4]	The method of claim 1, wherein the conditions comprise changes in demand expressions of the plurality of programs.
[5]	The method of claim 1, wherein, for at least one program of the plurality of programs, configuring the selected field-programmable gate arrays results in configuring two or more of the selected field-programmable gate arrays as two or more parallel copies of a given task of the at least one program as a parallelized processing stage.
[6]	The method of claim 5, wherein dynamically varying the structures of the field-programmable gate arrays comprises repeatedly configuring the structures for a plurality of iterations to dynamically adjust performance of the different tasks based at least on the conditions.
[7]	The method of claim 5, wherein repeatedly configuring the structures comprises, for at least one iteration of the plurality of iterations, deactivating a given copy of the two or more parallel copies in the parallelized processing stage while remaining copies of the two or more parallel copies continue executing the parallelized processing stage, wherein the given copy is deactivated based at least on the conditions.
[8]	The method of claim 7, wherein the conditions comprise a decrease in processing load destined for the at least one program.

Microsoft Corporation (“Petitioner” or “Microsoft”) petitions for *Inter Partes* Review of claims 1-8 (“Challenged Claims”) of U.S. Patent No. 11,036,556 (“the ’556 Patent”).

I. REQUIREMENTS FOR IPR UNDER 37 C.F.R. §42.104

A. Standing

Microsoft certifies that the ’556 Patent is available for IPR. The present Petition is being filed within one year of service of a complaint against Microsoft asserting the ’556 Patent. *See* MS-1101 (complaint served April 19, 2022). Microsoft is not barred or estopped from requesting this review challenging the Challenged Claims on the below-identified grounds.

B. Challenge and Relief Requested

Microsoft requests IPR of the Challenged Claims on Grounds 1A-1D below.

Additional explanation and support for Grounds 1A-1D is set forth in the expert Declaration of Jeffrey Chase, Ph.D. (MS-1003), referenced throughout this Petition.

Ground	Claims	Basis for Rejection (35 U.S.C. § 103)
1A	1-2, 4-8	Kupferschmidt in view of Tuan
1B	1-8	Kupferschmidt in view of Tuan and Brent
1C	4	Kupferschmidt in view of Tuan and Sandstrom-501
1D	4	Kupferschmidt in view of Tuan, Brent, and Sandstrom-501

The '556 patent indicates priority through a chain of applications to a provisional application filed on August 23, 2013. Without acquiescence, Petitioner treats August 23, 2013 as the earliest effective filing date (“Critical Date”) of the Challenged Claims for purposes of this IPR. Each of the prior art references applied in Grounds 1A-1D qualifies as prior art to the '556 Patent on at least the bases shown below:

Reference	Filed	Published	35 U.S.C. §102 Prior Art Basis
Kupferschmidt	06/30/2009	12/30/2010	§102(a)(1)-(2)
Tuan	---	10/2009	§102(a)(1)
Brent	01/19/2010	05/27/2010	§102(a)(1)-(2)
Sandstrom-501	07/15/2011	03/29/2012	§102(a)(1)-(2)

II. SUMMARY OF THE '556 PATENT

A. Brief Description

The '556 Patent describes, among other things, “[a]n architecture for a load-balanced groups [sic] of multi-stage manycore processors shared dynamically among a set of software applications.” MS-1001, Abstract. Figure 2, reproduced below, depicts an embodiment of this architecture, which includes “hardware logic

implemented capabilities for scheduling tasks of application program instances and prioritizing inter-task communications (ITC) among tasks of a given app instance.” MS-1001, 10:10-13; *generally id.*, 10:7-11:16.

“In the architecture per FIG. 2, the multi-stage manycore processor system 1 is shared dynamically among tasks of multiple application programs (apps) and instances (insts) thereof.” *Id.*, 10:29-32. “[F]or each of the apps, each task [is] located at one of the (manycore processor) based processing stages 300,” although “for any given app-inst, copies of same task software (i.e. copies of same software code) can be located at more than one of the processing stages 300 of a given system 1.” *Id.*, 10:32-42, 6:23-33, 10:56-11:16. Thus, each worker stage 300 can host a single task for each of one or more software applications. A worker stage 300 can further manage one or multiple instances of each hosted task for each application. *Id.*

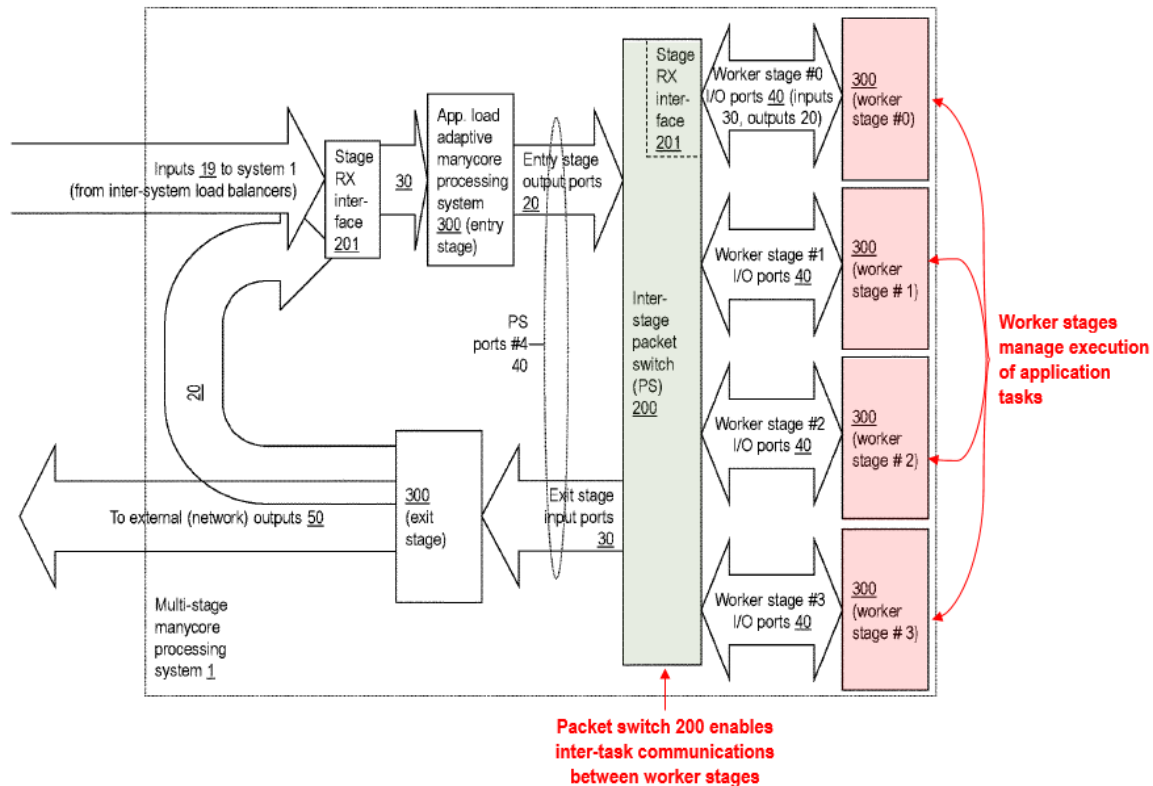


FIG. 2

MS-1001, FIG. 2 (annotated).

The '556 Patent further discloses that each processing stage 300 includes an array 515 of processing cores 520, and includes logic to periodically assign sets of app-task-instances to the cores for execution. MS-1001, 20:7-63; *generally id.*, 15:21-20:63, FIGS. 5-7.

Petitioner notes that the Challenged Claims include various limitations directed to field-programmable gate arrays (FPGAs). *See, e.g.*, [1.1]-[1.7]. Notwithstanding any incorporated disclosures, however, the only mention of FPGAs in the specification of the '556 Patent is at col. 20:21-29, which states that

“[a]ny of the cores 520 of a processor per FIG. 7 can comprise any types of software program and data processing hardware resources, e.g. central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs) or application specific processors (ASPs) etc., and in *programmable logic (FPGA) implementation*, the core type for any core slot 520 is furthermore reconfigurable per expressed demands of its assigned app-task.” MS-1001, 20:21-29¹; MS-1003, ¶¶35-38. Nothing in the four corners of the ’556 Patent’s specification explains how reconfiguration occurs on FPGAs or how structures might be dynamically varied on the FPGAs to implement assigned app-tasks. *Id.*

B. Prosecution History

ThroughPuter filed the application that issued as the ’556 Patent (U.S. Serial 17/195,174) with a request for accelerated examination on March 8, 2021—i.e., the same month that ThroughPuter initiated its litigation campaign against Microsoft with a complaint filed in the U.S. District Court for the Eastern District of Virginia asserting nine patents including an ancestor of the ’556 Patent (U.S. Patent 10,318,353). MS-1104.

On May 14, 2021, little more than two months after ThroughPuter filed the ’174 Application, and without a single rejection having been made or other sub-

¹ All emphasis in quoted material is added in this Petition unless otherwise noted.

stantive negotiation having occurred over claim scope, the Examiner issued a notice of allowance. MS-1002, 9-19. The notice stated that the claims “are considered allowed because no prior art or combination of prior art references disclose or suggest the combination of limitations specified in the independent claims.” *Id.*, 10-11.

There is no indication in the file history, however, that the Examiner was ever aware of references that form the basis of the grounds set forth in this Petition, including either Kupferschmidt or Tuan. Nor does the file history contain any indication that the Examiner substantively considered Brent in relation to the features recited in elements [1.4]-[1.5] or claim 3, or Sandstrom-501 in relation to the features recited in claim 4 of the ’556 Patent. *See infra*, Section VI.B. If these references had been considered, and for the reasons addressed in Grounds 1A-1D of this Petition, the ’174 Application should not have been allowed and the ’556 Patent should not have issued. MS-1003, ¶¶39-40.

III. LEVEL OF ORDINARY SKILL

For purposes of this IPR, Petitioner submits that a person of ordinary skill in the art at the time of the alleged invention (“POSITA”) would have had a Master’s degree in computer science, computer engineering, or a related field, and 2-3 years of practical computer programming or engineering experience, including experience designing or researching parallel processing systems. MS-1003, ¶¶18-20.

Additional graduate education could substitute for professional experience, or significant experience in the field could substitute for formal education. *Id.*

IV. CLAIM CONSTRUCTION UNDER 37 C.F.R. §§ 42.104(b)(3)

All claim terms should be construed according to the *Phillips* standard.

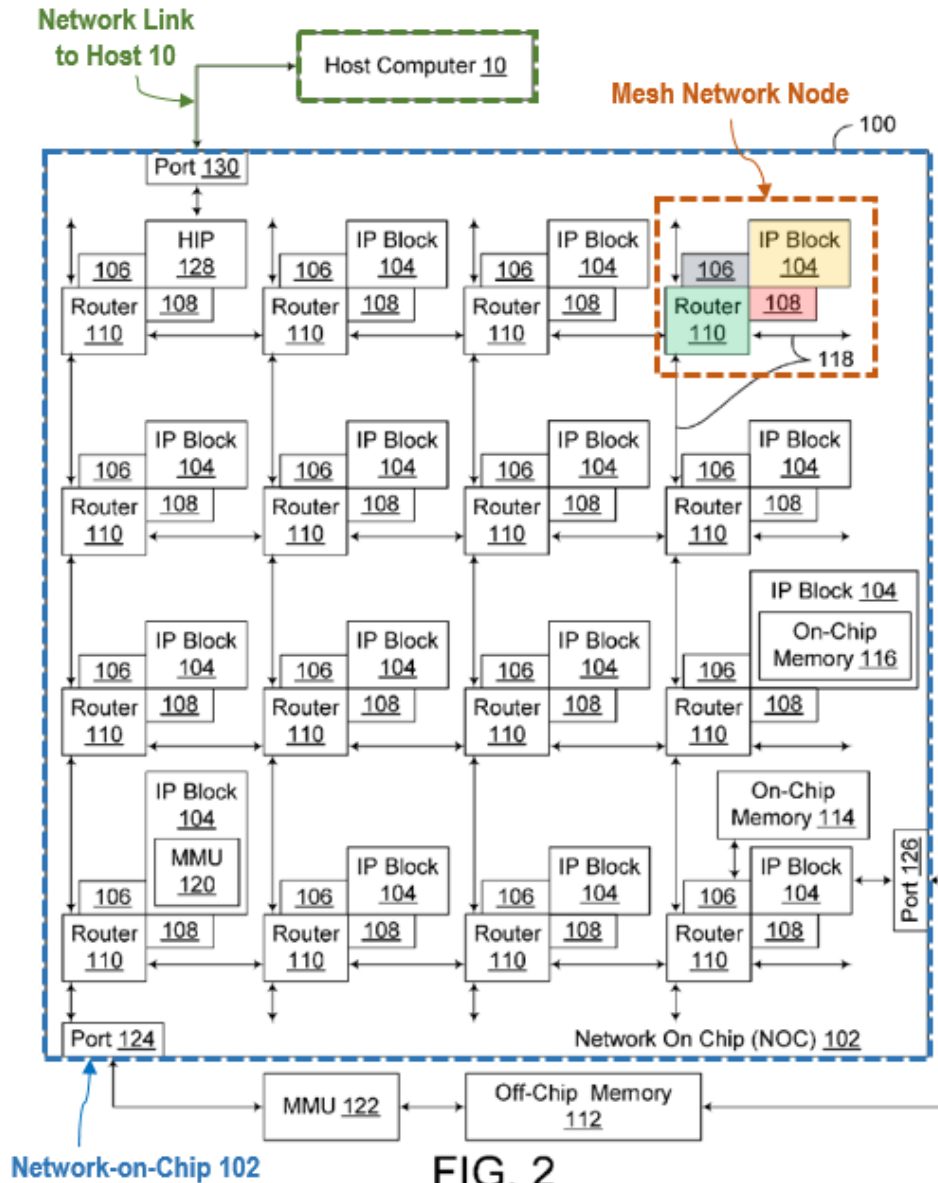
Phillips v. AWH Corp., 415 F.3d 1303 (Fed. Cir. 2005); 37 C.F.R. §42.100. Microsoft submits that formal constructions are presently unnecessary to resolve issues of controversy in the Petition. *Wellman, Inc. v. Eastman Chem. Co.*, 642 F.3d 1355, 1361 (Fed. Cir. 2011). To the extent the scope of any claim term is later brought into controversy during this proceeding, Petitioner will timely reply in accordance with due process.

V. THE CHALLENGED CLAIMS ARE UNPATENTABLE

A. [Ground 1A]: Kupferschmidt in view of Tuan (Claims 1, 2, and 4-8)

1. Overview of Kupferschmidt

Like the '556 Patent, Kupferschmidt discloses techniques for parallelizing execution of tasks on a multi-core processing system. MS-1003, ¶¶43-47. Figure 2, for example (reproduced below), illustrates a network-on-chip (NOC) device 102 that provides an array of interconnected processing cores referred to as “integrated processor” (“IP”) blocks 104. MS-1004, [0041]-[0047], FIG. 2.

