

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Bismuth et al.
U.S. Patent No.: 12,271,636 Attorney Docket No. 54788-0026IP1
Issue Date: April 8, 2025
Appl. Serial No.: 18/502,275
Filing Date: November 6, 2023
Title: ANALYTICS, ALGORITHM ARCHITECTURE, AND DATA
PROCESSING SYSTEM AND METHOD

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. 12,271,636 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42**

TABLE OF CONTENTS

I.	REQUIREMENTS FOR IPR	1
A.	Grounds for Standing	1
B.	Challenge and Relief Requested	1
II.	THE '636 PATENT	2
A.	Overview	2
B.	Prosecution History	3
III.	LEVEL OF ORDINARY SKILL	4
IV.	CLAIM CONSTRUCTION	4
V.	THE CHALLENGED CLAIMS ARE UNPATENTABLE.....	5
A.	GROUND 1A: Dalal-Ng Renders Claims 1–3, 6–12 Obvious.....	5
1.	Dalal	5
2.	Ng	8
3.	Motivation to Combine	10
4.	Claim Element Analysis.....	15
B.	GROUND 1B: Dalal-Ng-Raghava Renders Claims 4, 5 Obvious	51
1.	Raghava	51
2.	Dalal-Ng-Raghava Combination.....	52
3.	Claim Element Analysis.....	54
C.	GROUND 2A: Heil-Jun Renders Obvious Claims 1–4, 6–9, 11–12.....	56
1.	Heil	56
2.	Jun.....	62
3.	Heil-Jun Combination	63
4.	Analysis	67
D.	GROUND 2B: Heil-Jun-Raghava Renders Obvious Claims 5, 10.....	96
1.	Heil-Jun-Raghava Combination.....	96
2.	Analysis	101
VI.	PAYMENT OF FEES	102
VII.	MANDATORY NOTICES—37 C.F.R. §42.8(a)(1)	102
A.	Real Party-In-Interest—37 C.F.R. §42.8(b)(1)	102
B.	Related Matters—37 C.F.R. §42.8(b)(2)	102
C.	Lead And Backup Counsel—37 C.F.R. §42.8(b)(3).....	102
D.	Service Information.....	103

EXHIBITS

AMD1001	U.S. Patent No. 12,271,636 to Bismuth et al. (“the ’636 patent”)
AMD1002	Excerpts from the Prosecution History of the ’636 Patent (“the Prosecution History”)
AMD1003	Declaration of Dr. Donald Alpert
AMD1004	U.S. Publication No. 2019/0109793 (“Dalal”)
AMD1005	U.S. Publication No. 2019/0114538 (“Ng”)
AMD1006	U.S. Publication No. 2019/0108145 (“Raghava”)
AMD1007	U.S. Patent No. 6,731,649 (“Silverman”)
AMD1008	U.S. Patent No. 9,043,526 (“Freking”)
AMD1009	U.S. Publication No. 2015/0301900 (“Whitehead”)
AMD1010	U.S. Publication No. 2016/0306668 (“Heil”)
AMD1011	Jun et al. “BlueDBM: An Appliance for Big Data Analytics” ACM (June 13, 2015) (“Jun”)
AMD1012	Liu et al. “BlueFlash : a reconfigurable flash controller for BlueDBM” Massachusetts Institute of Technology (2014) (“Liu”)
AMD1013	Declaration of June Ann Munford
AMD1014	Putnam et al. “A Reconfigurable Fabric for Accelerating Large- Scale Datacenter Services” Microsoft, (2014)

AMD1015	Vipin et al. “FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications” ACM Comput. Surv. Vol. 51, No. 4, Article 72 (July 2018)
AMD1016	U.S. Patent No. 7,606,362 (“Streicher”)
AMD1017	Kempahonnaiah “MultiBoot and Fallback Using ICAP in UltraScale+ FPGAs” XAPP1296 (v1.0) (June 23, 2017)
AMD1018	Verma et al. “PCIe BUS: A State-of-the-Art-Review” (2017)
AMD1019	[RESERVED]
AMD1020	U.S. Patent No. 11,169,727 (“Doucette”)
AMD1021	U.S. Patent No. 10,802,995 (“Singh”)
AMD1022	Park et al. “PFFS: A Scalable Flash Memory File System for the Hybrid Architecture of Phase-change RAM and NAND Flash” (2008)
AMD1023	Prasad et al. “Design and Verification of Phy Interface for PCIe Gen 3.0 and USB Gen 3.1 Using UVM Methodology” (2017)
AMD1024	U.S. Patent No. 11,055,252 (“Constantinides”)
AMD1025	“AXI Central Direct Memory Access v4.1: LogiCORE IP Product Guide” (April 4, 2018)
AMD1026	“PCI Express Base Specification Revision 1.0” (April 29, 2002)
AMD1027	IEEE “IEEE Standard for In-System Configuration of Programmable Devices” IEEE Std 1532™ (2002)
AMD1028	U.S. Patent No. 10,831,376 (“Jung”)

AMD1029	[RESERVED]
AMD1030	IEEE “The Authoritative dictionary of IEEE Standards Terms, Seventh Edition” (2000)
AMD1031	[RESERVED]
AMD1032	McCarthy et al. “Visualizing the Five-dimensional Torus Network of the IBM Blue Gene/Q” (2014)
AMD1033	“Serial ATA: High Speed Serialized AT Attachment” Revision 1.0a (January 7, 2003)
AMD1034	U.S. Patent No. 6,711,701 (“Roohparvar”)
AMD1035	U.S. Publication No. 2013/0318285 (“Pignatelli”)
AMD1036	[RESERVED]
AMD1037	Ousterhout et al. “The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM” SIGOPS Operating Systems Review, Vol. 43, No. 4, December 2009, pp. 92-105.
AMD1038	Kang et al. “Performance Trade-Offs in Using NVRAM Write Buffer for Flash Memory-Based Storage Devices” IEEE Transactions on Computers, Vol. 58, No. 6, June 2009, pp. 744-758
AMD1039	Zazo et al. “A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances” 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig) (2015)
AMD1040	U.S. Patent No. 3,995,119 (“Pachynski”)
AMD1041	<i>AMD I/O Virtualization Technology (IOMMU) Specification</i> , 48882 Rev 2.62 (February 2015)

- AMD1042 Goossens et al. “Power/Performance Trade-Offs in Real-Time SDRAM Command Scheduling” IEEE Transactions on Computers, Vol. 56, No. 6, June 2016, pp. 1882-1895.
- AMD1043 U.S. Publication No. 2020/0233588 (“Li”)
- AMD1044 Voluntary Search Disclosure Declaration of Kenneth Darby
- AMD1045 U.S. Patent No. 6,000,019 (“Dykstal”)
- AMD1046 U.S. Patent No. 8,527,949 (“Pleis”)
- AMD1047 Toulson et al. “Fast and Effective Embedded Systems Design: Applying the ARM mbed. 2nd ed.” Newnes (2017)
- AMD1048 U.S. Publication No. 2017/0235671 (“Jung”)

LISTING OF CLAIMS

Claim 1	
[1pre]	A compute node for use in a data processing system, the compute node comprising:
[1a]	a communications link communicatively coupling the compute node to one or both of a host compute system and an adjacent compute node in an execution pipeline;
[1b]	a data store comprising records associated with a data processing operation;
[1c]	a programmable logic component to execute the data processing operation in cooperation with the data store;
[1d.i]	a node memory comprising data and instructions to support operation of the programmable logic component in connection with the data processing operation and to facilitate inter-node data communications with the adjacent node via the communications link,
[1d.ii]	wherein the node memory further comprises algorithm memory including resource-specific memory locations to support the data processing operation;
[1e]	a data mover component to facilitate intra-node data communications between the programmable logic component and the node memory; and
[1f]	a storage interface component to facilitate intra-node data communications between the programmable logic component and the data store;
[1g]	wherein the storage interface component utilizes a plurality of communications channels to transfer data between the programmable logic component and the data store; and
[1h]	wherein the data processing operation is associated with a computationally expensive algorithm and the compute node is deployed in a distributed processing compute environment to execute the computationally expensive algorithm.

Claim 2	
[2]	The compute node of claim 1 wherein the communications link employs a serial communications protocol enabling bi-directional data communication between the compute node and other compute nodes in the execution pipeline.
Claim 3	
[3]	The compute node of claim 2 wherein the compute node is communicatively coupled to the adjacent compute node in series via the communications link.
Claim 4	
[4]	The compute node of claim 1 wherein the data [store] comprise[s] an Electronically Erasable Programmable Read Only Memory (EEPROM) device.
Claim 5	
[5]	The compute node of claim 4 wherein the storage interface employs an Open NAND Flash Interface (ONFI) protocol.
Claim 6	
[6]	The compute node of claim 1 wherein the programmable logic component comprises a field programmable gate array (FPGA).
Claim 7	
[7]	The compute node of claim 1 wherein the compute node further comprises a processor to manage operation of the data mover component.
Claim 8	
[8]	The compute node of claim 7 wherein the data mover component comprises a device controller.

Claim 9	
[9]	The compute node of claim 1 wherein the communications link enables the compute node to receive instructions from the host compute system, the instructions to influence operation of the compute node.
Claim 10	
[10]	The compute node of claim 1 wherein the compute node is combined with a router module and integrated into an interface card communicatively coupled to the host compute system.
Claim 11	
[11]	The compute node of claim 1 wherein the compute node executes a respective operation associated with the computationally expensive algorithm using information in a computer-executable code associated with the computationally expensive algorithm and results of a respective preceding operation, and passes respective additional results of the respective operation to a next compute node in the execution pipeline.
Claim 12	
[12]	The compute node of claim 11 wherein the computer-executable code associated with the computationally expensive algorithm is incorporated in a bit stream.

CONVENTIONS OF THE PETITION

- All emphases are added unless noted otherwise;
- Bold-italic emphases correlate to claim language;
- Quotations are generally from exhibits, not claim language; and
- The phrase “as discussed” and equivalent phrases incorporate fully the analysis of the cross-cited portion of the Petition.

Advanced Micro Devices, Inc. (“Petitioner”) petitions for IPR of claims 1–12 (“Challenged Claims”) of U.S. Patent No. 12,271,636 (“the ’636 patent”) based on compelling evidence of obviousness presented throughout this Petition. AMD1003, ¶29.

I. REQUIREMENTS FOR IPR

A. Grounds for Standing

Petitioner certifies that the ’636 patent is available for IPR, and that Petitioner is not barred or estopped from requesting IPR. *See* §VII.B.

B. Challenge and Relief Requested

Petitioner requests IPR based on the grounds noted below.

ID	Claims	Prior Art Basis Under §103
1A	1–3, 6–12	Dalal-Ng
1B	4, 5	Dalal-Ng-Raghava
2A	1–4, 6–9, 11–12	Heil-Jun
2B	5, 10	Heil-Jun-Raghava

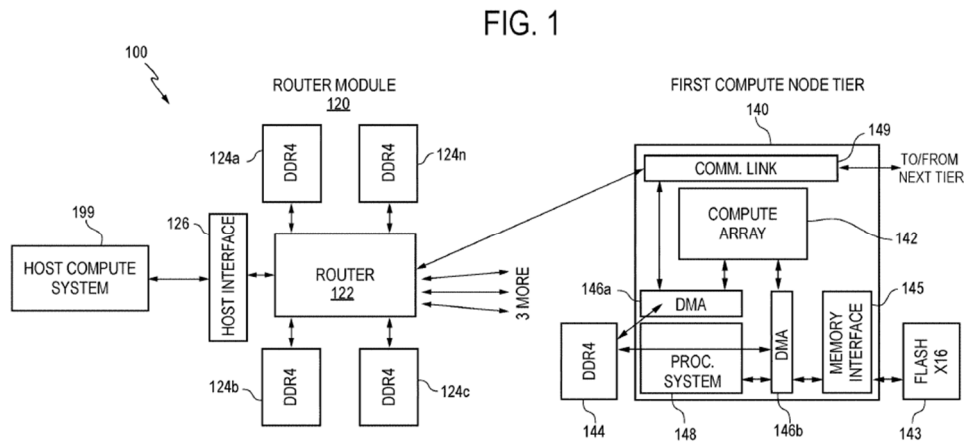
Petitioner does not concede that the ’636 patent is entitled to the benefit of its priority claim. Petitioner has used February 6, 2019, as the earliest possible priority date (“Critical Date”) of the Challenged Claims. Each reference applied in the Grounds listed above qualifies as prior art under 35 U.S.C. §§ 102 or 103.

Reference	Filing Date	Publication Date
Dalal- (AMD1004)	2018-09-12	2019-04-11
Ng- (AMD1005)	2017-10-17	2019-04-18
Raghava- (AMD1006)	2018-12-06	2019-04-11
Heil- (AMD1010)	2015-05-20	2016-10-20
Jun- (AMD1011)	—	2015-06

II. THE '636 PATENT

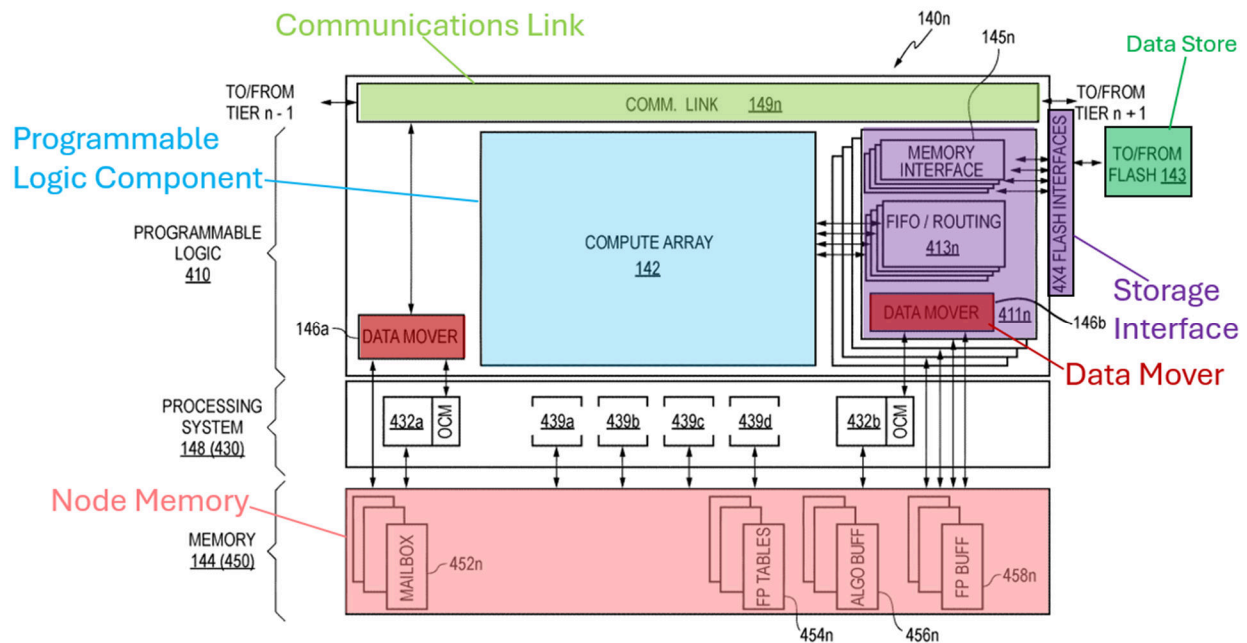
A. Overview

The '636 patent describes a “data processing system” that includes multiple nodes performing operations in an execution pipeline. AMD1001, Title, 7:60–8:11, FIG. 1. AMD1003, ¶¶22–24.



AMD1001, FIG. 1.

A node (e.g., 140n shown in Figure 4 below) in the pipeline includes programmable logic, data store, storage interface, data mover, node memory, and communications link. AMD1001, 10:41–11:45, 55:8–56:41; FIGS. 1, 4; AMD1003, ¶23.



AMD1001, FIG. 4.¹

B. Prosecution History

During prosecution associated with the '636 patent, among other rejections, the Examiner rejected all claims as being anticipated by Whitehead (AMD1009). AMD1002, pp.46–71. In response, Patent Owner amended claim 1 to include Element_[1d.ii] and Element_[1h] while arguing that Whitehead lacked a similar “operational context” and failed to disclose “computationally expensive

¹ Annotations to the figures throughout this petition are shown in color.

algorithms.” *Id.* Patent Owner cited “Big Data or artificial intelligence applications” as examples of “computationally expensive algorithms.” *Id.*; AMD1003, ¶25.

III. LEVEL OF ORDINARY SKILL

A person of ordinary skill in the art (“POSITA”) relating to the subject matter of the ’636 patent as of the Critical Date would have had a bachelor’s degree in Computer Engineering, Electrical Engineering, Computer Science, or a comparable field and at least two years of experience related to computer architecture. AMD1003, ¶¶27–28. Additional experience could substitute for a formal degree or formal training (and *vice versa*). *Id.*

IV. CLAIM CONSTRUCTION

The Challenged Claims are obvious under any reasonable interpretation advanced pursuant to the *Phillips* standard. Although the Board need not resolve any express constructions to adjudicate this Petition, Petitioner nevertheless identifies the claim term “programmable logic component” as a “logic component programmable by reconfiguring its internal hardware connections, such as an FPGA.” AMD1003, ¶26. This interpretation is supported by the intrinsic record and consistent with the ordinary meaning. AMD1001, 15:55–56, 39:60–64, 20:43–50.

V. THE CHALLENGED CLAIMS ARE UNPATENTABLE

A. GROUND 1A: Dalal-Ng Renders Claims 1–3, 6–12 Obvious

1. Dalal

Dalal² describes a system including servers connected in a distributed computing environment. AMD1004, [0026]–[0048], FIGS. 1–2C; AMD1003, ¶30. Figure 1 (below) shows a server in Dalal’s system that includes hardware accelerator modules (hwa modules) that include computing elements that process data. *Id.* Each server includes “server switches” coupled to the hwa modules and a host processor. *Id.*

² General descriptions provided for this and other references and combinations thereof are incorporated into each subsection and mapping of the claims that includes citations to these references.

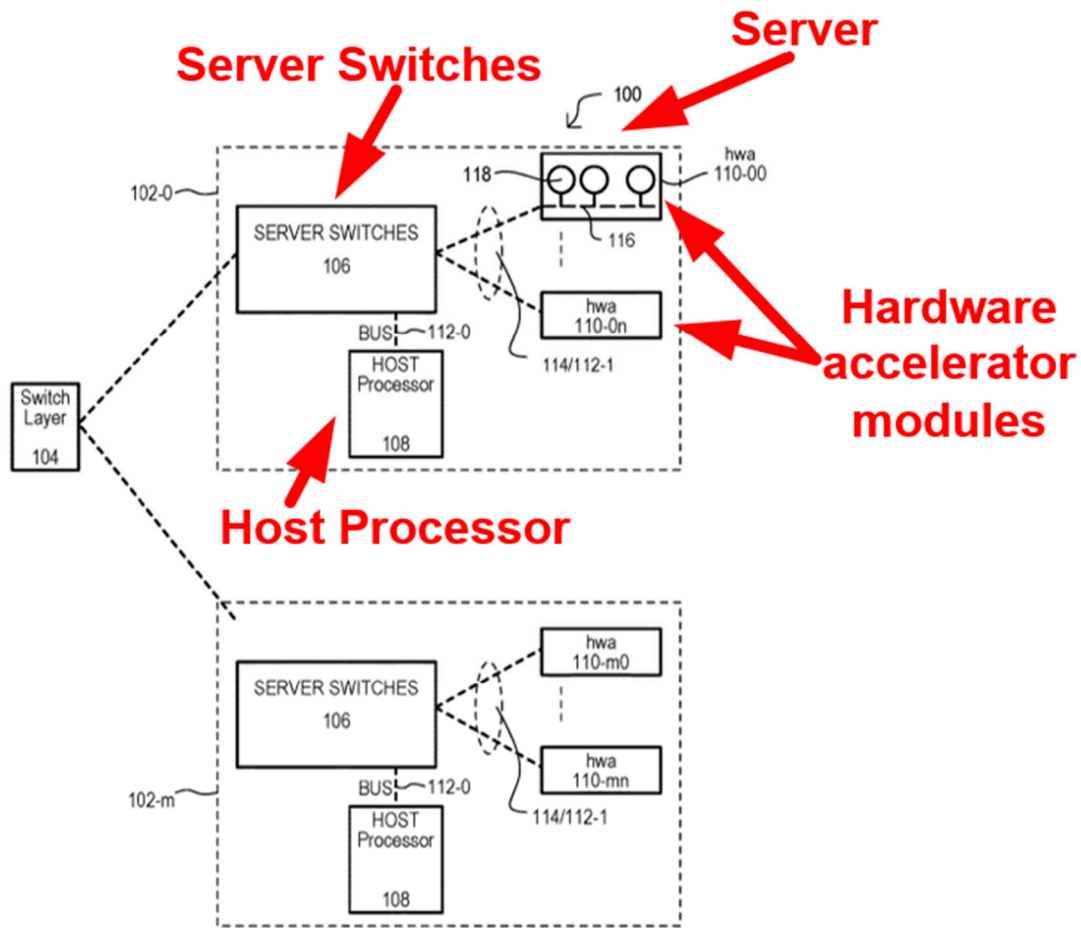
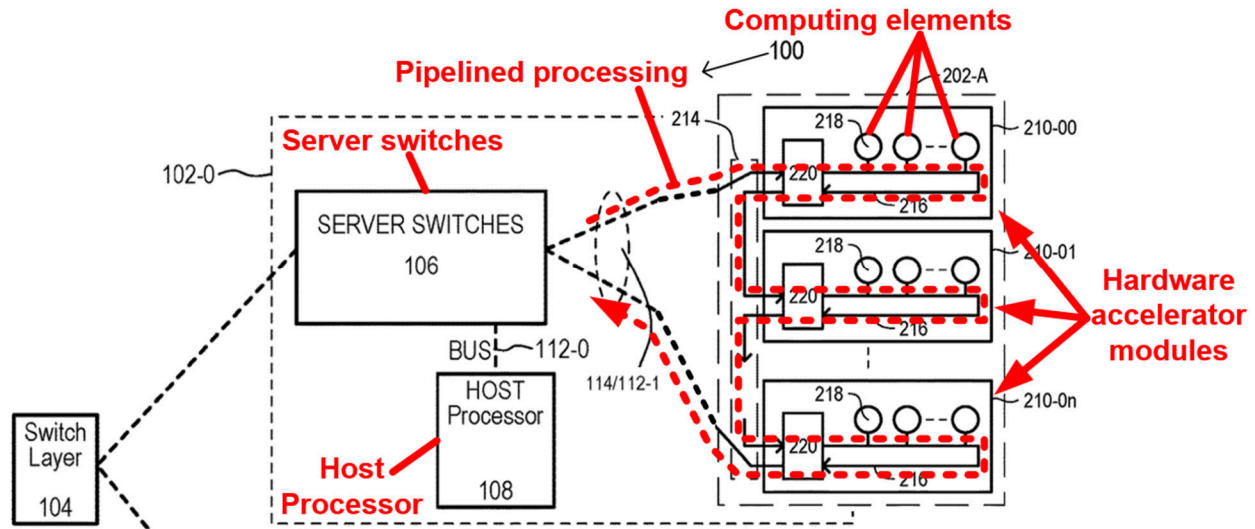


FIG. 1

AMD1004, FIG. 1.