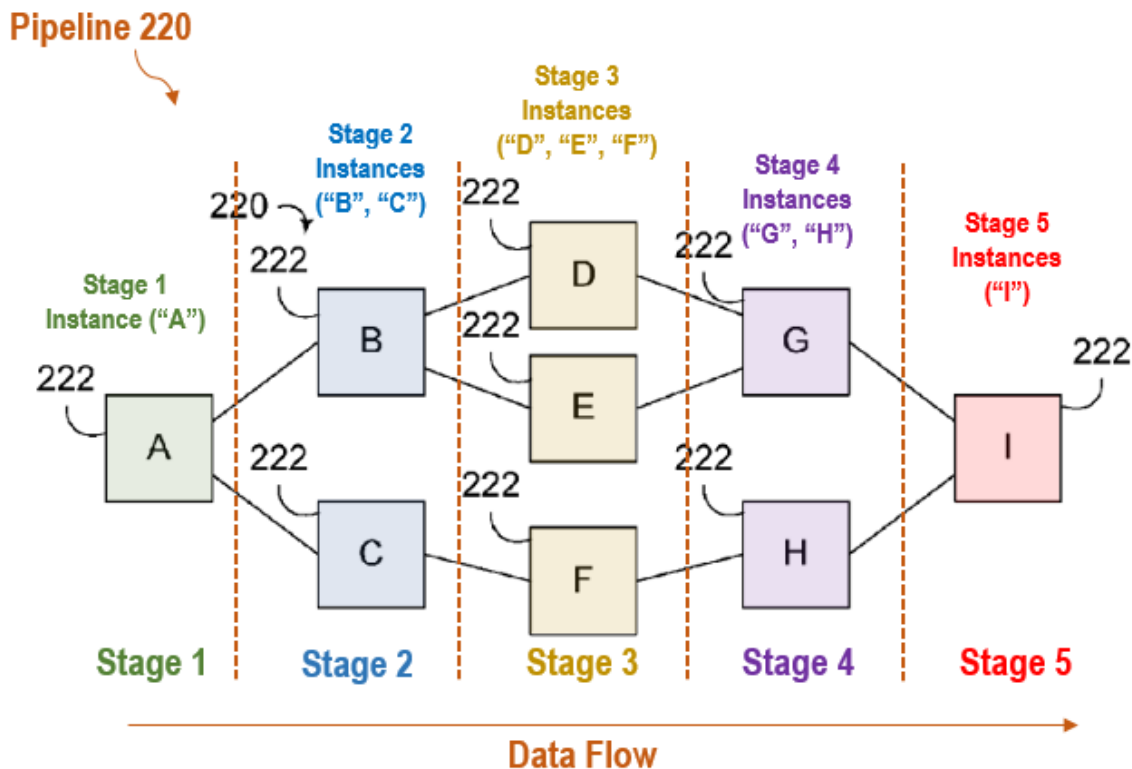


MS-1004, FIG. 2 (annotated).

In Kupferschmidt, “each IP block represents a reusable unit of synchronous or asynchronous logic design.” MS-1004, [0045]. In some embodiments, IP block 104 is implemented as a “generally programmable microcomputer.” *Id.*, [0053];

see also id., FIGS. 3-4. However, IP blocks 104 are not limited to general-purpose processors, and can alternatively be formed as “FPGA logic designs.” *Id.*, [0045].

Kupferschmidt further describes how IP blocks 104 can be configured to run software “pipelines” by assigning each “stage-instance” of one or more pipelines to execute on a respective IP block 104. MS-1004, [0075]-[0085], FIGS. 5-6. Figure 6, for example, depicts a pipeline that processes data sequentially through five stages. Each instance of each stage of the pipeline implements a data processing task that consumes an input and provides an output for processing by an instance of a next stage until a final result is produced. *Id.*



MS-1004, FIG. 6 (annotated).

2. Overview of Tuan

Tuan is a doctoral dissertation titled “A Study on a Multitasking Environment for Dynamically Reconfigurable Processors,” which published in 2009—i.e., some four years before the earliest possible priority date of the ’556 Patent. *See* MS-1005; *see also* MS-1007, ¶¶9-11; *see also id.*, ¶¶1-8, pp. 22-30; MS-1003, ¶¶48-51. According to Ms. June Munford, an experienced expert in library and information sciences, Tuan “was made available to the public shortly after its initial acquisition [by the Keio University library] in Fall 2009.” MS-1007, ¶11. Ms. Munford’s determination is supported by library records that demonstrate Keio University library’s acquisition of Tuan in October 2009, and is consistent with the year (2009) printed on Tuan’s title page. MS-1005, 1.

Among other things, Tuan explores the use of reconfigurable processors such in multi-core systems. MS-1005, 14, 32-33, 103-104. In a multi-core architecture, tasks from one or more applications are assigned to various cores for parallel execution. *Id.*, 32-33.

Along these lines, Tuan describes several types of reconfigurable processors that may be suitable for use in a multi-core architecture—including FPGAs. *Id.*, 25-26. “[A] multi-FPGA system ... can be seen as a multicore system, where each core corresponds to an FPGA.” *Id.*, 25. Each FPGA can be “assigned to a task for

execution, and all parts can be reconfigured and executed independently.” *Id.* Figure 2.9 depicts an example multi-core system comprised of FPGAs or other reconfigurable processors, each core executing a different task in parallel with other cores:

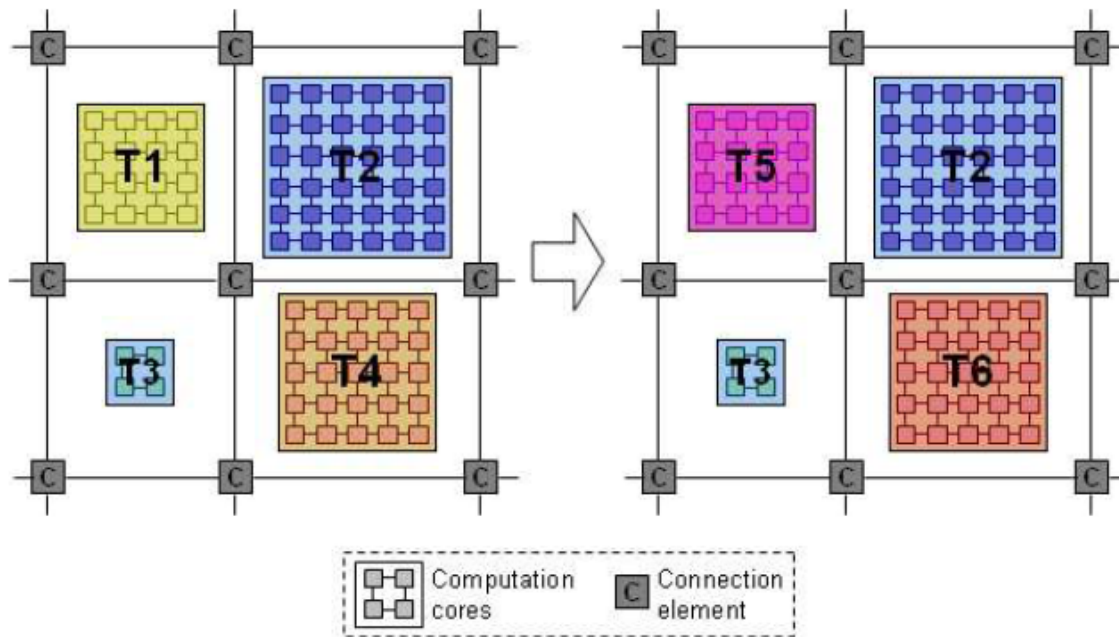


Fig. 2.9: Time-sharing, multicore partitioning model

MS-1005, 33 (FIG. 2.9); *see also id.*, 99-111 (“Chapter 6: Multicore Reconfigurable Architecture”).

3. Obviousness Based on Kupferschmidt-Tuan Combination

(a) Overview of the Kupferschmidt-Tuan Combination

As noted above (Section V.A.1), Kupferschmidt discloses that “each stage of a pipeline is typically implemented ... on a separate IP block.” MS-1004, [0081]; *see also id.*, [0075]; *supra*, Section V.A.1. Kupferschmidt also discloses that IP

blocks “can be formed as ... FPGA logic designs.” MS-1004, [0045]. However, lacks much additional detail about the “FPGA logic designs,” including whether each IP block would be implemented using a different FPGA or whether different IP blocks would be implemented using partitions of a shared FPGA. MS-1003, ¶59; *generally* ¶¶59-71. Kupferschmidt also does not specifically address how to configure an FPGA-based IP block to implement a task associated with a stage-instance in a pipeline that has been assigned to the IP block. *Id.*

For additional detail on FPGA logic designs in multitasking systems like Kupferschmidt’s, a POSITA would have found it obvious to look to Tuan. MS-1003, ¶60. As discussed above (Section V.A.2), Tuan describes multitasking, multicore systems similar to Kupferschmidt’s, but further clarifies that each “core” (e.g., IP block) can be formed from its own FPGA (e.g., rather than a portion of a partitioned FPGA). *E.g.*, MS-1005, 15, 25 (“each core corresponds to an FPGA”), 32-33; *supra*, Section V.A.2. Tuan also describes how assigning tasks to FPGA-based cores involves configuring or reconfiguring the FPGA to implement an assigned task in reconfigurable hardware, which would be understood to include varying structures of the FPGAs (e.g., fusing configurable logic blocks (CLBs)). MS-1005, 32-33, 36 (“dynamic reconfiguration offers a great advantage with respect to area- and power-efficiency because the PE array can be reconfigured so as to be optimized for various kinds of functions”), 37 (describing known configurations

schemes); MS-1003, ¶60. A scheduler or other resource manager can map tasks to available FPGAs so that individual FPGAs can be shared by different applications and different tasks over time. *Id.*

It would have been obvious to apply Tuan's teachings to Kupferschmidt so that, in a resulting Kupferschmidt-Tuan combination, IP blocks for pipeline stage-instances would be implemented as FPGA-based cores. MS-1003, ¶61. In the combination, the FPGA-based core of each IP block could be configured/reconfigured in hardware to implement a task for a respective stage-instance assigned to the particular FPGA. MS-1003, ¶61. Indeed, multiple reasons would have motivated a POSITA to combine the teachings of Kupferschmidt and Tuan to realize such a system before the Critical Date of the '556 Patent. *Id.*

First, Tuan's suggested use of FPGA-based cores in the combination provides a predictable option to further Kupferschmidt's express goal of forming IP blocks from "FPGA logic designs." MS-1004, [0045]; MS-1003, ¶62. As explained in Tuan, various architectures or models were known that could be used to implement a multitasking system using FPGA logic designs. These models included, for example, (1) non-time sharing, static partitioning models, (2) time-sharing, two-dimensional partitioning models, and (3) time-sharing multicore partitioning models. MS-1005, 29-33 (Section 2.4); MS-1003, ¶62. In non-time sharing, static partitioning and time-sharing, two-dimensional partitioning models, tasks are

assigned to different regions (partitions) of a single hardware element (e.g., a single FPGA). MS-1005, 30-32. Multicore models, by contrast, connect “independent computational cores ... by a [] network” and “a task can be mapped to any core for execution.” *Id.*, 32-33. For example, “a multi-FPGA system containing multiple FPGA chips can be seen as a multicore system, where each core corresponds to an FPGA.” *Id.*, 25.

Kupferschmidt’s limited description of “FPGA logic designs” does not specifically indicate whether its IP blocks would be partitioned from a single FPGA, or whether each IP block would instead employ its own FPGA as an individual core. MS-1003, ¶63. While either option may be suitable to meet the needs of a particular system, a POSITA would have found the option of implementing IP blocks from individual FPGAs to be especially useful and obvious in combination with Kupferschmidt. MS-1003, ¶63. In such a combination, the system would then dynamically invoke software pipeline(s) responsive to requests on behalf of program instances as taught in Kupferschmidt. MS-1004, [0075]-[0085], FIGS. 5-6. This process requires that stage-instances of the invoked pipeline(s) be assigned to IP blocks, and would further entail configuring or reconfiguring a *subset* of the FPGA-based cores (i.e., the subset to which stage-instances of the invoked pipeline(s) have been assigned)—a process which Tuan refers to as “partial reconfiguration.” MS-1005, 32-33. Tuan notes that “the process of *partial reconfiguration*

becomes much easier” in a multicore system “since cores are independent” of each other rather than partitions of a single FPGA. MS-1005, 32-33. Accordingly, forming the IP blocks in the combination as a multicore network of FPGA-based cores would have been obvious and beneficial to facilitate independent reconfigurations of individual IP blocks in a manner consistent with Kupferschmidt’s objective of forming IP blocks from “FPGA logic designs.” MS-1003, ¶63.

Second, a POSITA would have been motivated to apply Tuan’s suggested use of FPGA-based cores in the combination since such cores were readily available for use in “research and commerce” applications before the Critical Date of the ’556 Patent. MS-1005, 15 (“[R]econfigurable systems that are composed of multiple FPGA chips interconnected on a single processing board have been introduced for research and commerce.”); MS-1003, ¶64. The wide availability of FPGA-based cores would have been beneficial to reduce the cost, effort, and time implicated in constructing a system in accordance with the proposed combination. *Id.*

Third, a POSITA would have been motivated to apply Tuan’s suggested use of FPGA-based cores in the combination to increase the processing capacity available to the system to run one or more pipelines. MS-1003, ¶65. By implementing each IP block as an individual FPGA-based core, the entire capacity of the FPGA would be available for the respective task of each stage-instance rather than requiring resources to be shared among multiple tasks (as would occur in a shared-FPGA

partitioned model). MS-1005, 15, 25, 32-33; MS-1003, ¶65. Similarly, the multi-core multi-FPGA model would allow larger tasks to be hosted on a single core (IP block) than would otherwise be possible if a task was required to compete with other tasks for a share of logic blocks or other resources of the same FPGA. MS-1005, 31-33; MS-1003, ¶65.

Fourth, a POSITA would have been motivated to apply Tuan’s suggested use of FPGA-based cores in the combination to achieve well-known benefits of FPGA technology (and reconfigurable systems more generally). MS-1003, ¶66. These benefits are highlighted in Tuan, which notes that reconfigurable systems such as FPGAs “achieve potentially much higher performance than software, while maintaining a higher level of flexibility than hardware.” MS-1005, 14; *see also* MS-1004, [0077]-[0081], [0088]; MS-1003, ¶66. An IP-block implemented as an FPGA-based core would have allowed the IP block to be reconfigured for use in running different tasks for different stages and/or pipelines at different times, while still enjoying the comparative benefits of implementing the task in reconfigurable hardware rather than software. *Id.*

Fifth, a POSITA would have been motivated to apply Tuan’s suggested use of FPGA-based cores as IP blocks in the combination to better facilitate the identification and selection of available IP blocks to which to assign stage-instances in a pipeline. MS-1003, ¶67. Additional detail in this regard is provided below in the

analysis of element [1.4] (Ground 1A). *Infra*, Section V.A.3(b) [1.4]; MS-1003, ¶67.

Sixth, a POSITA would have found it obvious to apply Tuan’s suggested use of FPGA-based cores for multitasking systems in combination with Kupferschmidt since the combination involves no more than the use of known solutions to improve similar systems and methods in the same way. “[W]hen a patent ‘simply arranges old elements with each performing the same function it had been known to perform’ and yields no more than one would expect from such an arrangement, the combination is obvious.” *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 417 (2007). Indeed, the combination results in a straightforward reconfigurable multitasking system, which is the precise context in which Tuan originally envisioned use of such cores. A POSITA would have had a reasonable expectation of success implementing the combination, especially since Tuan and Kupferschmidt both pertain to multitasking systems in which different tasks are implemented on different processors (e.g., IP blocks), and Kupferschmidt already

acknowledged that IP blocks were compatible with FPGA logic designs.² MS-1004, [0045]; MS-1005, 2-3, 14-16, 25-26, 32-33; MS-1003, ¶¶68.

(b) Mapping of the Kupferschmidt-Tuan Combination to Claims 1, 2, and 4-8

In combination (*see supra*, Section V.A.3(a)), the teachings of Kupferschmidt and Tuan render obvious claims 1, 2, and 4-8 of the '556 Patent, and the combination provides each element of claims 1, 2, and 4-8 for at least the reasons addressed below. MS-1003, ¶¶72-132.

Element [1.P]

To the extent the preamble is a limitation, the teachings of Kupferschmidt in view of Tuan provide a method performed in a data processing system. MS-1003, ¶¶73-76. Kupferschmidt discloses methods “related to data processing, and in particular to processor architectures and inter-communication of hardware threads therein.” MS-1004, [0001]; *see also id.*, [0007] (“a **method** of processing a workload with a plurality of hardware threads”), [0009] (“a **method** is provided to process a workload in a multithreaded processor”). The methods for processing a

² FPGA-based cores also would have been advantageous relative to coarser-grained reconfigurable hardware such as DRPAs due to the increased flexibility and programmability of FPGAs. MS-1003, ¶¶69-70. The cores could be integrated on a single chip or distributed across multiple chips. *Id.*, ¶71.

workload can also include methods for “process[ing] at least a portion of a software pipeline.” *Id.*, [0033]; *see also id.*, [0036], [0075]-[0085], FIGS. 5-6; *supra*, Sections V.A.1, V.A.3(a).

Kupferschmidt also discloses, and the combination provides, a “data processing system” in which the methods are performed. As one example, a computer 10 like that depicted in FIG. 1 maps to the claimed “data processing system.” MS-1003, ¶74.

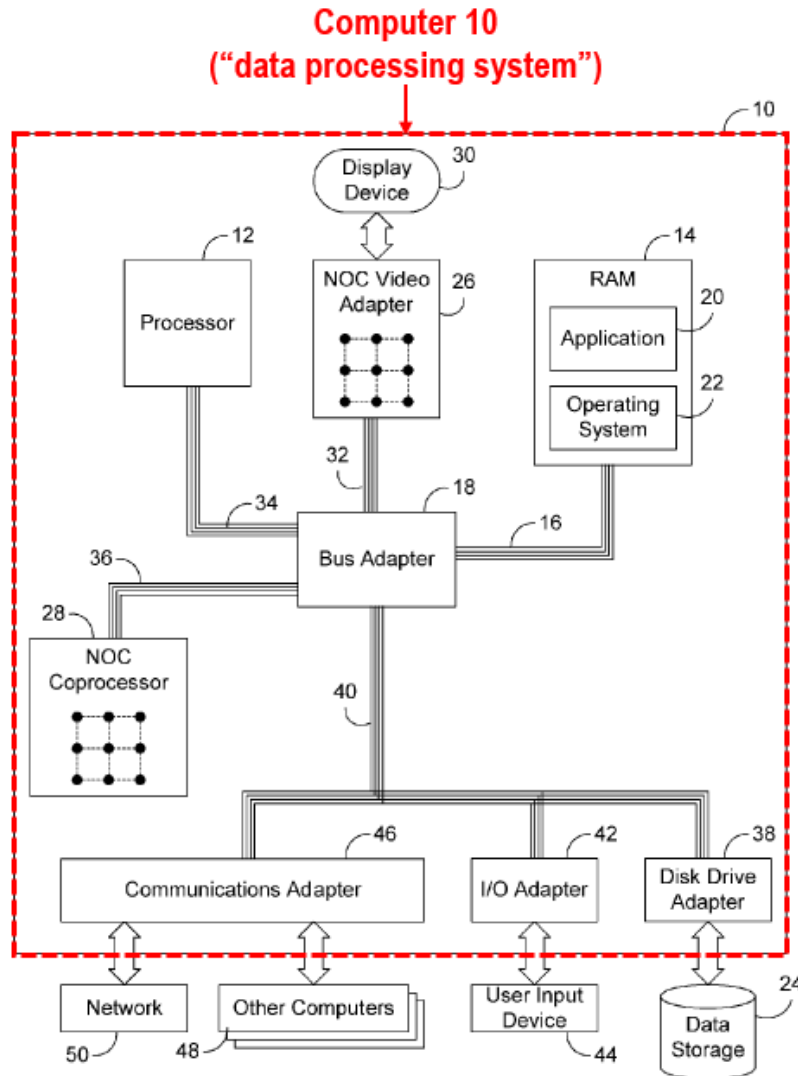


FIG. 1

MS-1004, FIG. 1 (annotated); *see also id.*, [0038] (“FIG. 1 illustrates exemplary automated computing machinery including an exemplary computer 10 useful in data processing consistent with embodiments of the invention.”), [0039]-[0043].

As another example, the computer 10 together with all or some of the external components that interface with computer 10—e.g., network 50, other comput-

ers 48, user input device 44, and/or data storage 24—map to the claimed “data processing system.” MS-1004, [0038]-[0043], FIG. 1; MS-1003, ¶75.

As a third example, a processing unit 200 like that depicted in FIG. 5 maps to the claimed “data processing system.” MS-1003, ¶76.

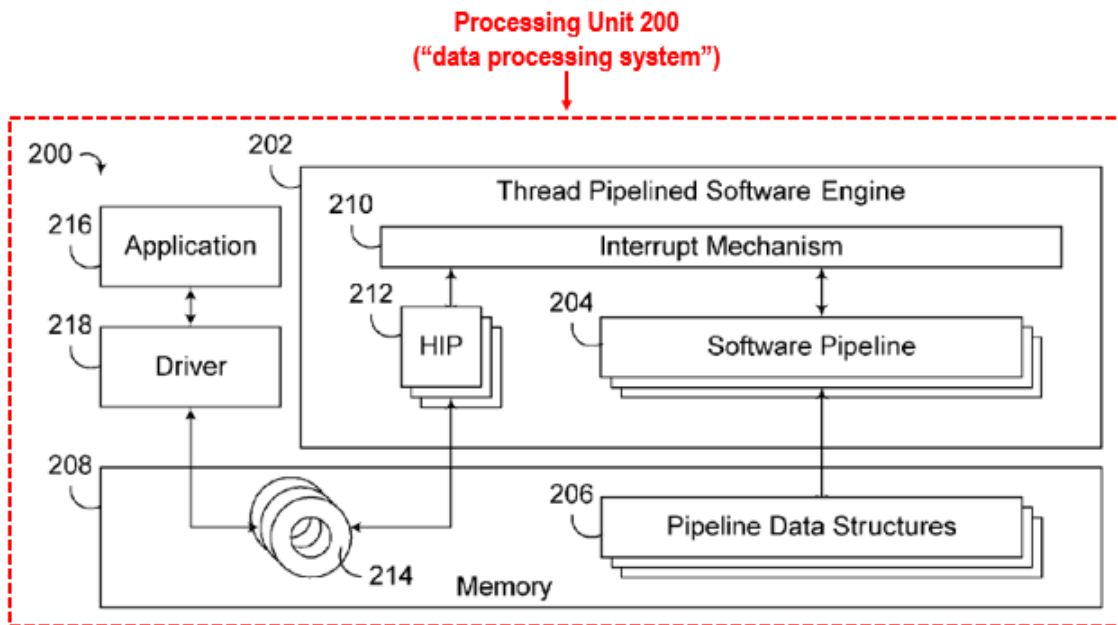


FIG. 5

MS-1004, FIG. 5 (annotated); *see also id.*, [0075] (“NOC 102 may be used in some embodiments to implement a software-based pipeline. In particular, FIG. 5 illustrates an exemplary processing unit 200 incorporating a thread pipelined software engine 202 that may be used to implement and execute one or more software pipelines 204 on top of a NOC architecture.”), [0076]-[0085].

Element [1.1]

Kupferschmidt discloses, and the combination provides, a data processing

system that manages a plurality of programs. MS-1003, ¶¶77-84. Figure 1 shows an application 20 and an operating system 22 on computer 10, as one example of a plurality of programs:

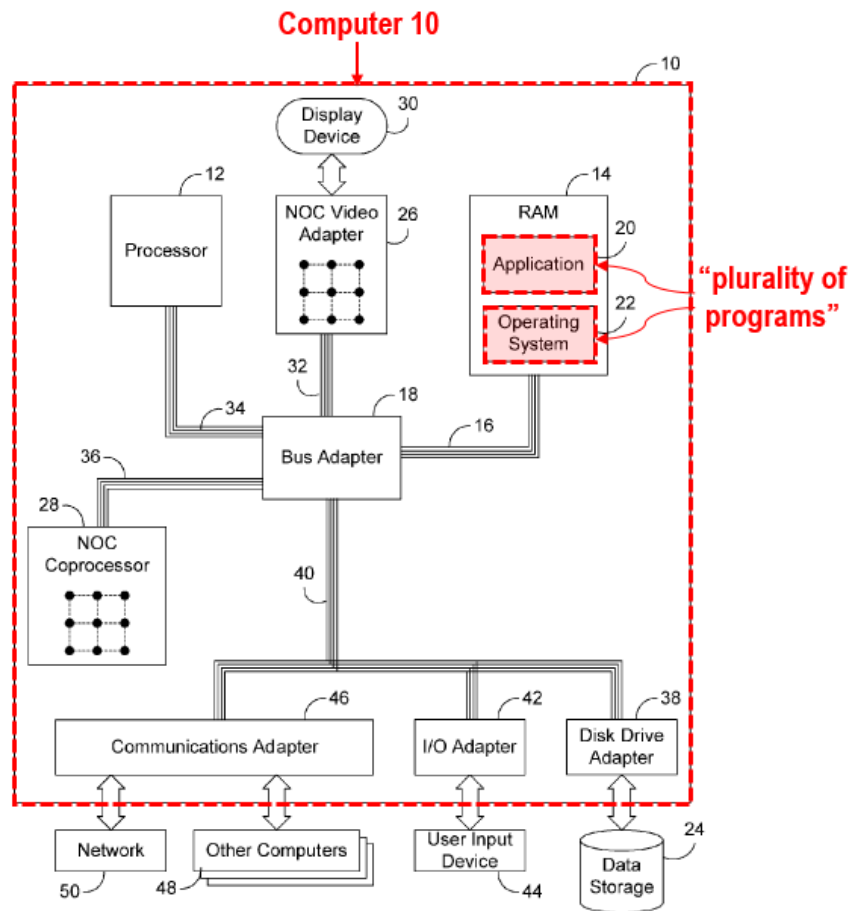


FIG. 1

MS-1004, FIG. 1 (annotated); *see also id.*, [0039] (program 20 “carr[ies] out particular data processing tasks such as, for example, word processing, spreadsheets, database operations, video gaming, stock market simulations, atomic quantum process simulations, or other user-level applications”; operating system 22 can be “UNIX™, Linux™, Microsoft Windows XP™, Microsoft Windows VISTA™,

AIX™, IBM's i5/OS™, and others as will occur to those of skill in the art”).

With respect to FIG. 5, Kupferschmidt similarly discloses a representative application 216:

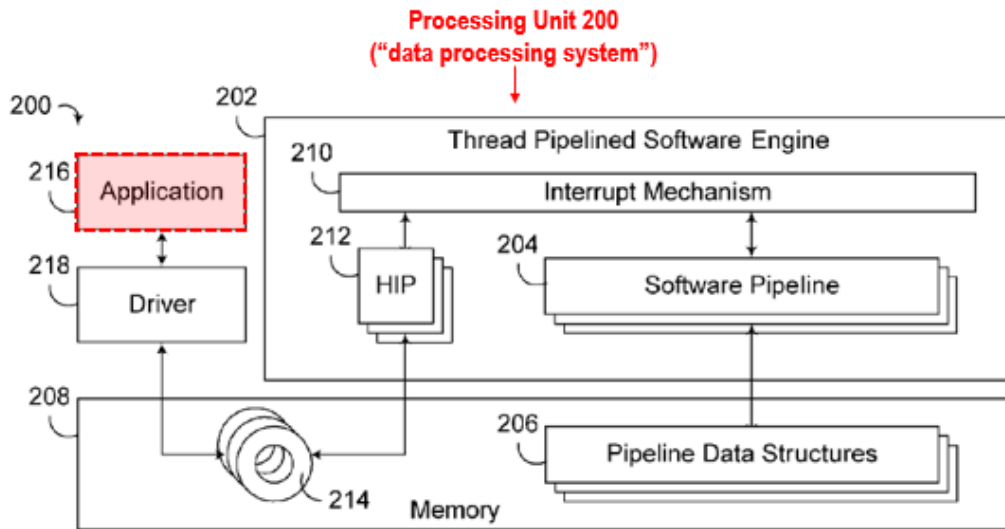


FIG. 5

MS-1004, FIG. 5 (annotated); *see also id.*, [0076].

A POSITA would have understood application 216 to be representative of any of a plurality of programs managed by the data processing system, and would have understood processing unit 200 and the system generally to manage multiple programs capable of executing in parallel. MS-1003, ¶79. To this point, Kupferschmidt explicitly states that “processing unit 200 incorporat[es] a thread pipelined software engine 202 that may be used to implement and execute one **or more** software pipelines 204 on top of a NOC architecture.” MS-1004, [0075], FIG. 5

(depicting multiple pipelines 204). In other words, the system is not limited to executing a single software pipeline. MS-1003, ¶79. Multiple pipelines can instead be invoked in parallel, even when the pipelines belong to or otherwise are associated with different programs. *Id.*; MS-1004, [0080] (“performing different functions in different ***applications***”), [0047] (“messages carrying data and instructions for processing the data among IP blocks in parallel ***applications*** and in pipelined ***applications***”), [0060] (“pipelined ***applications***”). Whether taken as a component of computer 10 (e.g., co-processors 26 or 28) or as an external component communicably coupled to computer 10 as a remote host, a POSITA would have understood Kupferschmidt as teaching that the data processing system including processing unit 200 would be capable of running software pipelines for any requesting program on the computer 10 including operating system 22 and any of the multiple applications 20 that are ordinarily supported by operating systems like Microsoft Windows or others disclosed in Kupferschmidt. MS-1004, [0039], [0041], [0044], [0075], FIGS. 1-2, 5; MS-1003, ¶79.

To the extent Kupferschmidt does not expressly disclose a “plurality” of programs managed by the data processing system, such features still would have been obvious in the combination by the relatively late Critical Date of the ’556 Patent (August 23, 2013). MS-1003, ¶80. Tuan, for instance, discloses that FPGAs and other reconfigurable hardware have made it “feasible for ***several applications***” to

concurrently share processing resources in a multitasking system. MS-1005, 2; *see also id.*, 2 (“sharing a piece of reconfigurable hardware among ***multiple concurrently executing applications***”), 16 (“Multitasking aims at providing an appropriate mechanism to allow ***multiple applications*** to efficiently share a piece of reconfigurable hardware.”), 65 (“multiple target applications could be mapped onto a candidate architecture”), 81 (“concurrent applications in a multi-FPGA system”). Considering these teachings from Tuan, it would have been obvious for the data processing system in the combination to manage multiple programs (such that multiple programs could invoke software pipelines consistent with the teachings of Kupferschmidt). MS-1003, ¶80. A POSITA would have sought to implement these capabilities in the resulting combination to more flexibly support a range of programs that were typically installed on computer systems in the timeframe of the 2013 Critical Date of the ’556 Patent. MS-1003, ¶80. Additionally, a POSITA would have implemented the combination to manage multiple programs so that pipelines could be implemented on behalf of instances of multiple programs ***in parallel***—e.g., rather than restricting programs to invoking pipelines only serially or restricting pipelines to be invoked for only a single program on the system. MS-1003, ¶80.

In the combination, Kupferschmidt further discloses hardware logic and/or software logic—e.g., host interface processors (HIPs) 212 in a first example or the

thread pipelined software engine 202 more generally in a second example—that are configured to receive requests to perform different tasks on behalf of instances of the plurality of programs managed by the data processing system. MS-1003, ¶81. “In order to initiate work in a pipeline, a software application 216 **issues requests** through an appropriate driver 218 in the form of API calls, which then generates appropriate requests for the HIP and stores the requests in a push buffer 214.” MS-1004, [0076], FIGS. 2-5. “The HIP 212 for the relevant pipeline **pulls work requests** off of push buffer 214 and initiates processing of the request by the associated pipeline.” *Id.*, [0052] (HIP receives and dispatches “data processing **requests** from the host computer”); MS-1003, ¶81.

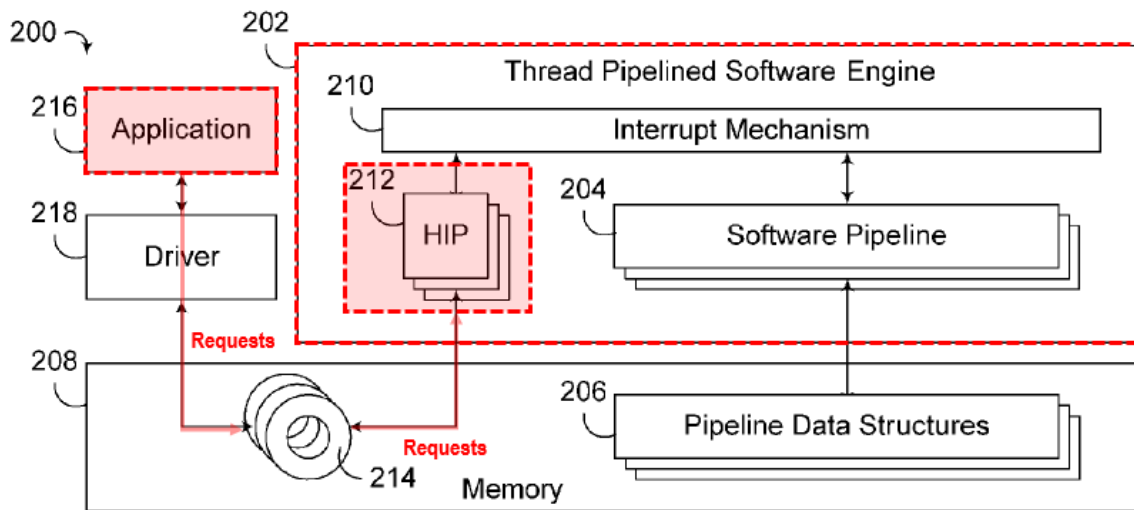


FIG. 5

MS-1004, FIG. 5 (annotated); *see also id.*, FIG. 2 (depicting a HIP 128 implemented on an IP block 104 in NOC 102).