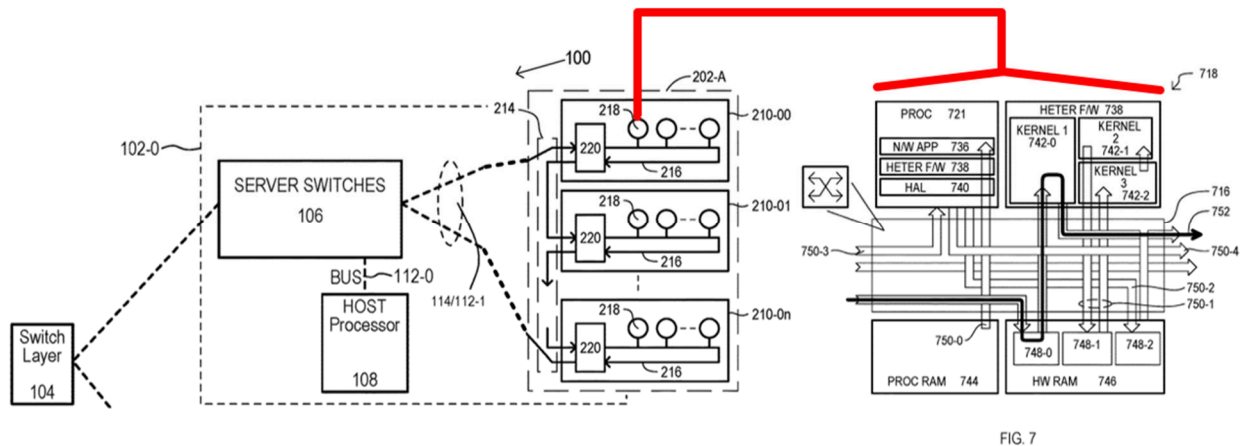
Dalal's "FIG. 2A to 2C show various systems, which...can be particular versions of system 100 of FIG. 1." AMD1004, [0039]. The below drawing is a combined figure that integrates the system of Figure 1 with the "particular version" of Figure 2B. AMD1003, ¶31. In the combined figure, the hwa modules 210 in box 202-A from Figure 2B replace the hwa modules 110 shown in Dalal's original Figure 1. In the combined figure (below), each server includes hwases that are serially

coupled to one another so that the modules process data sequentially. AMD1004, [0035]–[0048], FIGS. 2A–2C; AMD1003, ¶32.



AMD1004, FIGS. 1, 2B Combination Figure.

The computing elements located in Dalal's hwa modules include components used for processing data according to a data processing operation (e.g., machine learning), including processing circuits (e.g., programmable logic), a first memory, and a second memory. AMD1004, [0075]–[0088], FIG. 7; AMD1003, ¶33.



AMD1004, FIGS. 1, 2B Combination Figure; AMD1004, FIG. 7.

2. Ng

Ng describes a host computer system that communicates with a hardware accelerator to perform complex data processing operations. AMD1005, [0025]–[0042], FIGS. 1–4; AMD1003, ¶34. The hardware accelerator includes a programmable integrated circuit (IC) (*e.g.*, a field programmable gate array (FPGA)) that communicates with memory and a host computer system. AMD1005, [0032]–[0039], FIGS. 2–4.

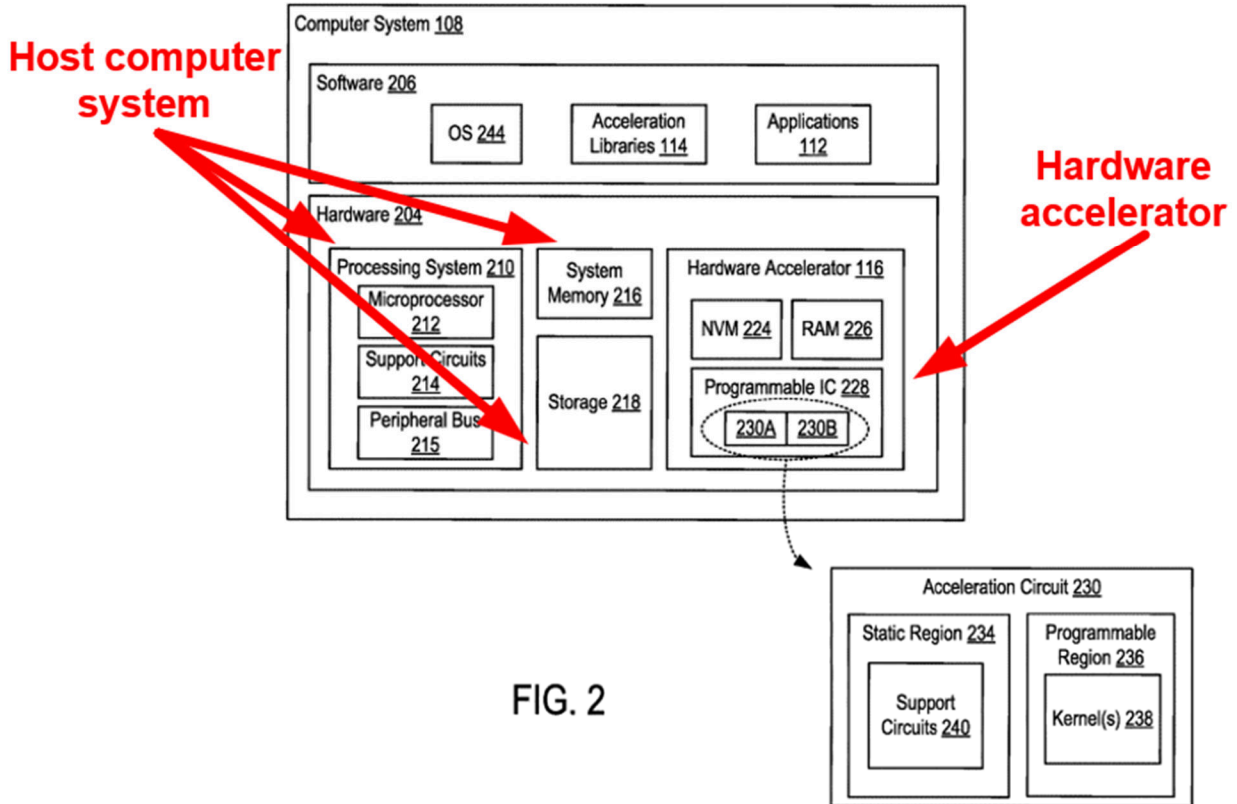


FIG. 2

AMD1005, FIG. 2.

The hardware accelerator performs complex data processing operations to execute multiple layers of a neural network. AMD1005, [0007]–[0008], [0021]–[0046], FIGS. 1–4; AMD1003, ¶35.

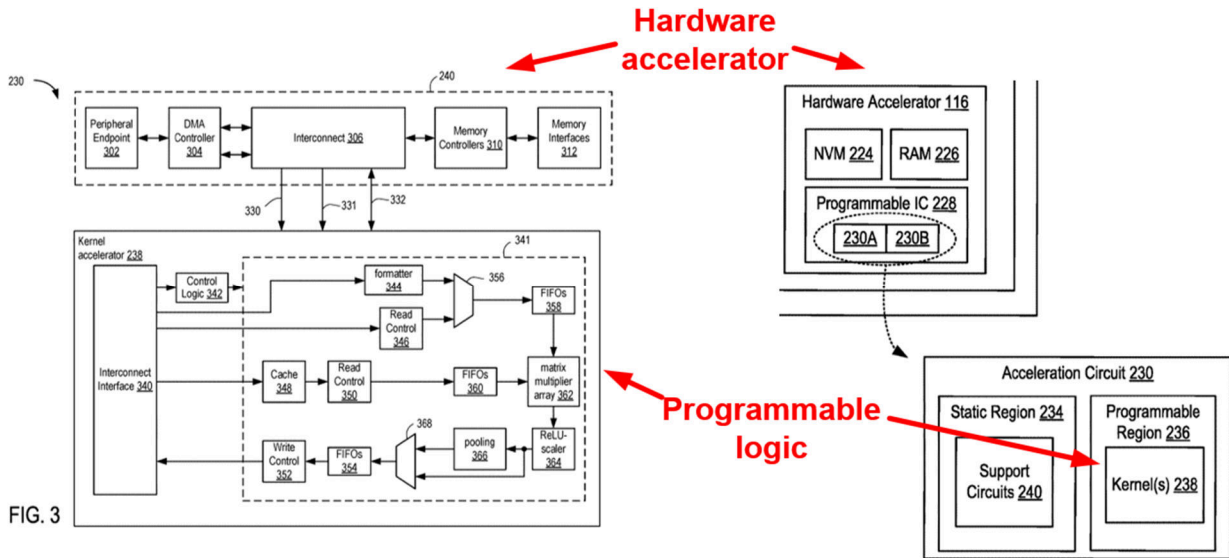


FIG. 3

AMD1005, FIGS. 2, 3.

3. Motivation to Combine

Dalal and Ng are analogous art to the '636 patent. AMD1003, ¶36. The '636 patent is directed to “a distributed hardware architecture” that “facilitate[s] high throughput data processing and analytics solutions” such as “machine learning and artificial intelligence.” AMD1001, Abstract, 1:38–56, 2:4–17, 48:36–64, 49:59–50:10. Dalal “network architecture” includes “hardware accelerator (hwa) modules” arranged to perform complex operations from a “machine learning library, such as TensorFlow,” which falls within the same field of endeavor as the '636 patent. AMD1004, [0026], [0032]–[0036], [0040]–[0050], [0078], FIGS. 1, 2B–2C; *supra*, §V.A.1. Ng involves a host computer system that provides tailored instructions to a single hardware accelerator for executing every layer of a neural network and therefore (1) is within the same field of endeavor as the '636 patent and (2) describes

solutions relevant to one or more problems addressed by the '636 patent. AMD1005, [0021]–[0022], [0024], [0039]–[0042], [0044]–[0046], FIGS. 2–4; AMD1003, ¶36.