Through these teachings, Kupferschmidt discloses and renders obvious requests to perform different tasks on behalf of program instances managed by the data processing system since a pipeline by its nature comprises a set of “tasks” that process data for a requesting program instance. MS-1004, (“a software pipeline 204 implements a function that is segmented into a set of modules or ‘stages’ of computer program instructions that cooperate with one another to *carry out a series of data processing tasks in sequence*”), [0079]-[0082], FIG. 6. Thus, for example, requests from instances of different programs to invoke work in different pipelines would constitute requests to perform the different tasks defined by each pipeline. MS-1003, ¶82.

Moreover, each of the multiple requesting applications/programs in the combination provides at least one respective “instance” of the requesting program.³

³ Kupferschmidt separately uses the term “instance” when referring to instances of stages in a pipeline (i.e., so-called “stage-instances”). MS-1004, [0075]-[0081], FIGS. 5-6. This Petition, however, maps instances of the requesting programs (e.g., instances of programs such as application 216), rather than “stage-instances” of pipelines, to the recited “instances of a plurality of programs” in claim 1. In the alternative mapping described at the end of the analysis of Element [1.1], instances

Similarly, the HIPs 212 and engine 202 would receive requests on behalf of “instances of a plurality of programs” as claimed (e.g., each HIP could receive requests on behalf of a single running instance of each requesting program). MS-1003, ¶83. The specification of the ’556 Patent broadly describes an “instance” as “a process, a job, a task, a thread, a method, a function, a procedure or an instance any of the foregoing, *or an independent copy of the given program.*” MS-1001, 2:55-58; *see also* MS-1011, 3 (defining “instance” as “a single copy of a software product in operation ...”); MS-1012, 3 (defining “instance” as “[a] single copy of a running program”). As each requesting program in the combination (e.g., application 216) would be running a respective copy of the program in order to issue a request, it follows that the requests originate from “instances” of the programs. MS-1003, ¶83.

To the extent the language of [1.1] further encompasses requests *received* on behalf of the instances of a plurality of programs, the Kupferschmidt-Tuan combination provides another mapping to the features recited in this claim element. MS-1003, ¶84. In this mapping, instances of whole pipelines 204 or 220 correspond to the claimed “instances of a plurality of programs.” MS-1003, ¶84. Here, each

of whole pipelines are mapped to the recited “instances of a plurality of programs” rather than individual stage-instances.

pipeline 204 or 220 corresponds to a recited “program” (e.g., a program defined by the respective modules/stages of that pipeline). MS-1004, [0077] (“a software pipeline 204 implements a function that is segmented into a set of modules or ‘stages’ of computer program instructions”), [0081]; MS-1003, ¶84. The HIPs (e.g., HIPs 212) then constitute hardware logic and/or software logic that receive requests from client programs (e.g., application 216) to perform the different respective tasks defined by each pipeline. MS-1004, [0076]; *generally id.*, [0075]-[0085]; MS-1003, ¶84. Requests are “receiv[ed] ... *on behalf*” of instances of the pipelines since the pipelines themselves do not directly receive requests from the client programs. The HIPs instead receive requests on behalf of the pipelines when they “pull[] work requests off of push buffer 214” to “initiate[] processing of the request by the associated pipeline.” MS-1004, [0076], FIG. 5.⁴

⁴ The language of [1.1] does not expressly limit the entity or entities that *issued* the requests. This is confirmed by subsequent language in claim 1 that alternately refers to “requesting instances” and a “requesting client,” although neither the “requesting instances” nor the “requesting client” are specifically tied to or identified as the issuer(s) of the particular requests recited in [1.1]. Claim 1 thus leaves open the possibility that the requests may be issued by the instances of the plurality of programs, a requesting client (whether the client is or is not among the instances of

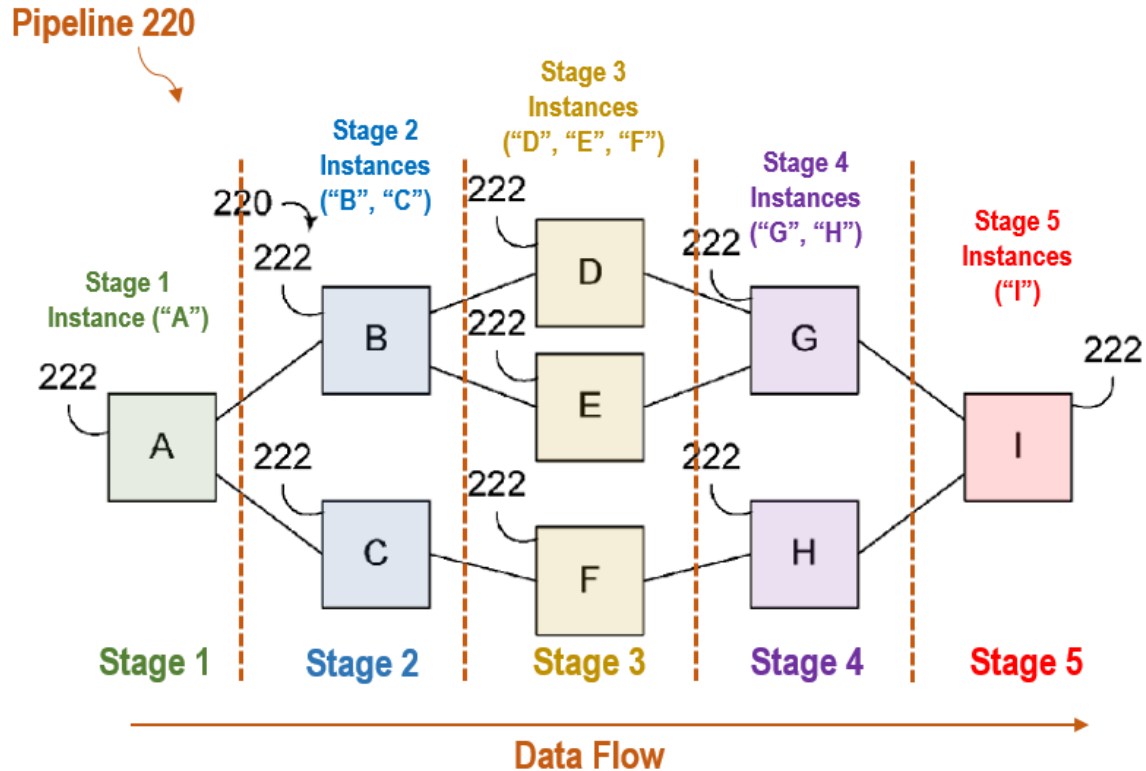
Element [1.2]

Kupferschmidt discloses “a thread pipelined software engine 202 that may be used to implement and execute one or more software pipelines 204” on a networked array of IP blocks. MS-1004, [0075]-[0085], FIGS. 2, 5-6. “[E]ach instance of each stage of a pipeline is typically implemented ... on a separate IP block,” and a respective HIP for each pipeline “typically sets up the pipeline by configuring each stage with a desired number of instances, ***with the network location of each instance of each stage provided to other instances of other stages to enable each instance to send its resultant workload to the proper instance in the next stage.***” *Id.*, [0081]; *see also id.*, [0088] (“HIP 408 sets up the software pipeline, [and] assigns threads of execution to stage-instances in the pipeline[] ...”); *supra*, Sections V.A.1, V.A.3(a). For example, “[e]ach stage is assigned a stage ID, and each instance of a stage is assigned an identifier.” MS-1004, [0081]. Stages are then “configured with a stage ID for each instance of a next stage,” including “the number of instances in the next stage as well as the network location of each

the plurality of programs), or other entities. It follows that the terms “requesting instances” and “requesting client” similarly encompass instances and clients, respectively, that issue requests, regardless of whether those entities issue the particular requests in [1.1].

instance” of the next stage such that the output of a given stage-instance can be communicated to an appropriate network location for an IP block of an instance in a next stage. *Id.*, [0082], [0075] (“an interrupt mechanism 210 is provided to enable stages of a pipeline to notify one another of pending work to be performed”). Through these operations, the respective HIP (***hardware logic and/or software logic***) for each of the pipelines—and hence for each of the program instances (*see supra*, [1.1])—identifies communication interdependencies between different processing stages of a set of processing stages of the respective pipeline or program instance as recited in [1.2]. *Supra*, [1.1]; MS-1003, ¶¶92-93; *see also id.*, ¶¶85-93.

In more detail, Kupferschmidt describes techniques for implementing and running “pipelines” on a set of IP blocks (i.e., FPGA-based cores in the combination with Tuan). *Supra*, Section V.A.1, V.A.3(a); MS-1004, [0075]-[0085], FIG. 6.



MS-1004, FIG. 6 (annotated).

As shown in FIG. 6, pipelines include multiple stages, and each stage includes one or more instances (i.e., stage-instances) for processing data received at that stage. MS-1004, [0078]-[0082], FIG. 6. The processing stages of a pipeline provide a set of processing stages of a respective instance of a program (e.g., application 216) that requested work to be performed by the pipeline. *Id.* For example, when an instance of a first application initiates work in a first pipeline and an instance of a second application initiates work in a second pipeline, the stages of the first pipeline perform processing for the instance of the first application and the

stages of the second pipeline perform processing for the instance of the second application. MS-1004, [0076]-[0077]; MS-1003, ¶¶85-86.

To this point, Kupferschmidt describes how different client program instances can issue requests directed to different pipelines. MS-1004, [0076] (“HIP 212 *for the relevant pipeline* pulls work requests off of push buffer 214 and *initiates processing of the request by the associated pipeline.*”). In the case of a 1:1 correspondence between program instances and pipelines, each pipeline effectively implements a set of processing stages *of* the respective program instance that issues work to the pipeline—albeit on specialized processors (e.g., FPGA IP blocks) that can execute tasks of the set of processing stages much more efficiently than the processor(s) on which other aspects of the program instances may run (e.g., general-purpose processor 12)). MS-1004, [0040] (“The NOC coprocessor of FIG. 1 may be optimized, for example, *to accelerate particular data processing tasks* at the behest of the main processor 12.”), [0041] (“coprocessor are each optimized for programs that use parallel processing and also require fast random access to shared memory”); MS-1003, ¶¶86-88.

Indeed, Kupferschmidt’s description of “accelerating particular data processing tasks at the behest of the main processor 12,” i.e., the main processor 12 that is primarily charged with executing instances of programs such as user applications 20/216 or OS 22, specifically suggests that the accelerated tasks would be

part *of* the program instances that execute on processor 12 themselves. MS-1003, ¶89. Hardware acceleration was well known before the '556 Patent, and ordinarily involves diverting certain tasks or functions of a program to execute on one or more accelerated processors. MS-1003, ¶89. In this context, a POSITA would have understood that accelerated tasks implemented as pipelines on IP blocks would be tasks *of* the requesting program instances themselves. *Id.* At minimum, using the pipelines for hardware acceleration of tasks in a set of processing stages of the respective program instances would have been obvious in view of Kupferschmidt's suggestion in paragraph [0040], and a POSITA would have sought to implement the system with hardware acceleration of tasks from the requesting program instances to more efficiently execute parts of the program instances than they otherwise could be executed as software on the main processor (e.g., a general-purpose processor) and to free up resources or capacity of the main processor for other items.⁵ MS-1003, ¶89. Implementing the pipelines as accelerated parts of the respective program instances likewise would have been obvious in view of Tuan,

⁵ Moreover, in the alternative mapping described above in [1.1] in which each instance of pipeline 204/220 is taken as an instance of a program (and the set of pipelines 204/220 collectively constitute "instances of a plurality of programs"), the respective processing stages of each instantiated pipeline directly provides "a set of

which expressly describes such techniques, and a POSITA would have been motivated to apply Tuan's teachings in this regard in the combination for similar reasons (e.g., to more efficiently execute parts of the program instances than they otherwise could be executed as software on the main processor and to free resources of the main processor). MS-1005, 23 ("hardware tasks are the *parts of an application* implemented in reconfigurable logic"), 29 ("reconfigurable platforms ... are often used for accelerating *certain parts of an application*"), 99 ("dynamically reconfigurable accelerators ... have been popularly used"), FIGS. 2.2, 4.3, 4.9 (depicting examples of hardware accelerated tasks of programs).

As noted above, the respective HIP 212 in engine 202 for each pipeline (and thus for each program instance associated with each pipeline, i.e., "hardware logic and/or software logic for each of the instances") identifies communication interdependencies between different processing stages in the set of processing stages of each pipeline. MS-1004, [0075], [0081]-[0082] (describing techniques for configuring stage-instances with the network locations of IP blocks where produced workloads can be communicated to instance(s) of a next stage), FIG. 6; MS-1003, ¶¶90-93.

processing stages of the respective instance" as recited in [1.2]. MS-1004, [0079]-[0082], FIG. 6; MS-1003, ¶91.

Element [1.3]

The Kupferschmidt-Tuan combination provides [1.3] through the process of setting up and dynamically adjusting pipelines on interconnected FPGA IP blocks. *Supra*, Section V.A.3(a); *see also supra*, Sections V.A.1-2; MS-1003, ¶¶94-99.

In this regard, Kupferschmidt discloses that a respective host-interface processor (HIP) 212 in engine 102 (“***hardware logic and/or software logic***”) for each pipeline “sets up the pipeline by configuring each stage with a desired number of instances,” and assigning each stage-instance to an IP block. MS-1004, [0081]-[0082], [0088] (“HIP 408[] ... assigns one or more IP blocks to handle each stage of the pipeline[]”), [0075] (“implement and execute one or more software pipelines 204 on top of a NOC architecture”).

To illustrate, consider an example of (i) a first pipeline for an instance of a first application having nine stage-instances “A”-“I” like those illustrated in FIG. 6 and (ii) a second pipeline for an instance of a second application having four stage-instances “J”-“M”. Consistent with Kupferschmidt’s teachings, respective HIPs for the first and second pipelines could initially setup the pipelines by assigning their respective stage-instances to different IP blocks, for example, as follows:

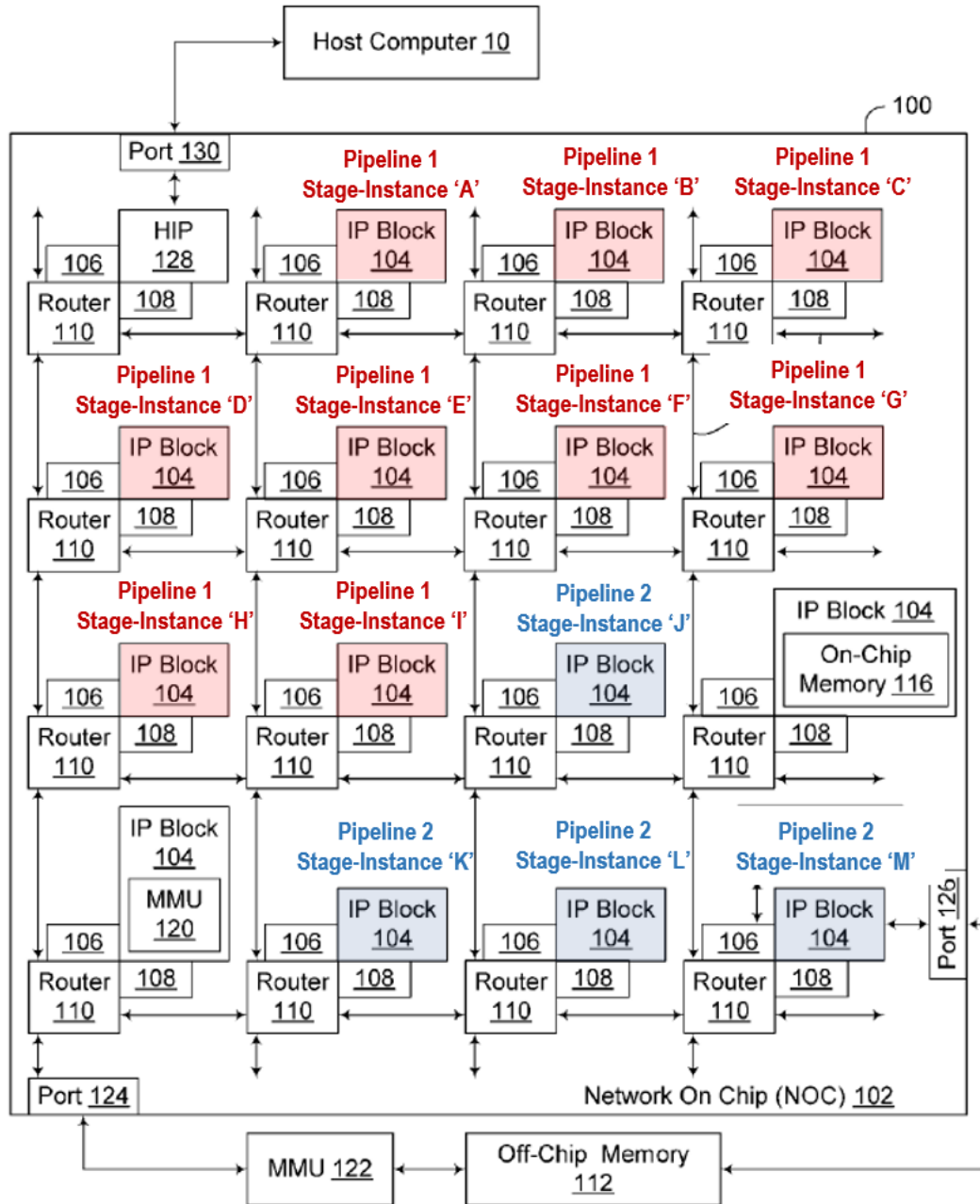


FIG. 2

MS-1004, FIG. 2 (annotated), [0044]-[0052]; MS-1003, ¶¶94-95.

Kupferschmidt further discloses that “IP blocks are logic cores that can be formed as ... FPGA logic designs,” but provides little additional detail about these

designs. MS-1004, [0045]. For these details, a POSITA would have looked to Tuan. *Supra*, Sections V.A.2, V.A.3(a). Among other things, Tuan describes a networked array of FPGA-based cores, each core implemented as or including an FPGA that can be dynamically reconfigured to implement an assigned task. MS-1005, 25 (“a multi-FPGA system containing multiple FPGA chips can be seen as a multicore system, where each core corresponds to an FPGA”), 26, 32-33; *see also id.*, 99-111 (“Chapter 6: Multicore Reconfigurable Architecture”); *supra*, Sections V.A.2. Tuan’s Figure 2.9, for instance, depicts a set of four cores (e.g., FPGAs) configured at a first time to implement tasks T1-T4, respectively. MS-1005, 33. At a later time, new tasks T5 and T6 are assigned to the top-left and bottom-right cores, which in turn are reconfigured to implement tasks T5 and T6 in place of T1 and T4, respectively. *Id.*

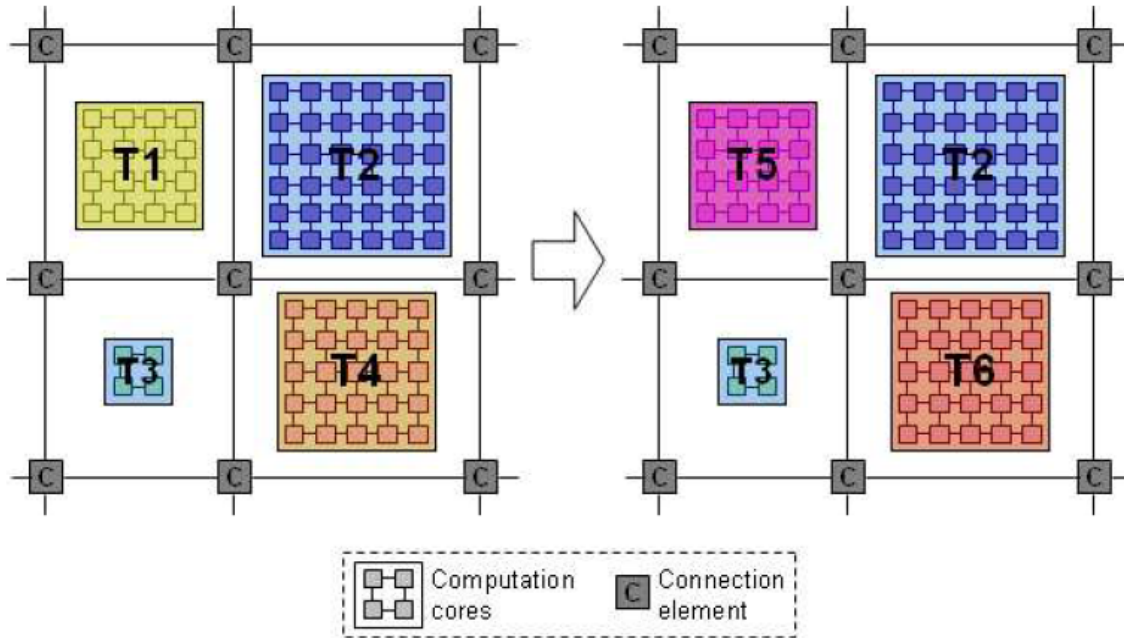


Fig. 2.9: Time-sharing, multicore partitioning model

As described in detail above (*supra*, Section V.A.3(a)), it would have been obvious to combine Kupferschmidt and Tuan such that IP blocks in the combination would be implemented with FPGA-based cores, stage-instances of pipelines would be assigned to respective FPGA-based cores, and the operations for assigning a stage-instance to an FPGA-based IP block would predictably entail reconfiguring structures of the FPGA to implement a task defined by the stage-instance, as taught by Tuan. MS-1005, 32-33 (describing “process of partial reconfiguration” in a multicore reconfigurable system that allows cores (e.g., FPGAs) to be reconfigured for new tasks substantially independent of other cores); *supra*, Section V.A.3(a); MS-1003, ¶¶96-97.

In the combination, and consistent with Kupferschmidt's teachings, HIPs in pipeline engine 202 would dynamically vary structures of the FPGA-based cores (IP blocks) used to process different tasks of the instances of the plurality of programs both when initially setting up pipelines for the program instances and when dynamically scaling or adjusting the number of stage-instances in different stages of the pipeline(s) to optimize performance.⁶ MS-1007, [0077], [0081]-[0082]; MS-1003, ¶98. For example, referring to the first and second pipelines depicted in annotated FIG. 2 above, structures of nine FPGA-based IP blocks in the combination would be dynamically varied to initially setup the first pipeline by reconfiguring these FPGAs to implement the corresponding tasks of the respective stage-instances from the first pipeline. MS-1004, [0077]-[0082], [0088]; MS-1003, ¶98. Similarly, structures of four additional FPGA-based IP blocks would be dynamically varied to initially setup the second pipeline. *Id.* As taught in Kupferschmidt,

⁶ Kupferschmidt discloses that each instance of a processing stage in a pipeline implements a data processing task. MS-1004, [0079]-[0080]. The pipelines thus provide different tasks of the instances of the plurality of programs for similar reasons that the pipelines provide sets of processing stages of respective program instances as described above in [1.2].

conditions in the data processing system that dictate the initial setup of the pipelines would include the amount of work (e.g., the “workload”) requested to be performed in the pipelines and the relative throughputs of different stages in the pipelines. MS-1004, [0076]-[0077], [0081]-[0082]. In particular, these conditions are considered by the HIP in determining how to scale and balance the number of instances that are to be provided among stages in the pipelines. MS-1004, [0077] (“a pipeline may be scaled by instantiating additional instances of a stage as needed depending on workload”), [0080]-[0082]; MS-1003, ¶98.

Kupferschmidt also recognized that workload and other conditions of the data processing system can change over time, and provided an ability for the HIPs to respond to these changes by dynamically scaling pipelines either as a whole or in individual stages to improve performance, including increasing throughput and preventing bottlenecks from developing between stages. MS-1004, [0077], [0081]-[0082], [0088] (“dynamically reallocate threads of execution to different stages of the pipeline to maximize throughput and minimize bottlenecks”). For example, “workload monitoring may be performed during runtime, and [] instances may be dynamically added or removed from a stage as needed for balancing the load among the stages of the pipeline.” MS-1004, [0081]; *see also id.*, [0082], [0094] (“optimize throughput”). In the combination, dynamically adding or removing stage-instances from a pipeline based on monitored conditions as taught in

Kupferschmidt would thus entail dynamically varying structures of the FPGA IP blocks to which additional stage-instances are assigned or stage-instances are removed (e.g., by reconfiguring logic structures in the FPGA IP blocks to implement the task of an assigned stage-instance). MS-1003, ¶99.

Element [1.4]

Identifying available FPGAs as recited in [1.4] would have been an obvious facet of the Kupferschmidt-Tuan combination as described above. *Supra*, Section V.A.3(a), [1.1]-[1.3]; MS-1003, ¶¶100-105. Recall that, as taught in Kupferschmidt, a respective host-interface processor (HIP) 212 in engine 102 (“**hardware logic and/or software logic**”) for each pipeline “sets up the pipeline by configuring each stage with a desired number of instances,” and assigns each stage-instance to execute on a different IP block. MS-1004, [0081]-[0082], [0088] (“HIP 408[] ... assigns one or more IP blocks to handle each stage of the pipeline[]”), [0075] (“implement and execute one or more software pipelines 204 on top of a NOC architecture”). In identifying IP blocks (i.e., FPGAs in the combination) to which stage-instances of a pipeline can be assigned, a POSITA would have understood and it would have been obvious that the respective HIP for the pipeline would identify “available” FPGAs that are available to process the different pro-

cessing stages of the pipeline (and thus the different processing stages of “requesting instances of respective programs” as recited in [1.4] (*see supra*, [1.1]⁷)). MS-1003, ¶101. The identification of available FPGAs would predictably occur whenever stage-instances are to be assigned to IP blocks, both when initially setting up a pipeline and when later dynamically reconfiguring the pipeline (e.g., adding instances to one or more stages for load balancing purposes). MS-1004, [0081]-[0082]; MS-1003, ¶101.

As one example, when HIPs first setup one or more pipelines on the FPGA-based IP blocks, and before any other pipelines have been setup (or after any previously instantiated pipelines have been deactivated), none of the FPGA-based IP blocks would yet be busy or otherwise purposed for another task at the time the pipeline(s) are setup. MS-1003, ¶102. Each FPGA-based IP block in the system

⁷ In the alternative mapping discussed above in [1.1] where instances of whole pipelines 204/220 are taken as the “instances of a plurality of programs,” each instance of a pipeline is a “requesting instance” that issues requests as described in Kupferschmidt. *See* MS-1004, [0061] and [0064] (describing “request messages” sent by IP blocks in a pipeline), [0090] (describes nodes in a pipeline capable of issuing interrupts to other nodes through “requests” to an “interrupt mechanism 441”); MS-1003, Footnote 3.

would thus be “available” to process a new task or stage-instance, and the HIPs’ operations for assigning stage-instances from the new pipeline(s) to IP blocks would predictably involve identifying “available” FPGAs—especially since each FPGA-based IP block that could process a stage-instance of a pipeline is currently inactive/unused. *Id.* Likewise, it would have been obvious to identify inactive/unused FPGA-based IP blocks as “available” IP blocks when setting up new pipelines or scaling up existing pipelines even when a subset of FPGA-based IP blocks in the system have already been assigned to other stage-instances that have not yet completed execution. MS-1003, ¶102. A POSITA would have understood this approach to be obvious and beneficial in order to mitigate overhead associated with preempting tasks of stage-instances that were previously assigned to IP blocks, particularly when other IP blocks are already free and the assignment of new stage-instances to IP blocks would not require preemption. *Id.* Avoiding preemption in these circumstances likewise would have been obvious and beneficial to avoid disruption in the execution of pipelines that currently rely on the stage-instances previously assigned to those IP blocks. *Id.*

Moreover, to the extent the term “available” encompasses not just currently unused/inactive FPGAs but also currently used/active FPGAs that can nonetheless be preempted for a new assignment, Tuan describes techniques for identifying

available cores (e.g., FPGAs) in this context.⁸ For example, Tuan identifies a core as available for a new task if the core is either currently inactive/unused, or if the core's previously assigned task has completed or can be preempted. MS-1005, 32-33 (“When a task completes, the core it occupies is freed and another task can be assigned to the core. ... When tasks *T1* and *T4* finish or are preempted, their correspondent cores are freed”); MS-1003, ¶104. Further, in a “heterogeneous” system in which at least some cores differ in relevant characteristics (e.g., size), Tuan accounts for relevant core characteristics in determining the availability of a core for a given task. MS-1005, 32 (cores identified “based on their sizes or functionalities in order to make the best use of advantages of cores”).

Tuan's disclosed criteria for determining core availability would have been obvious to apply in the combination, particularly to improve the ability of HIPs to identify suitable FPGA IP blocks that can be assigned new stage-instances in a pipeline—e.g., in contrast to FPGA IP blocks that execute tasks that are not currently susceptible to preemption. MS-1003, ¶105. By identifying and selecting FPGA IP blocks that have not yet been assigned a task, or whose current task has

⁸ In another patent involving similar technology, Patent Owner explicitly characterizes cores with preempted tasks as “available” cores. *See* MS-1003, Footnote 5; MS-1026, 18:28-30.

completed or can be preempted, the HIPs in a Kupferschmidt-Tuan combination would avoid interfering with the task of a currently executing stage-instance in a manner that could unacceptably degrade system performance. A POSITA would have sought, for example, to identify available FPGA IP blocks to distinguish them from other FPGA IP blocks that are busy executing other tasks for other stage-instances that cannot be preempted in a timely manner, or without interfering with the execution of higher-priority tasks or pipelines. *Id.* Likewise, by identifying and selecting cores in a heterogeneous system with an FPGA that is sufficiently sized for a particular task, the need to further sub-divide the task into smaller tasks that require multiple cores to execute can be avoided. MS-1005, 32-33; MS-1003, ¶105. This in turn would reduce the number of cores required to run a pipeline, reduce the amount of communications required between cores, and improve throughput of the pipeline. MS-1005, 109 (“With larger core sizes, the number of tasks each application needs to be divided into is reduced; so, less communication is required, and throughput is improved.”); MS-1003, ¶105.

Element [1.5]

As discussed in [1.4], Kupferschmidt and Tuan render obvious identifying available FPGAs that are available to process different processing stages from pipelines of requesting instances of respective programs. *Supra*, [1.4]; MS-1003, ¶¶100-105; *see also id.*, ¶¶106-110. For example, in the combination, HIPs in

pipeline engine 202 identify FPGA-based cores (i.e., FPGA IP blocks) that are available to be assigned stage-instances for new pipeline(s), dynamically updated pipeline(s), or both. *Id.*

In the combination, HIPs in pipeline engine 202 also assign stage-instances to FPGA-based cores selected from the identified available cores. *Id.*; *supra*, Section V.A.3(a); MS-1003, ¶107. To this point, Kupferschmidt explains that the HIP “sets up the software pipeline, [and] assigns threads of execution to stage-instances in the pipeline.” MS-1004, [0088]; *generally id.*, [0075]-[0085], FIGS. 5-6. In other words, the HIP “assigns one or more IP blocks to handle each stage of the pipeline” by pairing each stage-instance in a pipeline with a different FPGA IP block. *Id.* As a result, different FPGA IP blocks are assigned to different processing stages of the requesting instances of the respective programs. MS-1003, ¶107.