A POSITA would have been motivated to modify Dalal by incorporating Ng's:

- (1) host computer system;
- (2) programmable neural network accelerator;
- (3) memory interfaces; and
- (4) memory storing records such as matrices and work requests.

AMD1005, [0025]–[0051]; AMD1003, ¶37. An example illustration of how a POSITA would have combined Dalal and Ng is provided below. AMD1003, ¶37. In the Combination, Dalal's host processors are implemented with the host computer system, and each one of Dalal's hwa modules are implemented with Ng's hardware accelerator. AMD1003, ¶37. Doing so preserves Dalal's pipelined architecture and supplements Dalal's servers with aspects of the host-accelerator cooperation described in Ng. AMD1003, ¶37.

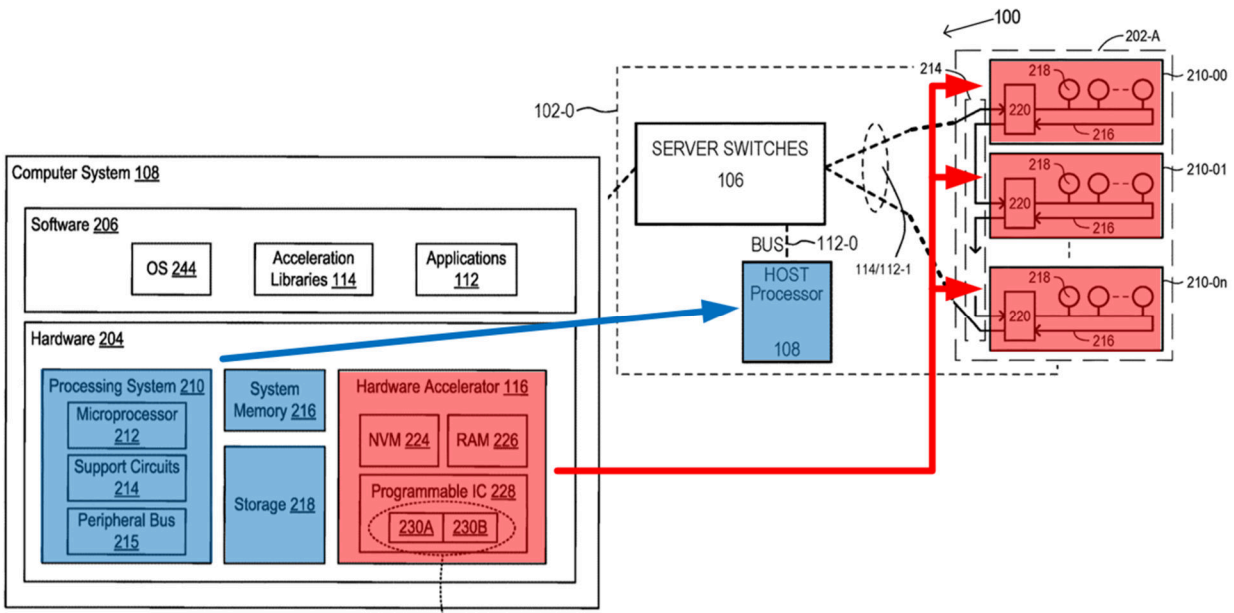


Illustration of a Portion of the Dalal-Ng Combination.

The following reasons support the Dalal-Ng Combination (DNC).
AMD1003, ¶38.

First, a POSITA would have implemented Dalal's host processor with Ng's host computer system that coordinates with a hardware accelerator to manage complex machine learning operations. AMD1003, ¶39. For example, Dalal's hardware accelerators use "pipelined processing" to sequentially process packet data. AMD1004, [0035]–[0036], [0042]; AMD1003, ¶39. Ng describes an effective division of labor between the host and hardware accelerator, whereby the accelerator performs computations for implementing each layer of a neural network using a hardware accelerator and the host "need only provide the control parameters to the neural network accelerator." AMD1005, [0007], [0023]; AMD1003, ¶39. Dalal

acknowledges that “processing circuits” of the hwa modules “consume less power than a host processor,” and Ng recognizes that concentrating “as much processing as possible in the neural network accelerator” will “improve system performance” by reducing an amount of host-accelerator communication. AMD1004, [0063], [0077]; AMD1005, [0023]; AMD1003, ¶39.

Second, a POSITA would have understood that implementing Dalal’s system with Ng’s design tools, such as a user interface for programming the hardware accelerators, would have allowed a developer to customize Dalal’s pipelined processing to achieve specialized goals. AMD1005, [0027]; AMD1003, ¶40. Because hardware accelerators include programmable logic, these accelerators would have been programmable to perform custom operations more efficiently than a general processor (*e.g.*, a CPU or GPU) could have performed these tasks, while consuming less power in the process. AMD1005, [0006]; AMD1004, [0063], [0077]; AMD1003, ¶40.

Third, configuring programmable logic of Dalal’s hwa modules to perform computations such as “matrix multiplication” and “convolution,” as described by Ng, would have provided an improved method for executing “machine learning applications,” which Dalal cites as an example operation performed by the hwa modules. AMD1005, [0021], [0039], [0041]–[0042]; AMD1004, [0078]; AMD1003, ¶41. Configuring Dalal’s hwa modules to perform convolution and

matrix multiplication would have positioned these modules to efficiently execute layers of a neural network and therefore implement machine learning applications. *Id.*

Fourth, memory interface circuits manage communication between programmable logic and memory. AMD1003, ¶42. Implementing Dalal's hwa modules with a memory interface circuit that operates according to a protocol of a coupled memory would have beneficially allowed the modules to ensure that outgoing data is arranged to be received by the memory and received data is identified and processed quickly. *Id.*

Fifth, it would have been obvious for a POSITA to implement Dalal's hardware accelerators with a cooperative host, as described by Ng, because doing so represents one of a finite number of predictable solutions for governing a data processing operation performed by accelerators. AMD1003, ¶43. A POSITA would have appreciated, for example, that "programmable" logic would have been programmed by a developer, and it would have been predictable to do so via a host computer system. *Id.*

Implementing Dalal's hwa modules with a cooperative host and interfaces for communicating with memory merely incorporates well known features that yield predictable results. AMD1003, ¶44. Supplementing Dalal's hwa modules in this

way would have enhanced the system's ability to handle multi-step operations in a systematic manner. *Id.*

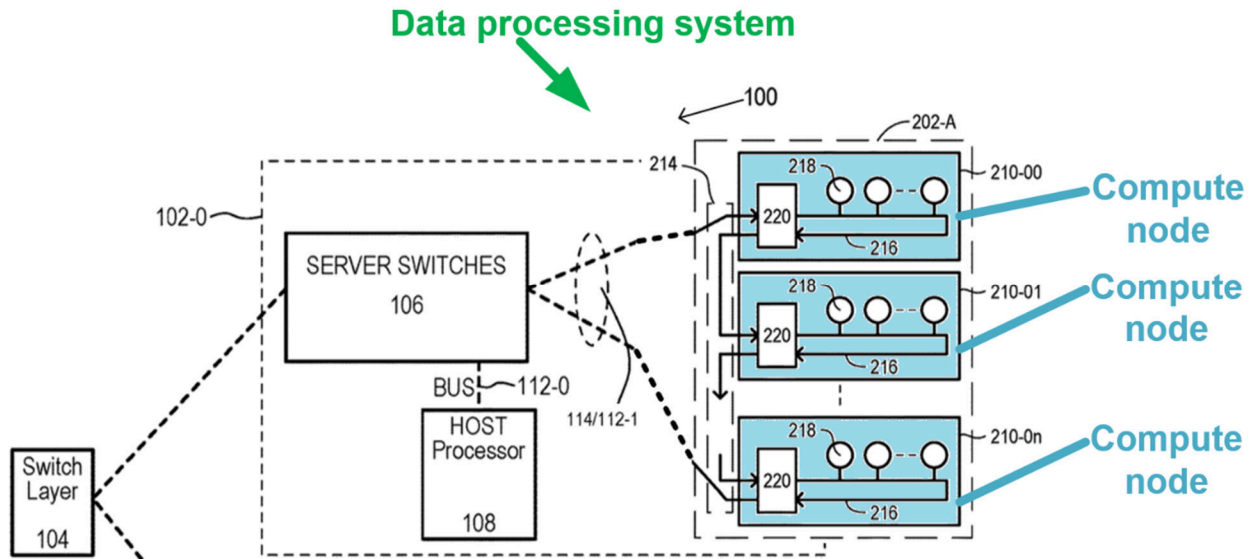
Furthermore, combining the teachings of Dalal and Ng would have been obvious and a POSITA would have reasonably expected success, because Dalal's hwa modules process data sequentially just as Ng's hardware accelerator executes each layer of a neural network in order. AMD1004, [0042]–[0047]; AMD1005, [0047]; AMD1003, ¶45. A POSITA would have reasonably expected that using a pipeline of hwa modules to execute a similar multilayer neural network, as described by Ng, would have resulted in a predictable and foreseeable chance of success because both approaches merely involve using the results of one layer as inputs to the next layer. *Id.*

4. Claim Element Analysis

Element_[1pre]

Dalal's **system** includes one or more servers, each of which includes “one or more hwa modules” (**compute nodes**). AMD1004, [0031]–[0047], [0075], [0089]–[0099], [0123]–[0125], [0170]–[0172], FIGS. 1–2C, 8, 12, 18; AMD1003, ¶46. Each hwa module includes one or more “computing elements” and therefore represents a **compute node** at least because it performs computation operations on received data. AMD1004, [0026], [0033], [0075], [0089]. The hardware accelerator modules (**compute nodes**) are connected to “server switches” and “host processors”

via a “bus” and are used to “execute operations on received *data*,” indicating that the compute nodes are *for use in a data processing system*. AMD1004, [0031]–[0038], [0040]–[0047], FIGS. 1–2C.



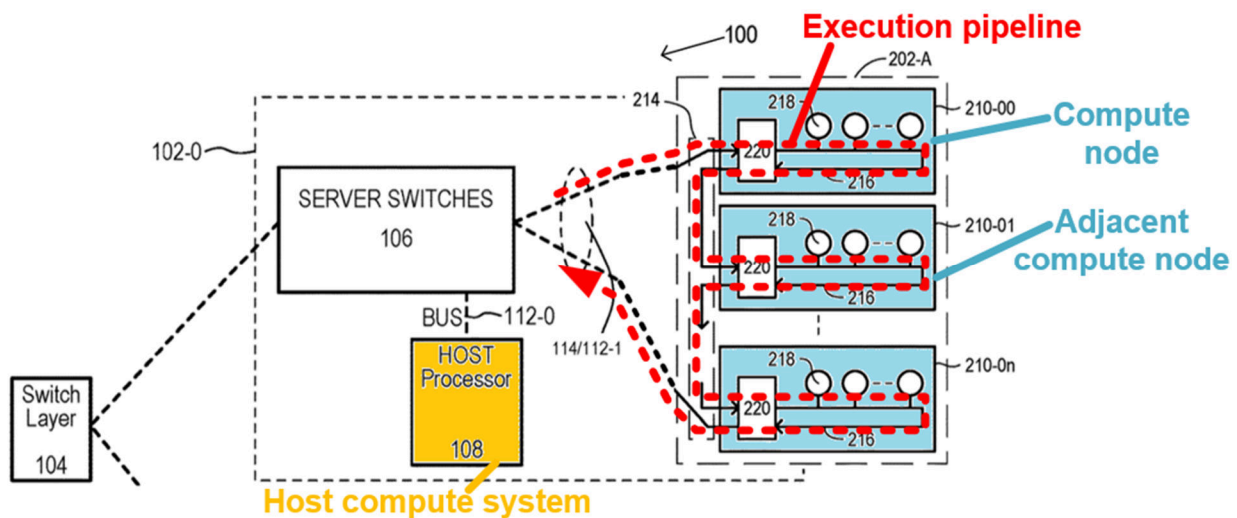
AMD1004, FIGS. 1, 2B Combination Figure³.

Element [1a]

Each server includes “one or more *host* processors” (*host compute system*) configured to process data. AMD1004, [0031]–[0038], [0040]–[0042], [0046]–[0047], [0063], [0069]–[0070], [0073], [0077], [0099], [0101]–[0103], [0123]–

³ Recall from §V.A.1 that Dalal explains that Figure 2B embodies a version of Figure 1. AMD1004, [0039]. Thus, Figures 1 and 2B would have been combined in the manner illustrated here consistent with Dalal’s teachings. AMD1003, ¶46.

[0125], [0170]–[0172], FIGS. 1–2C, 5–6, 12, 18, cl.1; AMD1003, ¶47; *supra*, Element_[1pre]. As shown in Dalal’s combination figure below, each server includes server switches 106 coupled to a host processor 108 (*host compute system*) and hwa modules 210 (*compute nodes*) via buses 112. AMD1004, [0031]–[0038], [0040], [0042], [0046], [0048], FIGS. 1–2C; AMD1003, ¶48.

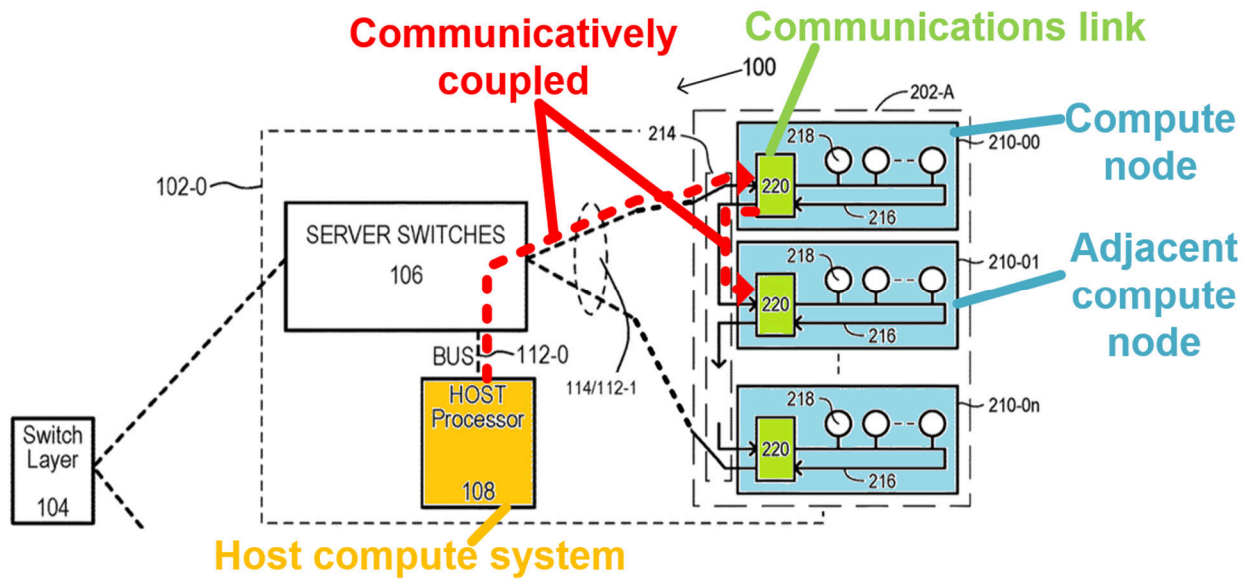


AMD1004, FIGS. 1, 2B Combination Figure.

Within the server, “[r]eceived packet data can be processed by...hwa module 210-00[*compute node*]...forwarded to a next hwa module 210-01[*adjacent compute node*], where it can be further processed...[s]uch processing can continue through multiple hwa modules (210-00 to -0n) of a same server,” demonstrating that the *compute node* is connected to an *adjacent compute node* in an *execution pipeline*. AMD1004, [0042], [0045]–[0047], [0035], [0036], FIGS. 1, 2B–2C; AMD1003, ¶48. Dalal explains that “*pipelined* processing can be scaled up within a server by

use of additional hwa modules,” confirming that hwa module 210-00 (*compute node*) and hwa module 210-01 (*adjacent compute node*) are connected in an *execution pipeline*. AMD1004, [0042], [0035] (“*pipelined* processing”), [0045] (“processing *pipeline*”), [0049].

Each of Dalal’s hwa modules (*compute nodes*) includes a switching circuit 220 (*communications link*) that receives and outputs data. AMD1004, [0040]–[0048], FIGS. 2A–2C; AMD1003, ¶49. The switching circuit is a *communications link* at least because it “can be any suitable structure for translating received packet data.” AMD1004, [0040]. This *communications link* includes the switching circuit and any buses, fabrics, links, and interfaces connected thereto (*e.g.*, data path 214). AMD1004, [0043]. The switching circuit (*communications link*) of hwa module 210-00 (*compute node*) is *communicatively coupled* to (1) the host processor (*host compute system*) and (2) the hwa module “210-01” (*adjacent compute node*). AMD1004, [0032], [0035], [0040], [0042], [0048].



AMD1004, FIGS. 1, 2B Combination Figure.

Element [1b]

Each hwa module includes “computing elements” that execute a *data processing operation*. AMD1004, [0033] (“computing elements...execute *operations* on received *data*”), [0035]–[0046], [0075]–[0087], [0026]–[0029], FIG. 7, FIGS. 1–2C, 7; *supra*, Element_[1pre]; AMD1003, ¶50. Dalal’s Figure 7 (reproduced below) depicts an example block diagram of a single computing/computation element 718 within a hardware acceleration module. AMD1004, [0075]; *supra*, §V.A.3. “A computation element...can include one or more processing circuits 721” that “run a program 736 for a networked application,” thereby confirming that the hwa module executes a *data processing operation*. AMD1004, [0076].

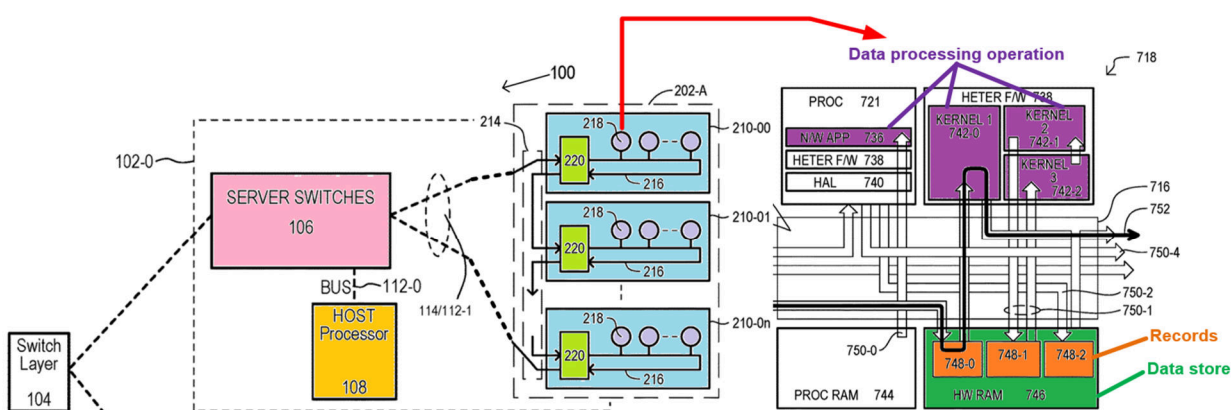


FIG. 7

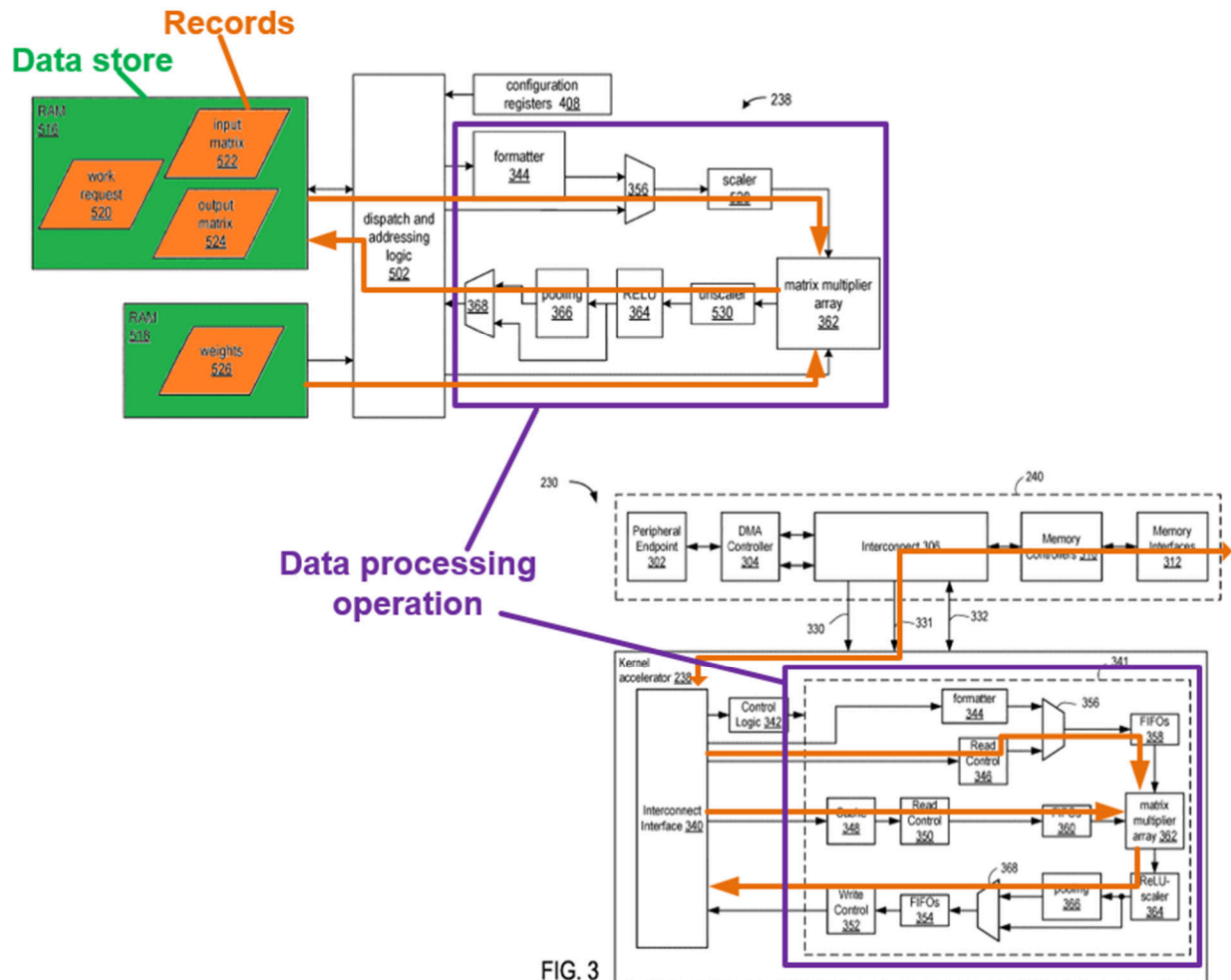
AMD1004, FIGS. 1, 2B Combination Figure; AMD1004, FIG. 7.