A POSITA would have understood the operations for assigning IP blocks to stage-instances (or vice versa) to involve a selection of IP blocks for the assignments that have been identified as available, or at least the selection of such available IP blocks would have been obvious for the reasons described above in [1.4]. MS-1003, ¶108; *supra*, [1.4]. This is further reinforced by Tuan’s teachings for selecting FPGAs that have been identified as available by certain criteria when as-

signing new tasks to cores in a multicore parallel processing system. *Id.* As discussed above in [1.4], Tuan identifies a core as available for a new task if the core is either currently inactive/unused, or if the core's previously assigned task has completed or can be preempted. MS-1005, 32-33; MS-1003, ¶108. In a system with heterogeneous cores, Tuan further teaches that available cores can be also limited “based on their sizes or functionalities in order to make the best use of advantages of cores.” MS-1005, 32. Tuan then selects an available core to which to assign a new task, as illustrated in Tuan's FIG. 2.9, for example:

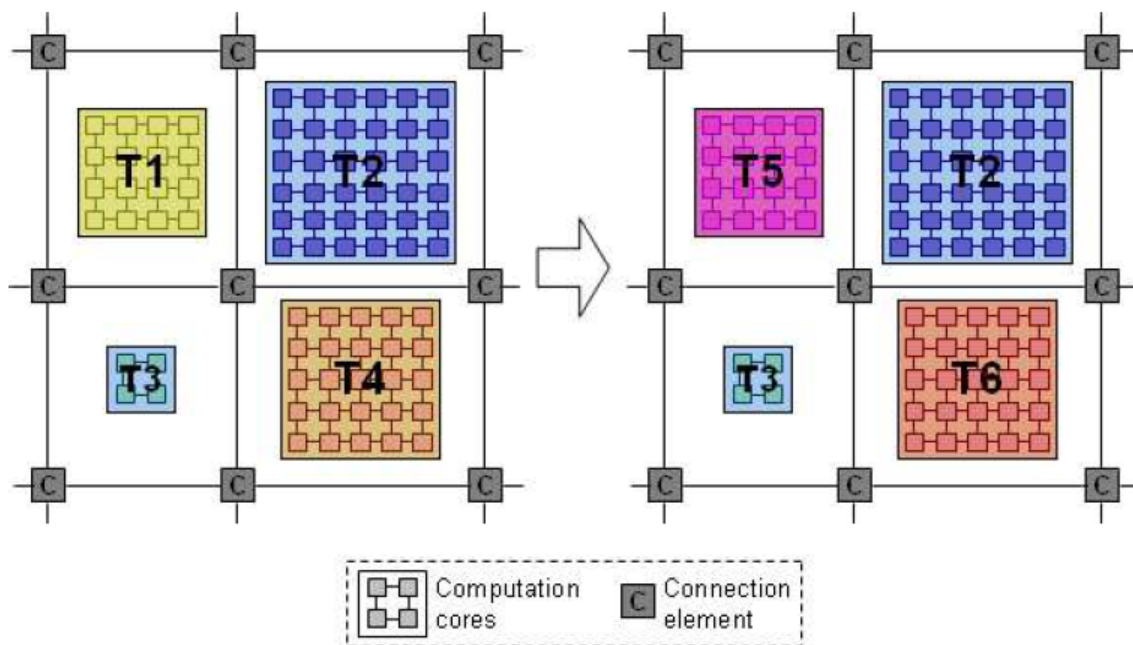


Fig. 2.9: Time-sharing, multicore partitioning model

MS-1005, 33 (depicting new tasks T5 and T6 assigned to selected available cores at the top-left and bottom-right of the array).

Based on Kupferschmidt and Tuan, the HIPs and pipeline engine 202 in the combination would “identify” selected available FPGA IP blocks both when configuring the pipelines of the requesting program instances and when executing the pipelines. MS-1003, ¶109. For example, once an available FPGA IP block has been selected and assigned to a particular stage-instance, the selected FPGA IP block would be identified so that it could be programmed/configured to implement the task that corresponds to the assigned stage-instance. MS-1005, 32-33; MS-1003, ¶109. HIPs would also identify the selected FPGAs in order to properly establish network locations and message pathways between consuming and producing stage-instances in the pipeline. *See* MS-1004, [0081]-[0082]; MS-1003, ¶109. Likewise, during execution of a pipeline, the selected FPGA IP blocks are identified when directing messages between the IP blocks assigned to different stage-instances in the pipeline. MS-1004, [0077] (“sending network-address based communications among other stages”), [0078]-[0079], FIG. 6; MS-1003, ¶109.

In the combination, the selection and identification of available FPGA IP blocks would be based on the conditions of the data processing system as described above in [1.3] and taught in Kupferschmidt. *Supra*, [1.3]; MS-1003, ¶110. These conditions include the amount of work (e.g., the “workload”) requested to be performed in the pipelines, the relative throughputs of different stages in the pipelines, and other monitored conditions indicative of pipeline performance. MS-1004,

[0076]-[0077], [0081]-[0082]. The number of instances per stage of a pipeline, and thus the number of IP blocks required to implement instances of each stage, can vary depending on the workload and other conditions both initially and during runtime, and the HIP can “scale[]” the pipeline by “instantiating additional instances of a stage as needed depending on workload.” MS-1004, [0077]. Similarly, “instances may be dynamically added or removed from a stage as needed for balancing the load among the stages of the pipeline.” *Id.*, [0081], [0094]. Since the number and locations of IP blocks selected depends on the workload and other monitored conditions of the pipelines as taught by Kupferschmidt, the identification of selected FPGAs in the combination would be based at least on the conditions in the data processing system as recited in [1.5]. MS-1003, ¶110.

Element [1.6]

As discussed above with respect to [1.4] and [1.5], it would have been obvious for a system based on Kupferschmidt and Tuan to configure selected FPGA IP blocks to process a respective processing stage of a respective requesting program instance. *Supra*, [1.4]-[1.5], Section V.A.3(a); MS-1003, ¶111. Kupferschmidt, for example, discloses that selected IP blocks are configured to process an assigned stage-instance of a pipeline in a thread of execution of the IP block. MS-1004, [0088] (HIP “assigns one or more IP blocks to handle each stage of the pipeline”; HIP “sets up the software pipeline, assigns threads of execution to stage-instances

in the pipeline”), [0077], [0080]-[0082]. In the combination, the IP blocks are FPGA-based cores as contemplated in Tuan, and configuring the IP blocks as suggested by Tuan would predictably involve configuring the FPGA-based core of the IP block to be capable of executing a task for the stage of the pipeline to which the stage-instance belongs. *See* MS-1005, 32-33 (describing “reconfiguration” in a “multicore partitioning” model); MS-1003, ¶111. As an example, the figure below depicts a set of selected IP blocks, which would be implemented as FPGA-based cores in the combination, and that would be configured according to Kupferschmidt to process respective processing stage-instances of a pair of pipelines of respective requesting program instances:

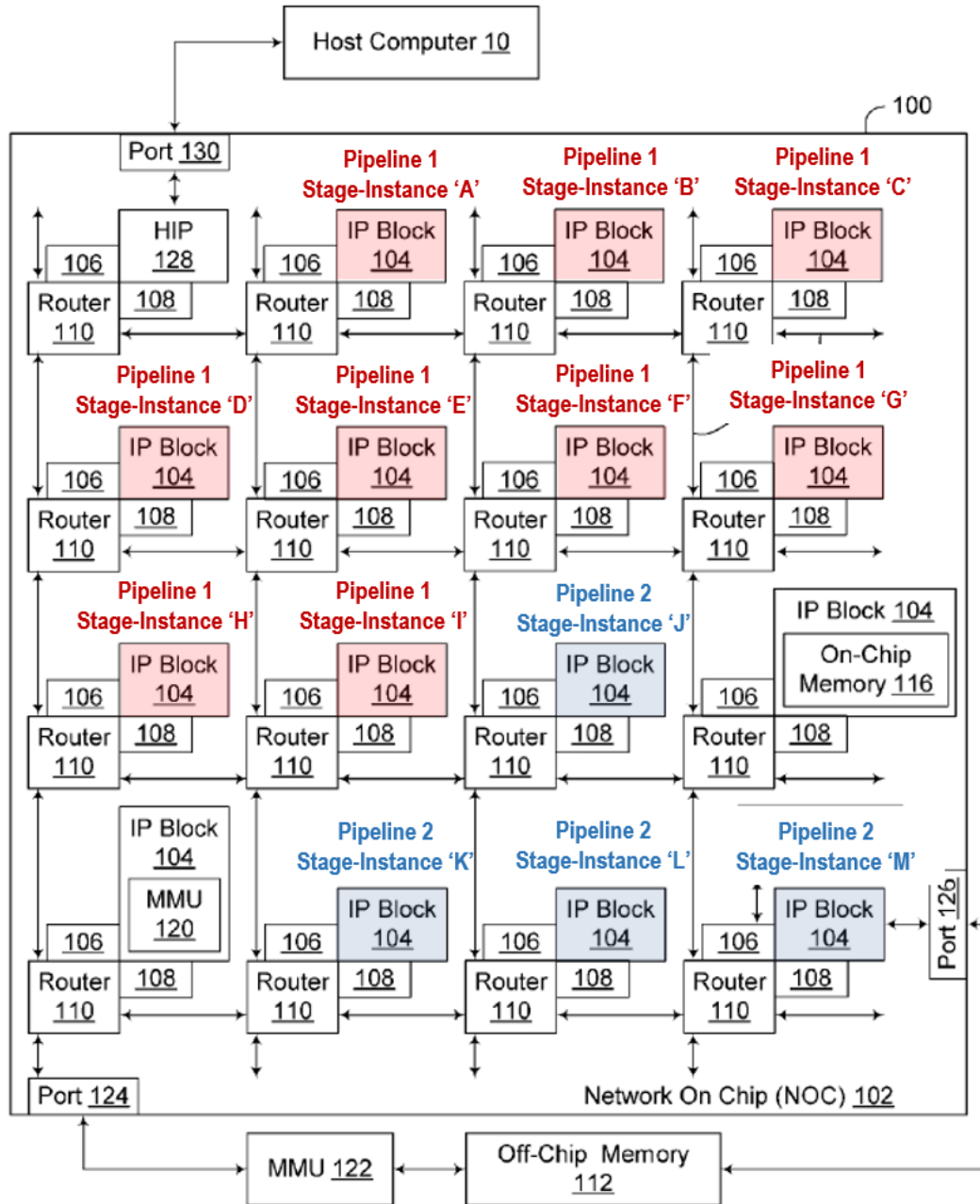


FIG. 2

MS-1004, FIG. 2 (annotated); *supra*, [1.1]-[1.2]; MS-1003, ¶111.

Element [1.7]

As described in detail above with respect to [1.2], Kupferschmidt discloses,

and the combination provides, the use of pipelines to process data through a series of stages of data processing tasks. *Supra*, [1.2], Section V.A.1; MS-1004, [0079]-[0081], FIG. 6. According to Kupferschmidt, data processing in a pipeline “continues to the last stage ..., which then stores its produced output data in an output data structure for eventual return through the HIP 212 to the originating application 216.” MS-1004, [0079], FIG. 5. Thus, the FPGA IP block(s) that were selected and assigned to execute the task(s) of the one or more instances of the last stage of a pipeline are configured to produce “final results” for the pipeline, and to support communicating the final results to the originating application 216, e.g., by storing the final results in an output data structure accessible by the HIP 212 that ultimately returns the results to the application 216.⁹ MS-1003, ¶112. The FPGA IP blocks(s) in the last stage of one or more pipelines thus provide the recited “certain selected FPGAs” of [1.7], as a first mapping. MS-1003, ¶112.

⁹ Petitioner notes that the language of [1.7] only requires the certain selected FPGAs to “**support** communicating” final results to a requesting client, but does not require that the certain selected FPGAs directly communicate the final results to the client. The specification of the ’556 Patent likewise discloses external (network) output ports 50 as the proximate interface for communicating final results to a client rather than any FPGAs. MS-1001, 10:42-55, FIG. 2.

In another example, the HIPs themselves for one or more pipelines constitute “certain selected [FPGAs]” configured to support communicating, by the task executing on the respective FPGA, final results to the originating application 216. MS-1003, ¶113. Kupferschmidt explains that a HIP “is itself typically implemented in an IP block,” which can be an FPGA in the combination with Tuan. MS-1004, [0088]; *see also id.*, [0079] (output data stored “for eventual return through the HIP 212 to the originating application 216”); *supra*, Section V.A.3(a).

Under either of the foregoing mappings of the “certain selected [FPGAs],” the certain selected FPGA IP blocks support communicating final results to a “*requesting client*” over a “*network*” in the data processing system. MS-1003, ¶114. For example, Kupferschmidt discloses that the IP blocks are arranged in a “mesh network” to facilitate communication among IP blocks and other components. MS-1004, [0068]. The mesh network is formed from an array of vertically and horizontally connected nodes, each node comprising an IP block 104, memory communications controller 106, network interface controller 108, and router 110. *Id.*, FIG. 2; *generally id.*, [0048] (“Routers 110, and the corresponding links 118 therebetween, implement the network operations”), [0044]-[0068] (describing details of the mesh network and communication among interconnected nodes and IP blocks). Communicating final results from one IP block to another (e.g., from an IP block for a final stage of a pipeline to a HIP), from an IP block to an output data

structure or memory, or from an IP block to a host interface port (e.g., from a HIP to port 130) all involve communicating the final results over one or more links 118 across one or more network nodes. *Id.*; MS-1003, ¶114. Communication of the final results can further extend across a network to deliver the results from the pipelines to a host (e.g., a host 10 that executes the requesting program instances on a main processor 12). MS-1004, [0052] (“host interface processor 128 is connected to the host computer through a data communications port 130”), FIGS. 1-2; *see also id.*, [0087] (“Application 402 and driver 404 are executed on the CPU”), FIG. 8A; MS-1003, ¶114.

Tuan similarly describes networks in multicore parallel processing systems. *See, e.g.*, MS-1005, 33 (FIG. 2.9 depicting connection elements among cores), 27 (“In a reconfigurable system, several communication mechanisms can be exploited to transfer data between tasks such as ... an interconnection network. ... [U]sing networks allow more flexibility in where tasks can be allocated” and is used “to form a multicore reconfigurable architecture”). Hence, whether FPGA IP blocks in the combination are implemented in a single chip (e.g., a NOC) or across multiple chips, the IP blocks (FPGA-based cores) in either case would form a network that would carry final results from certain selected FPGAs as described above. *See* MS-1004, [0041] (“the invention may be implemented in devices and device architectures other than NOC devices and device architectures”); MS-1003,

¶115.

By sending final results to an “originating application 216,” Kupferschmidt indicates that final results are communicated to a “*requesting client*” as recited in [1.7]. MS-1004, [0079]; MS-1003, ¶116. Application 216 is representative of a program instance that functions as a “client,” which is apparent from the fact that it issues requests from a host to a server (e.g., pipeline engine 202). MS-1004, [0076] (“software application 216 issues requests”), [0079], [0052], FIGS. 1-2, 5-6; *supra*, [1.1]; MS-1003, ¶116. A POSITA would have understood that clients and servers can be implemented on the same or different computers, and either option would have been an obvious and suitable design choice in the combination. MS-1003, ¶116.

Claim [2]

Kupferschmidt discloses that “[e]ach stage” in a pipeline “implements a producer/consumer relationship with a next stage,” such that each stage after the initial stage receives data produced from a prior stage, and operates on the received data to produce further data to be processed by a subsequent stage or to be returned as final results. MS-1004, [0079]-[0080], FIG. 6. One or more instances of each stage can be invoked in parallel, with each stage-instance implemented on a different FPGA IP block in the Kupferschmidt-Tuan combination. MS-1004, [0088]; *supra*, [1.2]-[1.4], Section V.A.3(a); MS-1003, ¶¶117-119.

In this context, the system in the resulting combination would configure one or more selected available FPGA IP blocks (*a portion of the selected [FPGAs]*) to implement respective instance(s) of the penultimate stage of a pipeline and to carry out the task(s) of the penultimate stage. MS-1004, [0075]-[0080]. In so doing, the FPGA IP block(s) for the penultimate stage would be configured to communicate intermediate results of the different tasks over the network (e.g., a mesh network (*see supra*, [1.7]) to the FPGA IP blocks that implement respective instance(s) of the last stage of the pipeline—i.e., “certain selected [FPGAs]” as recited in [1.7] and [2]. MS-1004, [0079] (“producer/consumer relationship”), [0080] (“instances may have a one-to-one, a one-to-many and/or a many-to-one relationship with other instances in the pipeline”), [0082] (“Each stage is configured with a stage ID for each instance of a next stage”); *supra*, [1.7]; MS-1003, ¶119. The certain selected FPGA IP blocks in the final stage would then use the intermediate results from the prior stage to produce final results. MS-1004, [0079] (“operations continue[] to the last stage of the pipeline, which then stores its produced output data in an output data structure for eventual return”).