To run the program, Dalal's processing circuits access *records* stored in predetermined memory spaces of a second memory 746 (*data store*) of each computing element. AMD1004, [0081], cls.1-2, [0075], [0080]-[0082], [0085], [0094], cls.1-2. Each memory space provides a functional unit of memory for storing packet data (*record*). *Id.*; AMD1003, ¶¶51-55, *citing corroborating*, AMD1030, 4 (a record represents "[a] set of related data items treated as a unit"). Collectively, the memory spaces and the second memory are a *data store* comprising *records* associated with a *data processing operation*. *Id.*

Moreover, DNC incorporates Ng’s RAM (*data store*) comprising *records* that include matrices and work requests. AMD1005, [0041], [0044], [0007]–[0008], [0024], [0032], [0034], [0036]–[0038], FIGS. 3–5; *see* §V.A.3; AMD1003, ¶56. As depicted below in Ng’s FIGS. 3 and 5, a programmable IC uses these *records* to execute a *data processing operation*, such as performing matrix multiplication for

each layer of a neural network using input and weight matrices stored in the RAM.

AMD1005, [0041], [0052], FIGS. 3–5.



AMD1005, FIGS. 3, 5.

In DNC, Dalal's hwa modules would have implemented Ng's *data processing operation* through storing *records* (e.g., matrices, work requests) in the memory spaces of Dalal's second memory. AMD1004, [0080]–[0081], FIG. 2B; AMD1005, [0024], [0047], [0094]; AMD1003, ¶57; *supra*, §V.A.3. In DNC, Ng's records would have been transmitted as packet data in Dalal. AMD1003, ¶¶57–59.

*Element*_[1c]

The computing elements in Dalal’s hwa module each include processing circuits 721, 738 that collectively represent a *programmable logic component*. AMD1004, [0075]–[0079], FIG. 7. Example processing circuits “include an ARM type processor” that is “integrated with other circuits” such as “*programmable logic*.” AMD1004, [0077], [0085] (“processing circuits...can be...ARM processors embedded with *programmable [logic]*”); AMD1003, ¶60.

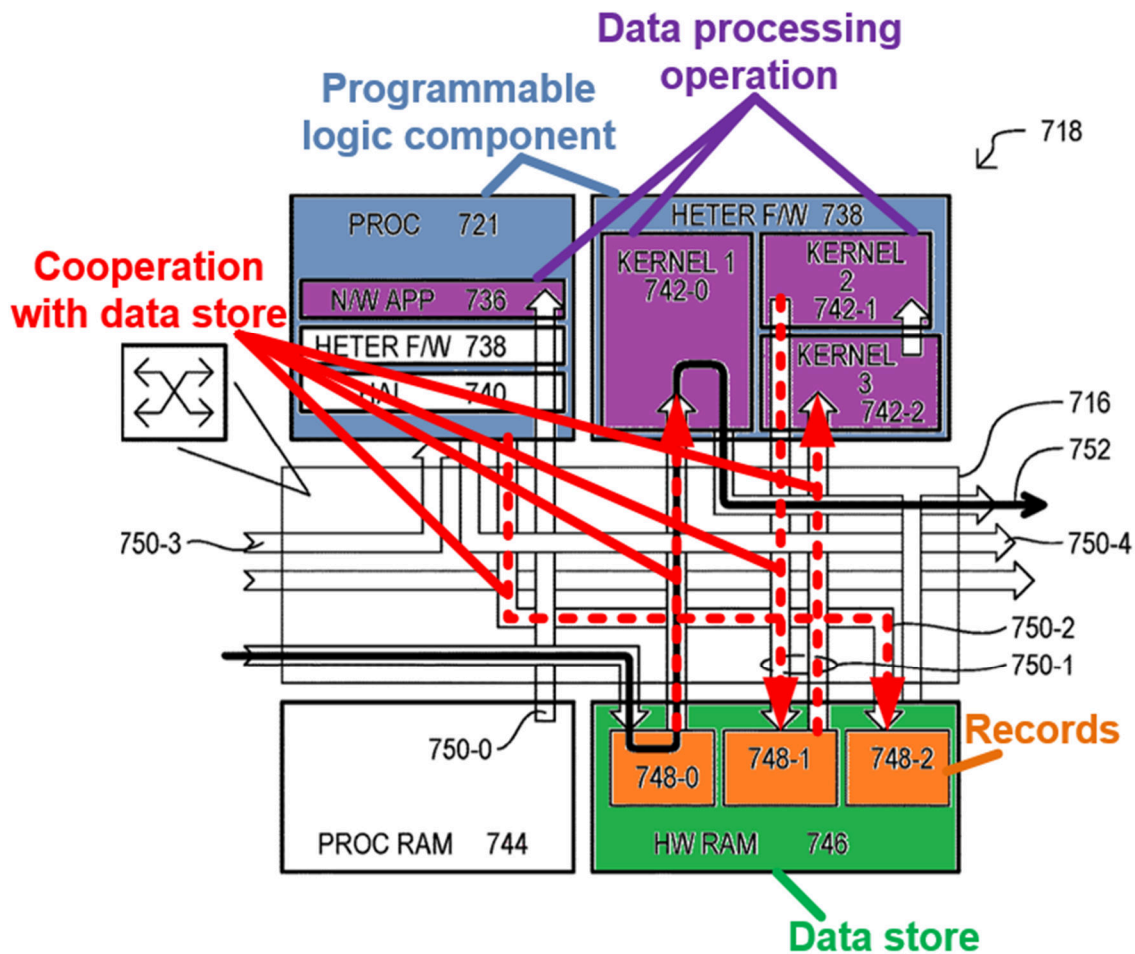


FIG. 7

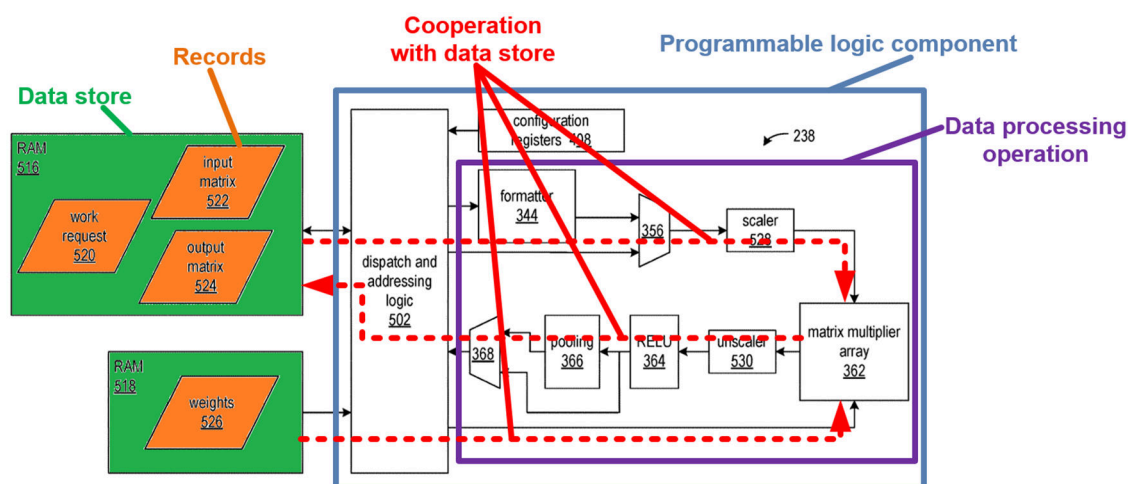
AMD1004, FIG. 7.

Internal hardware connections of Dalal's processing circuits (*programmable logic component*) are configurable (or reconfigurable) to "run a program for a networked application." AMD1004, [0076]–[0079], FIG. 7; AMD1003, ¶61. This program is "written on a framework" that "enable[s] various execution processes...to perform computation actions." AMD1004, [0078], FIG. 7. Execution processes retrieve *records* ("packet data") from the *data store* and therefore *execute* the *data processing operation* in *cooperation* with the *data store*. AMD1004, [0078], [0081]; *supra*, Element_[1b]; AMD1003, ¶62.

DNC also renders Element_[1c] obvious via Ng's supplemental teaching of a hardware accelerator including a *programmable logic component*. AMD1005, [0025]–[0039], [0042]–[0044], [0047]–[0051], FIGS. 1–5; AMD1003, ¶63. For example, Ng's hardware accelerator includes "*programmable* integrated circuits (ICs)" that include "field *programmable* gate arrays (FPGAs)." AMD1005, [0026], [0032]–[0035], FIGS. 2–3.

Ng explains that internal hardware connections of this *programmable* IC are configured (or reconfigured) to perform a specialized task, which would have motivated a POSITA to implement Dalal's hwa modules with Ng's *programmable logic component*. AMD1005, [0027] ("programming the hardware accelerator" using "configuration bitstreams for FPGAs"), [0034] ("*programmable* IC"), [0035] ("reconfigurations of the *programmable* region"), [0080] ("*programmable*

logic...includes *logic* cells...configured to implement general logic functions”); *supra* Element_[1b], *supra* §V.A.3; AMD1003, ¶64. As depicted below in Ng’s Figure 5, this programmable IC performs a *data processing operation* by retrieving *records* (e.g., “input” and “weight” matrices) from a *data store* for processing and delivering processed *records* (e.g., “output” matrices) to the *data store*. AMD1005, [0041], [0050], [0053], FIGS. 3–5; AMD1003, ¶64.



AMD1005, FIG. 5.

A POSITA would have understood that a *programmable logic component*, such as Dalal-Ng’s FPGA, would have included internal hardware connections that are reprogrammable by configuration data (e.g., bitstreams). AMD1003, ¶65 (*citing corroborating*, AMD1027, 31, AMD1006, [0023]–[0030]).

Element [1d.i]

As described above in Elements_[1b]–[1c], DNC’s *programmable logic component* runs programs to perform a *data processing operation*. AMD1004,

[0075]–[0082], FIG. 7; AMD1005, [0025]–[0051]. These programs are “loaded into first *memory*” 744 of each computing element in an hwa module, and the first memories collectively represent a *node memory*. AMD1004, [0079], [0075], [0082], FIG. 7. As depicted below in Dalal’s Figure 7, the programs are executed “by processing circuits” to perform execution processes and computation actions, and therefore support *operation of the programmable logic component in connection with the data processing operation*. AMD1004, [0076]–[0079], FIG. 7; *supra*, Element_[1b]; AMD1003, ¶66.

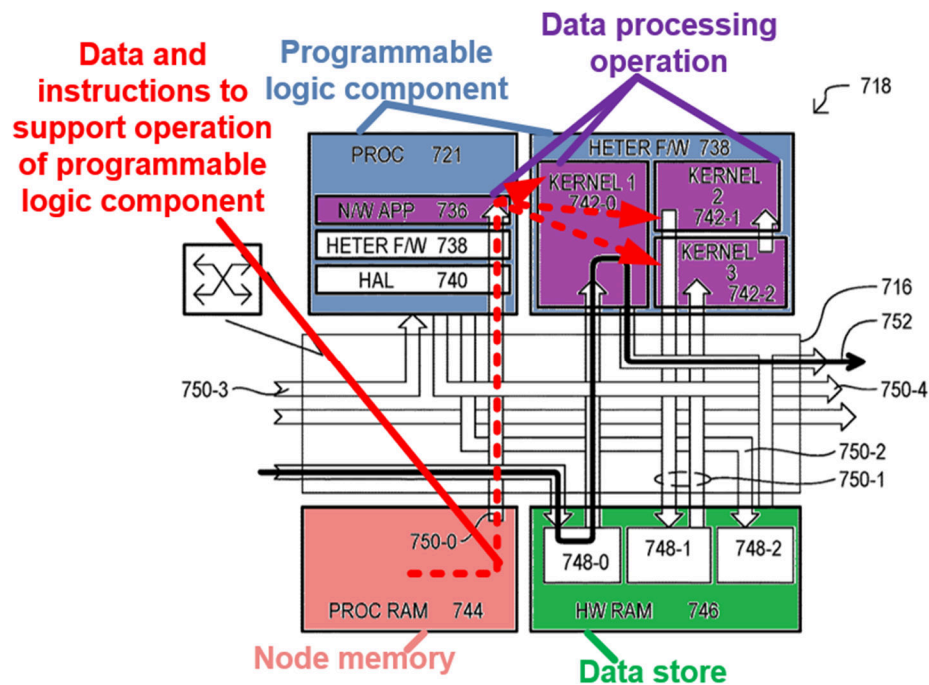


FIG. 7

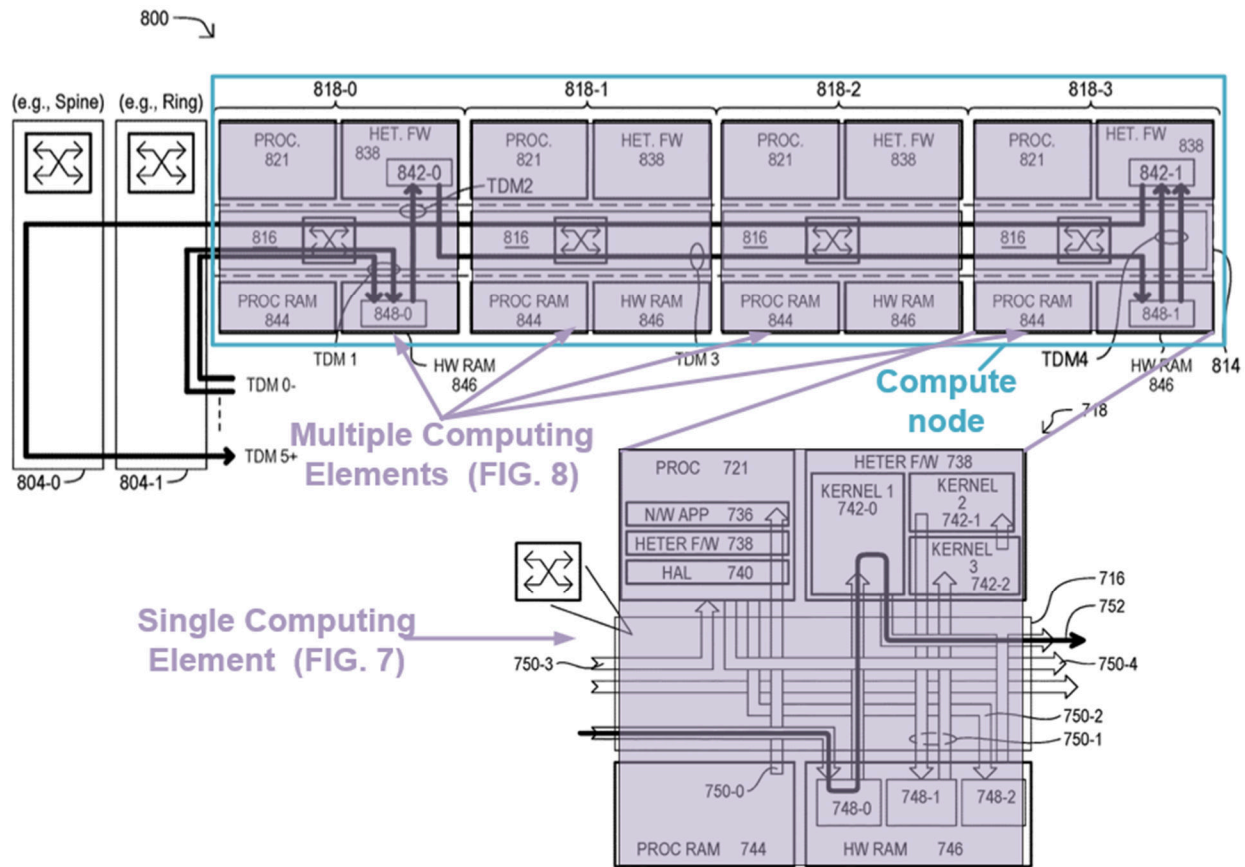
AMD1004, FIG. 7.

Dalal’s first memory 744 (*node memory*) “store[s] *instructions* for execution by the processing circuits” (*supporting operation* of the *programmable logic*

component). AMD1004, cl.11. Furthermore, a *data* transfer fabric 716 provides “*data* transfer paths between components” including connections between the processing circuits and first memory. AMD1004, [0081]–[0083], FIG. 7. Path 750-0 in the fabric transfers *data* and *instructions* from the first memory (*node memory*) to the processing circuits (*programmable logic component*). AMD1004, [0079]; AMD1003, ¶67. Recall also that “[f]irst memory...[is] accessed by processing circuits,” “programs can be loaded into first memory...for execution by processing circuits,” and a “framework can enable various execution processes...to perform computation actions called by a program.” AMD1004, [0078], [0079]. At least because the data and instructions stored in the first memory are used by the processing circuits (*programmable logic component*) to execute operations, DNC’s system renders obvious *a node memory comprising data and instructions to support operation of the programmable logic component in connection with the data processing operation*. AMD1003, ¶68.

The analysis will now turn to the *inter-node communications* aspect of Element_[1d.i]. As described above, Dalal’s Figure 7 depicts a single computing element located within an hwa module that includes one or more computing elements. AMD1004, [0075], [0033]; *supra*, Element_[1pre], Element_[1b], FIG. 7; AMD1003, ¶69. Figure 8 then depicts an hwa module having multiple computing

elements 818-0 to -3 that each “take the form” of the single computing element in Figure 7. AMD1004, [0089], [0090], FIGS. 7–8.



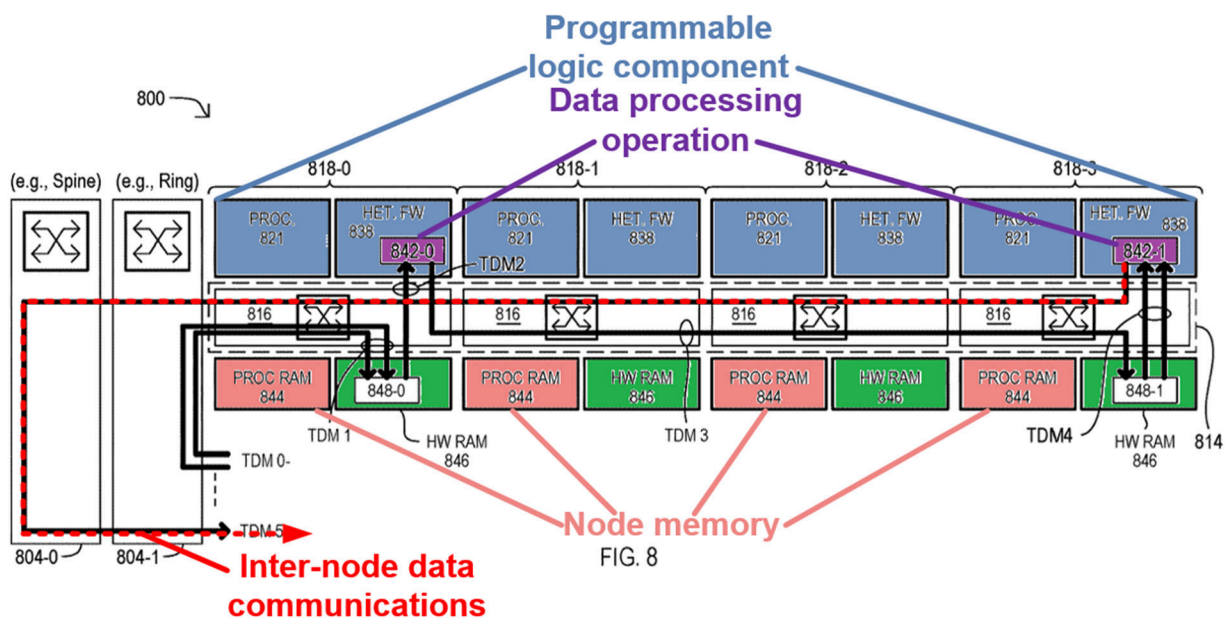
AMD1004, FIGS. 7–8.

Figure 8 depicts the understanding a POSITA would have had of a computing element 718, as shown in Figure 7, in connection with other similar computing elements, which is consistent with the connection of computing elements shown in Dalal’s Figure 2B. See §V.A.4, Element_[1b]; AMD1003, ¶70. In particular, Dalal explains that the computing elements (818-0 to -3) of the hardware acceleration module are connected to other parts of a system over a first connection layer 804-0

that couples hwa modules of the same server. AMD1004, [0091]. The first connection layer 804-0 “include[s] a bus...that can connect hwa modules to one another” and “can include inter-module connections...including direct inter-module connections.” AMD1004, [0092]. This supports other parts of Dalal’s disclosure (e.g., Figure 2B) confirming that *inter-node data communications* occur between the *compute nodes* and an *adjacent compute node* in the same server. AMD1004, [0042], FIG. 2B; *supra*, Elements_[1pre]–[1a]; AMD1003, ¶70. These *inter-node data communications* occur via the switching circuit (*communications link*) in each hwa module (*compute node*). AMD1004, [0040]–[0045], FIGS. 1, 2B; *supra*, Element_[1c].