Claim [4]

As discussed above with respect to [1.3] and [1.5], Kupferschmidt discloses various conditions that can impact the configuration of a pipeline such as the

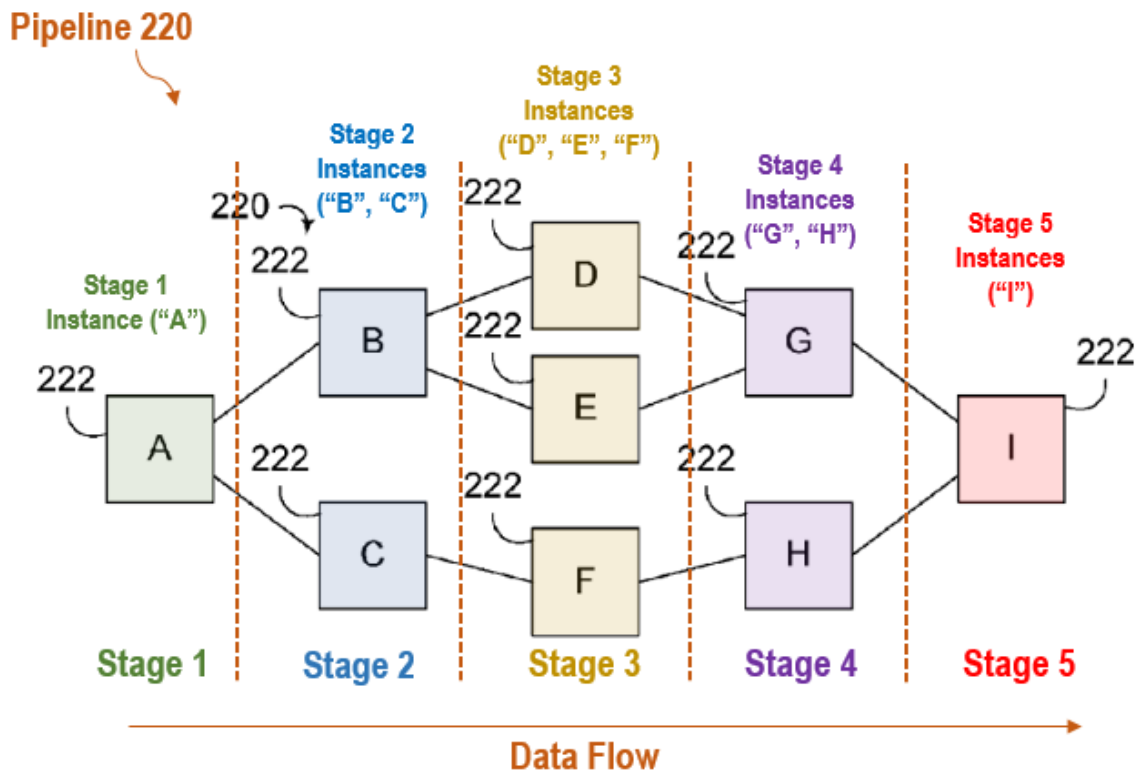
amount of work issued, stage-specific throughputs, and other monitored performance metrics. MS-1004, [0077], [0081]-[0082]; *supra*, [1.3], [1.5]. In the context of Kupferschmidt (and the resulting combination with Tuan), the amount of work, i.e., the workload, that a program requests a pipeline to process provides a “demand expression” of the program.¹⁰ MS-1004, [0075]-[0077]; MS-1003, ¶¶120-121. This follows since Kupferschmidt measures the workload of a program instance, and uses the workload to appropriately scale stages in the pipeline. *Id.* For example, Kupferschmidt teaches that a pipeline can be scaled up when the workload increases and scaled down when the workload decreases. MS-1004, [0077] (“a pipeline may be scaled by instantiating additional instances of a stage as needed depending on workload”), [0081] (“instances may be dynamically added or removed from a stage as needed for balancing the load”), [0082]; MS-1003, ¶121.

Claim [5]

Kupferschmidt discloses that a processing stage in a pipeline can include two or more parallel copies of a given task. MS-1004, [0080] (“Instances may operate collectively with one another in a particular stage to ***perform parallel tasks***”

¹⁰ The plain language of the term “demand expressions” does not restrict claim 4 to changes in “core demand ***figures***” (CDFs) like those described in particular embodiments of the ’556 Patent.

and share the workload, thus improving the overall throughput of the stage in performing the task.”), [0084] (“using a plurality of parallel threads of execution”), [0085] (“parallel threads of execution”), [0093] (same). The pipeline depicted in Figure 6 of Kupferschmidt, for example, includes multiple stage-instances that can be implemented as parallel copies of respective tasks in each of stages 2, 3, and 4:



MS-1004, FIG. 6 (annotated).

In the combination based on Kupferschmidt and Tuan, a parallelized processing stage would be implemented by configuring a subset of the selected FPGA IP blocks as instances of the parallelized processing stage to perform a given task

of the program associated with the pipeline. MS-1004, [0077], [0080]-[0081], [0088]; *supra*, [1.6], Section V.A.3(a); MS-1003, ¶¶122-124.

Claim [6]

Kupferschmidt discloses techniques for implementing dynamic load balancing among stages in a pipeline. *See, e.g.*, MS-1004, [0077], [0081], [0082], [0088].

“Such load balancing can be carried out, for example, by monitoring the performance of the stages and instantiating a number of instances of each stage in dependence upon the performance of one or more of the stages.” *Id.*, [0082].

“[W]orkload monitoring may be performed during runtime, and [] instances may be dynamically added or removed from a stage as needed for balancing the load among the stages of the pipeline.” *Id.*, [0081]; *see also id.*, [0077] (“pipeline may be scaled”) (“no single stage presents a bottleneck to performance”), [0088] (“monitors workflow to dynamically reallocate threads of execution to different stages of the pipeline to maximize throughput and minimize bottlenecks”).

A POSITA would have understood Kupferschmidt’s workload monitoring and load balancing operations to be repeatedly performed over a plurality of iterations during runtime of each pipeline. MS-1004, [0081] (“performed during runtime”); MS-1003, ¶¶125-128. Indeed, Kupferschmidt explains that load balancing events are prompted by conditions premised on monitored performance statis-

tics over a “particular time period,” and indicates that monitoring and load balancing are ongoing processes that occur repeatedly during execution of a pipeline.

MS-1004, [0082]; *see also id.*, [0077] (“a pipeline may be scaled by instantiating additional instances of a stage *as needed* depending on workload”), [0081] (“workload monitoring may be performed *during runtime*, and that instances may be dynamically added or removed from a stage *as needed*”), [0082] (“a separate monitoring application that in turn is installed *and running* on another thread of execution on an IP block or HIP”). The workload of a pipeline is not static, and a POSITA would have understood Kupferschmidt’s description of performing load balancing “as needed” during runtime to imply that load balancing can be performed repeatedly based on monitored conditions rather than being a single, isolated event. MS-1003, ¶127. Moreover, repeatedly performing load balancing to add or remove stage-instances in a pipeline as needed would have been obvious to optimize performance, maximize throughput, and minimize bottlenecks throughout runtime of the pipeline even as the workload and system conditions change. MS-1003, [0088]; MS-1003, ¶127.

Repeatedly load balancing by adding or removing stage-instances in a pipeline as needed would occur by repeatedly configuring the structures of the FPGAs for a plurality of iterations, thereby dynamically adjusting performance of the different tasks at different stages of the pipeline based at least on the conditions. MS-

1004, [0077], [0080]-[0081], [0088]; *supra*, [1.6], Section V.A.3(a); MS-1003, ¶128.

Claim [7]

Kupferschmidt discloses, and the combination provides, that “workload monitoring may be performed during runtime, and that instances may be dynamically added *or removed* from a stage as needed for balancing the load among the stages of the pipeline.” MS-1004, [0081]; *see also id.*, [0077], [0081], [0082], [0088]; *supra*, [5]-[6]. Thus, for example, when a pipeline is initially setup on the network of FPGA IP blocks with two or more instances in a stage as two or more parallel copies of a task for that stage, and conditions then dictate that at least one of those stage-instances should be removed for scaling or load balancing purposes (even as other instances of the same stage remain executing the same task), the system would achieve the features recited in [7]. MS-1003, ¶¶129-130.

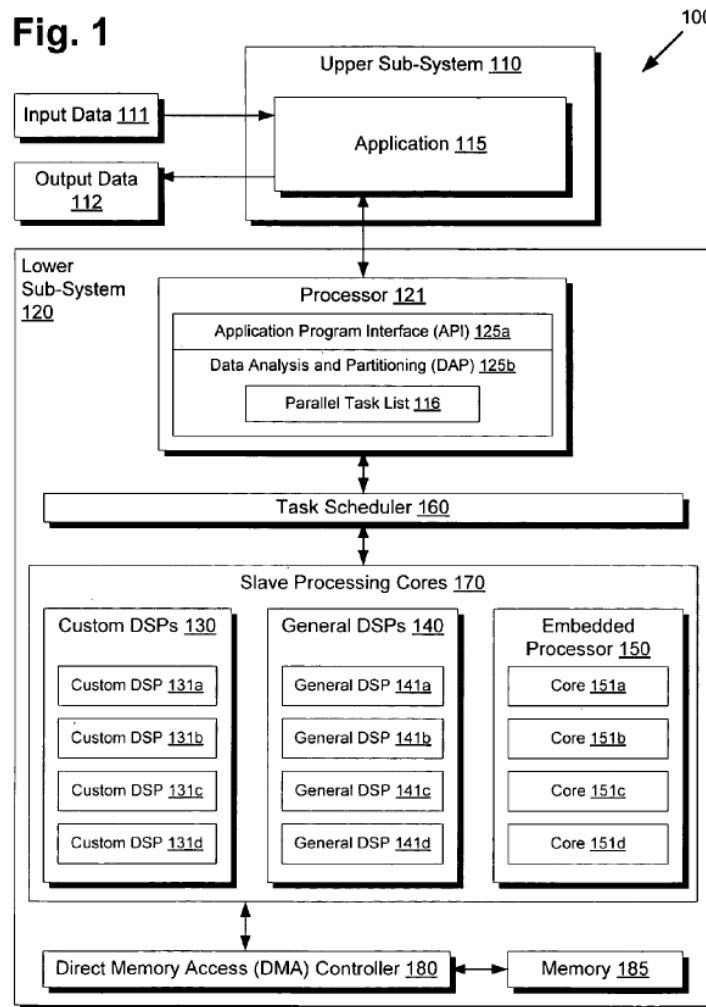
Claim [8]

Kupferschmidt discloses, and the combination provides, that parallel copies of stage-instances can be “dynamically ... removed from a stage as needed,” and that “a pipeline may be scaled ... as needed depending on workload.” MS-1004, [0077], [0081]; *supra*, [5]-[7]; MS-1003, ¶¶131-132. When the workload for a program decreases, the pipeline is scaled down by removing stage-instance(s) as described above with respect to [7]. MS-1003, ¶132.

B. [Ground 1B]: Kupferschmidt in view of Tuan and Brent (Claims 1-8)

1. Overview of Brent

Brent generally describes a “distributed multi-core system with an adaptive scheduler.” MS-1006, Abstract, [0014]; MS-1003, ¶¶52-53. Figure 1, reproduced below, depicts an embodiment of Brent’s system configured to process tasks of an application 115 on various slave processing cores 170. *Id.*, [0014]-[0017].



MS-1006, FIG. 1.

In Brent, a task scheduler “select[s] a subset of tasks from [a] parallel task list” and assigns the selected tasks “to their respective preferred core types” for execution. *Id.*, [0027], FIGS. 2-3; *generally id.*, [0026]-[0030]. In assigning tasks to cores for execution, the task scheduler obtains information that indicates the availability of each core including whether the core is “Busy” or in an “Idle” state. *Id.*, [0021]-[0023] (describing “idle cores” and task-to-core assignment techniques).

2. Obviousness Based on Kupferschmidt-Tuan-Brent Combination

(a) Overview of the Kupferschmidt-Tuan-Brent Combination

As described above in Ground 1A, the Kupferschmidt-Tuan combination renders obvious claim 1 including the “identifying available [FPGAs]” and “identifying selected [FPGAs]” steps of [1.4] and [1.5] respectively. *Supra*, Section V.A.3(a), [1.4]-[1.5]. Kupferschmidt and Tuan nonetheless lack detail as to how the system specifically keeps track of available FPGA-based cores (IP blocks) such that the available FPGAs can be assigned stage-instances of a pipeline. MS-1003, ¶134. For such details, a POSITA would have turned to Brent. *Id.* For example, Brent describes techniques for maintaining information about the availability of cores—including whether a given core is “Busy” or “Idle”—and then assigning tasks to available cores (e.g., cores that are not busy). MS-1006, [0021] (“Task

scheduler 260 may then analyze the state of slave processing cores 270 to determine free cores to distribute tasks. As shown in FIG. 2, some cores are already in the Busy state, which may indicate that the cores are locked from previously assigned processes and are presently unavailable.”), [0022]-[0023] (describing “idle cores” and task-to-core assignment techniques), FIG. 2.

It would have been obvious to apply Brent’s teachings in combination with Kupferschmidt and Tuan so that the HIPs and pipeline engine 202 would access availability information about the FPGA-based cores (IP blocks), and would use such information to identify and select cores to which stage-instances in one or more pipeline(s) could be assigned. MS-1003, ¶135. A POSITA would have been motivated to combine Brent with Kupferschmidt and Tuan in this manner to achieve known and predictable benefits, including to provide a straightforward and efficient method of determining core availability, and to improve the combination’s core selection and task / stage-instance assignment process to avoid undesirably interfering with previously assigned stage-instances that may be currently executing on a core or that a pipeline may be expecting to execute in the future. MS-1003, ¶135. The addition of Brent in the combination also would have been obvious since it merely involves the use of a known solution for core selection and task assignment as taught in Brent to improve similar systems and methods in the same way. *KSR*, 550 at 417; MS-1003, ¶135.

**(b) Mapping of the Kupferschmidt-Tuan-Brent
Combination to Claims 1-8**

In combination (*see supra*, Section V.B.2(a)), the teachings of Kupferschmidt, Tuan, and Brent render obvious claims 1-8 of the '556 Patent, and the combination provides the features recited in these claims at least the reasons addressed below. MS-1003, ¶136.

Claims 1-2 and 4-8

The teachings of Kupferschmidt in view of Tuan and Brent provide the respective elements of claims 1-2 and 4-8 for the reasons discussed above in Ground 1A. *Supra*, Section V.A.3(b); MS-1003, ¶136. Additionally, the Kupferschmidt-Tuan-Brent combinations discloses and renders obvious [1.4] and [1.5] further based on the teachings of Brent. As discussed above, Brent expressly describes a task scheduler that first identifies available cores, and then selects and identifies selected available cores to which tasks can be assigned and executed. MS-1006, [0021] (“determine free cores to distribute tasks”); *see also id.*, [0022], [0029] (“accommodate the number of free course available”), FIG. 2. In the combination, and based on Brent, HIPs and pipeline engine 202 would access availability information about the FPGA-based cores (IP blocks), would use such information to identify available cores, and would then select and identify selected available cores to which stage-instances in one or more pipeline(s) could be assigned and exe-

cuted. But for the addition of FPGA (core) availability information, the combination otherwise provides [1.4] and [1.5] (and the remaining claim elements) as described in Ground 1A. MS-1003, ¶136.

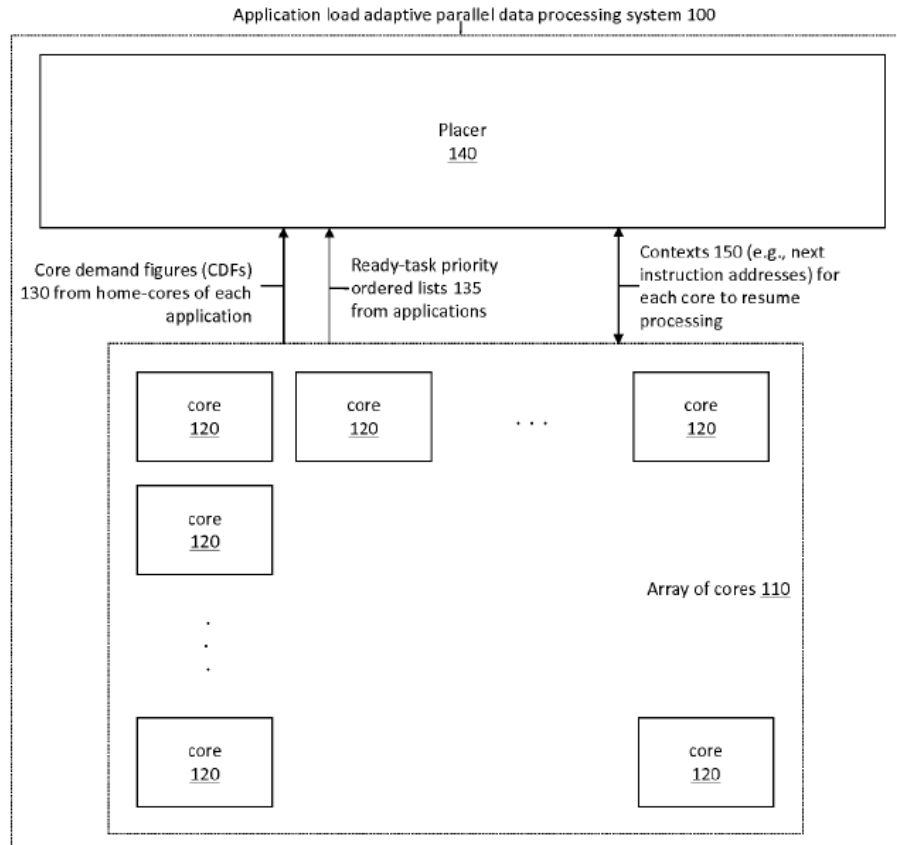
Claim [3]

As discussed above, Brent’s teachings in the combination disclose maintaining availability information for identifying availability of the FPGAs (FPGA-based core IP blocks). MS-1006, [0021] (“some cores are already in the Busy state, which may indicate that the cores are locked from previously assigned processes and are presently unavailable” while others are “Idle”), [0029] (“accommodate the number of free cores available”), FIG. 2; *supra*, Section V.B.2(a), Claim 1; MS-1003, ¶¶137-138. Brent further teaches that the availability information can be changed to indicate that a particular core (i.e., an FPGA in the combination) is in use, e.g., by identifying the core as “Busy.” *Id.*

C. [Ground 1C]: Kupferschmidt in view of Tuan and Sandstrom-501 (Claim 4)

1. Overview of Sandstrom-501

Sandstrom-501 is a pre-grant publication of an earlier-filed U.S. patent application by the named inventor of the ’556 Patent, Mark Sandstrom. *See* MS-1010; MS-1003, ¶¶54-57. Like the ’556 Patent, Sandstrom-501 discloses a multi-core data processing system configured to perform parallel processing of tasks from one or more applications. MS-1010, [0033]-[0036].



MS-1010, FIG. 1.

As shown in FIG. 1, Sandstrom-501's data processing system includes a "hardware logic based placer module 140." MS-1010, [0036]. The placer module 140 implements logic to "allocate[] and assign[] the cores 120 of the system 100 among the set of applications and their tasks, at least in part based on the [core-demand-figures (CDFs)] 130 of the applications." *Id.* The CDF 130 signals to the placer 140 "a level of demand of the system processing capacity by the [] applications," and "[i]n an embodiment, these indicators 130[] ... express how many cores

120 their associated application program is presently able [to] utilize for its data processing tasks.” *Id.*, [0035]; *see also id.*, (“[I]n an embodiment, the CDF 130 of a given application program is a function of the number of its schedulable tasks, such as processes, threads or functions (called collectively as tasks) that are ready to execute at a given time. In a particular embodiment of the invention, CDF of an application program expresses on how many processing cores the program is presently able to execute in parallel.”).

Sandstrom-501 further teaches that the “application task to core placement process” can be exercised “periodically, e.g. at even intervals such as once per a given number ... of processing core clock or instruction cycles.” *Id.*, [0036]. Alternatively, the “application task to core placement process” can be exercised when there is a change in conditions, e.g. “based on a change in the CDFs 130 of the applications 220.” *Id.*; *see also id.*, Claim 14 (“wherein the allocating is performed following a change in a capacity demand indicator of at least one program among the set of application programs”).

2. Obviousness Based on Kupferschmidt-Tuan-Sandstrom-501 Combination

Claim [4]

As discussed above, Sandstrom-501 describes conditions in a data processing system that prompt the identification and selection of processors to execute different tasks of respective programs running on the system. *Supra*, Section

V.C.1. In some examples, an “application task to core placement process” is initiated upon detecting a change in system conditions, e.g. “based on a change in the CDFs 130 of the applications 220.” MS-1010, [0036]; *see also id.*, Claim 14 (“wherein the allocating is performed following a change in a capacity demand indicator of at least one program among the set of application programs”). The CDF 130 signals to the placer 140 “a level of demand of the system processing capacity by the [] applications,” and “[i]n an embodiment, these indicators 130[] ... express how many cores 120 their associated application program is presently able [to] utilize for its data processing tasks.” *Id.*, [0035]; *see also id.*, [0036], FIG. 1.

It would have been obvious to apply the teachings of Sandstrom-501 in the combination with Kupferschmidt and Tuan (*see supra*, Section V.A.(3)(a)) such that program instances that request work in a pipeline as taught in Kupferschmidt would explicitly signal to the HIPs “a level of demand” associated with the request as taught in Sandstrom-501. MS-1010, [0035]; MS-1003, ¶¶139-142. Sandstrom’s expression of the demand level is not restricted to a particular form, and in the combination, a demand expression could predictably indicate the size of the workload requested by the program instance, or the number of stage-instances (and hence the number of FPGA IP blocks) requested to perform the work akin to Sandstrom’s “CDF.” *Id.* Based on Kupferschmidt, the HIPs in the resulting combination would predictably scale the pipeline and/or perform load balancing among

stages based on changes in the programs' expressed levels of demand. MS-1010, [0036], Claim 14; MS-1004, [0077], [0080]-[0082]; MS-1003, ¶142.

A POSITA would have been motivated to configure the HIPs in the combination to setup or dynamically adjust pipelines according to changes in programs' demand expressions in order to improve their ability to appropriately size and scale pipelines to accommodate programs' workloads. MS-1004, [0077], [0081]-[0082]; MS-1003, ¶143. Additionally, more explicitly signaling levels of demand would have facilitated better utilization of FPGA IP blocks among different programs by allocating more FPGA IP blocks to pipelines of programs with higher demand (e.g., higher workload) and fewer FPGA IP blocks to pipelines of programs with lower demand. *Id.* The addition of Sandstrom in the combination also would have been obvious since it merely involves the use of a known solution for expressing programs' processing demands as taught in Sandstrom to improve similar systems and methods in the same way. *KSR*, 550 at 417; MS-1003, ¶143.

D. [Ground 1D]: Kupferschmidt in view of Tuan, Brent, and Sandstrom-501 (Claim 4)

It would have been obvious to further modify the Kupferschmidt-Tuan-Brent combination of Ground 1B in view of Sandstrom-501 for substantially the same reasons discussed above in Ground 1C, and the resulting combination would have rendered claim 4 obvious also for substantially the same reasons discussed above in Ground 1C. MS-1003, ¶144.

VI. PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION

A. § 314(a) - *Fintiv*

The *Fintiv* factors weigh heavily against exercise of the Board’s discretion to deny institution under 35 U.S.C. § 314(a). See *Apple Inc. v. Fintiv, Inc.*, IPR2020-00019, Paper 11 (PTAB Mar. 20, 2020) (precedential) (“*Fintiv*”).

Factor 1 strongly favors institution. ThroughPuter currently asserts the ’556 Patent against Microsoft in litigation before the U.S. District Court for the Western District Washington (Case No. 2:22-cv-344-BJR) (“the Related Litigation”). MS-1101 (original complaint); MS-1002 (first amended complaint in proceeding consolidated with the Related Litigation). Microsoft moved for a stay of the Related Litigation on April 18, 2022, and the Court granted Microsoft’s motion on August 18, 2022. MS-1103. Specifically, the Court’s order stayed the Related Litigation “pending resolution of Microsoft’s petitions for *inter partes* review.” *Id.* The stay of the Related Litigation “strongly weigh[s] against exercising the authority to deny institution.” *Fintiv*, 6. As the Board recognized in *Fintiv*, “[a] district court stay of the litigation pending resolution of the PTAB trial allays concerns about inefficiency and duplication of efforts.”

Factor 2 strongly favors institution. With the Related Litigation stayed pending resolution of Microsoft’s IPRs, no trial date is currently set in the district

court proceeding. The Board's final written decision in the present IPR will necessarily precede any district court trial while the stay remains in place pending the PTAB's consideration of Microsoft's IPRs.

Factor 3 strongly favors institution. At the time the stay was ordered, the Related Litigation was still in an early phase. *Markman* had not yet occurred, and the parties had not even exchanged terms for construction let alone begun claim construction briefing. Likewise, no experts had weighed in on any issue in the proceeding, and Microsoft had not served any invalidity contentions. The parties' and the Court's investment in the Related Litigation has been minimal.

Factor 4 favors institution. With the Related Litigation having been stayed, there will be no overlapping issues between an IPR instituted based on this Petition and the district court proceeding. And, as noted in Factor 3, Microsoft has served no invalidity contentions in the Related Litigation pertaining to the '556 Patent.

Factor 5 (whether petitioner is also the defendant in district court) is neutral.

Factor 6 strongly favors institution. As demonstrated below, the merits of this Petition are particularly strong. This is also Microsoft's only challenge to the '556 Patent that has ever been or is currently before the Board, making considerations related to follow-on petitions moot.

In short, not one of the *Fintiv* factors weighs against institution, and at least five factors weigh strongly in favor of institution. Taken together, the Board

should not exercise discretion to deny institution of IPR.

B. § 325(d) – *Advanced Bionics*

This Petition presents new prior art and arguments against the Challenged Claims for which the file history contains no indication that the Office ever previously considered in connection with the '556 Patent. Indeed, neither Kupferschmidt nor Tuan—the only references applied in this Petition against independent claim 1 and dependent claims 2 and 5-8—were previously cited or applied in a rejection during prosecution of the '556 Patent.

Significantly, and as addressed above (*supra*, Section II.B), the Examiner never issued a rejection during the '556 Patent's prosecution, and instead issued a notice of allowance little more than two months after the filing of the application that issued as the '556 Patent. MS-1002, 9-19. Because the '556 Patent's prosecution did not involve negotiation between the applicant and the Examiner regarding the teachings of the prior art applied in the instant Petition, the instant Petition's art and arguments cannot be said to be the same as, or substantially similar to, art and arguments previously presented to the Office in connection with the '556 Patent. The first prong of the Board's precedential decision in *Advanced Bionics LLC v. MED-EL Elektromedizinische Gerate GmbH* is thus not satisfied. IPR2019-01469, Paper 6 (PTAB Feb. 13, 2020) (precedential) (citing *Becton, Dickinson and Co. v.*

B. Braun Melsungen AG, IPR2017-01586 Pap. 8 (PTAB Dec. 15 2017) (precedential)).

Petitioner acknowledges that Brent and Sandstrom-501, i.e., the tertiary and quaternary references applied in Grounds 1B-1D, are listed on the face of the '556 Patent as having previously been cited to the Examiner. *See* MS-1001, 1-2. But Brent and Sandstrom-501 were just two references among well over a hundred others that were cited to the Examiner. The file history includes no indication that the Examiner ever specifically considered Brent in relation to the additional features recited in [1.4]-[1.5] or claim 3, or Sandstrom-501 in relation to the additional features recited in claim 4 of the '556 Patent.

Even if the mere citation of Brent and Sandstrom-501 were sufficient to implicate the first prong of the *Advanced Bionics* framework, the second prong still would not be satisfied. The Examiner erred in overlooking the pertinent teachings of Brent and Sandstrom-501 as described in this Petition, and the Examiner could not have considered either Brent or Sandstrom-501 in combination with Kupferschmidt and Tuan since these additional references were not cited during original prosecution of the '556 Patent. Under these circumstances, Petitioner submits that exercise of the Board's discretion under § 325(d) is unwarranted.

VII. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8(a)(1)

A. Real Party-In-Interest Under 37 C.F.R. § 42.8(b)(1)

Petitioner, Microsoft Corporation, is the real party-in-interest.

B. Related Matters Under 37 C.F.R. § 42.8(b)(2)

ThroughPuter, Inc.—the alleged Patent Owner—filed a complaint against Microsoft on April 13, 2022, in the U.S. District Court for the Western District of Washington (Case No. 2:22-cv-00492-BJR). MS-1101. The district court subsequently consolidated this action with case 2:22-cv-00344-BJR (“the Related Litigation”), which involved nine other patents that ThroughPuter had previously asserted against Microsoft in 2021. MS-1102. The Related Litigation is currently stayed. MS-1103. ThroughPuter’s complaint asserting the ’556 Patent was served on Microsoft on April 19, 2022.

Microsoft has previously filed IPR petitions against ThroughPuter patents involved in the Related Litigation in IPR2022-00527, IPR2022-00528, IPR2022-00574, IPR2022-00757, and IPR2022-00758.

C. Lead And Back-Up Counsel Under 37 C.F.R. § 42.8(b)(3)

Microsoft provides the following designation of counsel.

Lead Counsel	Backup counsel
W. Karl Renner, Reg. No. 41,265 Fish & Richardson P.C. 60 South Sixth Street, Suite 3200 Minneapolis, MN 55402 Tel: 202-783-5070	David L. Holt, Reg. No. 65,161 Nicholas Stephens, Reg. No. 74,320 Jennifer Huang, Reg. No. 64,297 Fish & Richardson P.C. 60 South Sixth Street, Suite 3200

Fax: 877-769-7945 Email: IPR42342-0121IP1@fr.com	Minneapolis, MN 55402 Tel: 202-783-5070 Fax: 877-769-7945 PTABInbound@fr.com
--	---

D. Service Information

Please address all correspondence and service to the address listed above.

Petitioner consents to electronic service by email at IPR42342-0121IP1@fr.com

(referencing No. 42342-0121IP1 and cc'ing PTABInbound@fr.com, axf-ptab@fr.com, holt2@fr.com, nstephens@fr.com, and jjh@fr.com.

VIII. PAYMENT OF FEES – 37 C.F.R. § 42.103

Microsoft authorizes the Patent and Trademark Office to charge Deposit Account No. 06-1050 for the fee set in 37 C.F.R. § 42.15(a) for this Petition and further authorizes payment for any additional fees to be charged to this Deposit Account.

IX. CONCLUSION

Microsoft requests institution of IPR and cancellation of all Challenged

Claims based on Grounds 1A-1D.

Respectfully submitted,

Dated: November 7, 2022

/W. Karl Renner/
W. Karl Renner, Reg. No. 41,265
David L. Holt, Reg. No. 65,161
Nicholas Stephens, Reg. No. 74,320
Jennifer Huang, Reg. No. 64,297
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402

Tel: 202-783-5070

Fax: 877-769-7945

(Control No. IPR2022-01566)

Attorneys for Petitioner

CERTIFICATION UNDER 37 CFR § 42.24

Under the provisions of 37 CFR § 42.24(d), the undersigned hereby certifies that the word count for the foregoing Petition for *Inter partes* Review totals 13,939 words, which is less than the 14,000 allowed under 37 CFR § 42.24.

Dated: November 7, 2022

/W. Karl Renner/
W. Karl Renner, Reg. No. 41,265
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
Tel: 202-783-5070
Fax: 877-769-7945

Attorney for Petitioner

CERTIFICATE OF SERVICE

Pursuant to 37 CFR §§ 42.6(e)(4)(i) *et seq.* and 42.105(b), the undersigned certifies that on November 7, 2022, a complete and entire copy of this Petition for *Inter Partes* Review and all supporting exhibits were provided via Federal Express to the Patent Owner by serving the correspondence address of record as follows:

Gardella Grace P.A.
80 M Street SE, 1st Floor
Washington DC 20003

/Diana Bradley/
Diana Bradley
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
(858) 678-5667