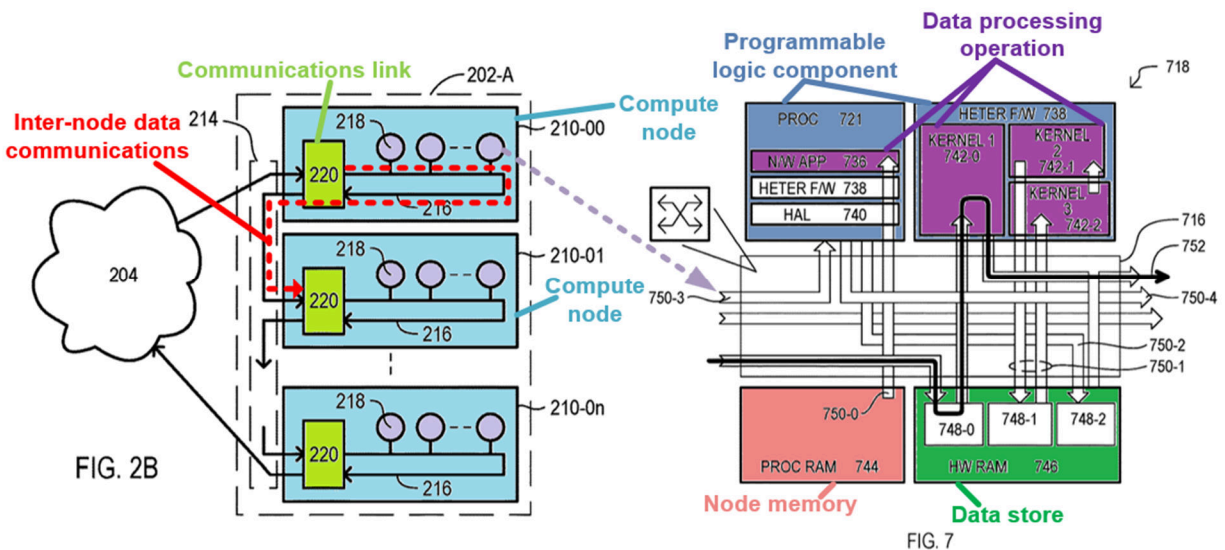
According to Dalal, a result of execution process 842-1 is sent via switching layer 804-0 to another location (e.g., another appliance with computing elements connected by an inter-module fabric). Execution process 842-1 “operate[s] according to a same network application” corresponding to the “programs” of Dalal’s Figure 7, which are run by the *programmable logic component* and stored in the *node memory*. AMD1004, [0095], [0076]–[0088], FIGS. 7–8; AMD1003, ¶71. Execution process 842-1, like the execution processes 742-0 to -2 of Figure 7, “perform computation actions called by...[a] program” which is “performed by processing circuits.” AMD1004, [0078]. Just as programs are “loaded into” the first memory 744 “for execution by processing circuits 721,” a POSITA would have

understood that programs are loaded into the first memory (PROC RAM 844) of Figure 8 for execution by processing circuits (PROC 821). AMD1004, [0079]. These programs define a network application that governs both what data is generated in the data processing operation performed by the hwa module (*compute node*) and that such data must be forwarded to the next *compute node* in the *execution pipeline*. AMD1004, [0076]–[0078], [0095]; AMD1003, ¶72. Because Dalal explains that the first connection layer 804-0 includes “direct inter-module connections” to hwa modules “of a same server,” a POSITA would have understood that the *data* and *instructions* stored in Dalal’s *node memory* facilitate *inter-node data communications* by causing execution process 842-1 to forward a result to another hwa module (*compute node*) in the same server. *Id.*



AMD1004, FIG. 8.

Furthermore, because running the *data and instructions* stored in Dalal's first memory (*node memory*) cause, upon processing, packet data to flow to a next hwa module, a POSITA would have understood that the data and instructions *facilitate inter-node data communications* via the switching circuit (*communications link*).
 AMD1004, [0035]–[0036], [0042], [0045], [0075]–[0078], FIGS. 1, 2B–2C, 7;
 AMD1003, ¶73.



AMD1004, FIGS. 2B, 7.

Element [1d.ii]

The portion of Dalal's node memory that stores “programs...for execution by processing circuits” represents an *algorithm memory* including *resource-specific memory locations* to support the *data processing operation*. AMD1004, [0076]–[0079], [0095], FIGS. 7–8; *supra*, Element [1d.i]; AMD1003, ¶74. Dalal explains that the program loaded into the *node memory* is written on a framework that “can

enable various execution processes...to perform computation actions called by” the program, which supports that Dalal’s node memory includes an *algorithm memory* for storing an *algorithm* that guides these computation actions. AMD1004, [0078].

Dalal’s *algorithm memory* includes *resource-specific memory locations*, at least because Dalal explains that the programs are loaded into the first memory (*node memory*) to be accessed via a data path 750-0 that is programmable to “provide connections as needed,” which indicates that a portion of the first memory (*node memory*) is pre-allocated to store these programs for access akin to the *algorithm memory* of the ’636 patent. AMD1004, [0079], [0084]; AMD1001, 40:25–41. Dalal explains that the first memory (*node memory*) represents a synchronous dynamic random access memory (SDRAM). AMD1004, [0079]. A POSITA would have understood that SDRAM comprises an array of cells having independent, addressable memory banks that can be pre-allocated to serve as *resource-specific memory*. AMD1003, ¶75, citing corroborating, AMD1042, p.3 (“SDRAM devices contain a...storage array”), AMD1045, 3:64–4:9 (allocating data among first and second banks of SDRAM).

Furthermore, it would have been obvious for a POSITA to implement Dalal’s first memory (*node memory*) with pre-allocated *resource-specific memory locations* to store the programs based on Ng’s disclosure of a “buffer” in a “RAM” that is associated with an “address” in the RAM. AMD1005, [0042], [0048], [0049],

[0065], [0066]; AMD1003, ¶76. Ng's buffers allocated for specific data are *resource-specific memory locations* akin to the '636 patent. AMD1001, 40:25–41 (“buffers” are “resource-specific...memory locations”); AMD1003, ¶¶76–77. In DNC, Dalal's first memory (*node memory*) would have been implemented with one or more buffers (*resource-specific memory locations*) pre-allocated at given addresses to assist the processing circuits in accessing information. AMD1006, [0078]; *supra*, §V.A.3.

Element [1e]

In Dalal, a data transfer fabric 716 provides data transfer paths. AMD1004, [0082], [0033]–[0036], [0040]–[0043], [0075], [0079], [0081]–[0089], FIGS. 1–2C, 7–8; *supra*, Element_[1d.i]. Dalal's processing circuits (*programmable logic component*) access the first memory (*node memory*) through the data transfer fabric via the path 750-0 (*data mover component*) such that path 750-0 facilitates *data communications* between these components. AMD1004, [0079], cl.1, FIG. 7; AMD1003, ¶78. The *data communications* are *intra-node* because the first memory and the processing circuits are located within the same hwa module (i.e., the same *compute node*). AMD1004, [0034], [0075]–[0088], FIG. 7; AMD1003, ¶79.

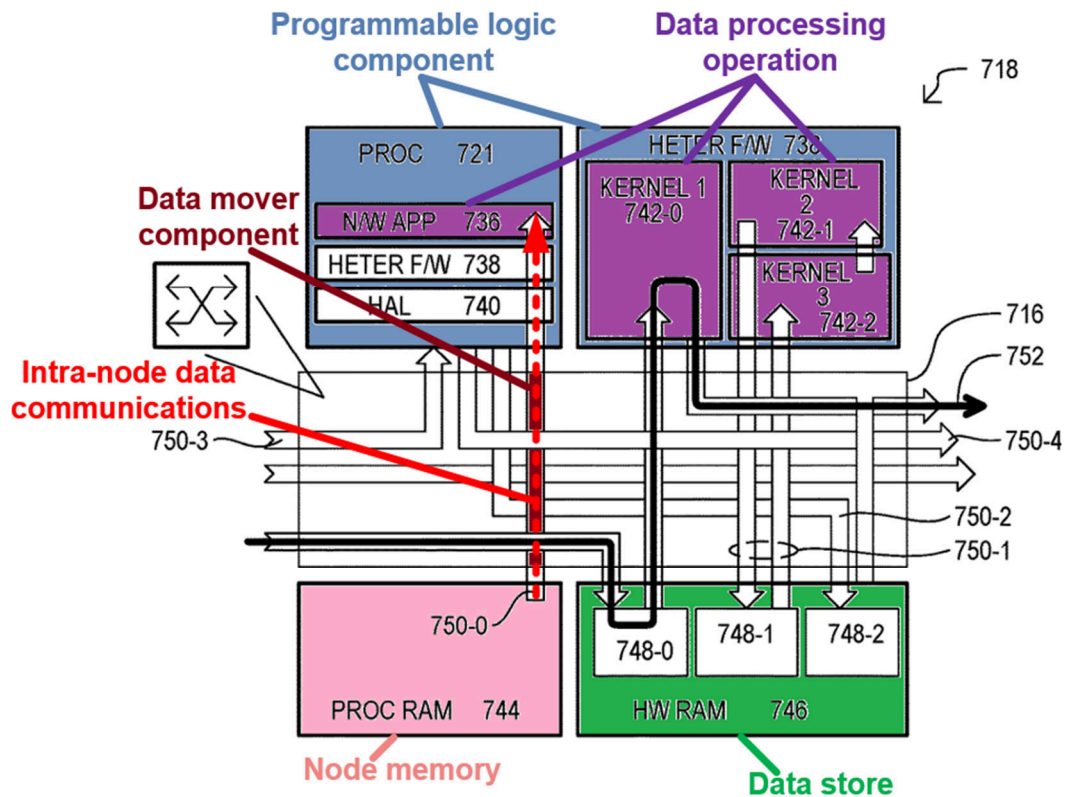


FIG. 7

AMD1004, FIG. 7.

Dalal explains that the data transfer fabric is “programmable” and “formed by buses on a programmable logic device, such as an FPGA,” much like the *data mover component* of the ’636 patent. AMD1004, [0084]–[0087], FIG. 7; AMD1001, 15:55–16:17, 16:45–61; AMD1003, ¶80, *citing corroborating* AMD1025, p.32.

Element [1f]

DNC renders Element [1f] obvious (1) based on Dalal’s teachings alone, and (2) based on a combination of Dalal-Ng. AMD1003, ¶81.

First, Dalal’s second memory 746 (*data store*) is accessed by various computing circuits including the processing circuits, 721, 738 (*programmable logic*

component) indicating that *data communications* occur between the *programmable logic component* and the *data store* of the same *compute node* (i.e., the communications are *intra-node*). AMD1004, [0078]–[0082], FIG. 7; AMD1003, ¶¶82–83. To *facilitate* these *intra-node data communications*, the computing element includes data transfer paths (*storage interface component*) through Dalal’s data transfer fabric 716 that connect the *programmable logic component* to the *data store*. AMD1004, [0078]–[0082], FIG. 7.

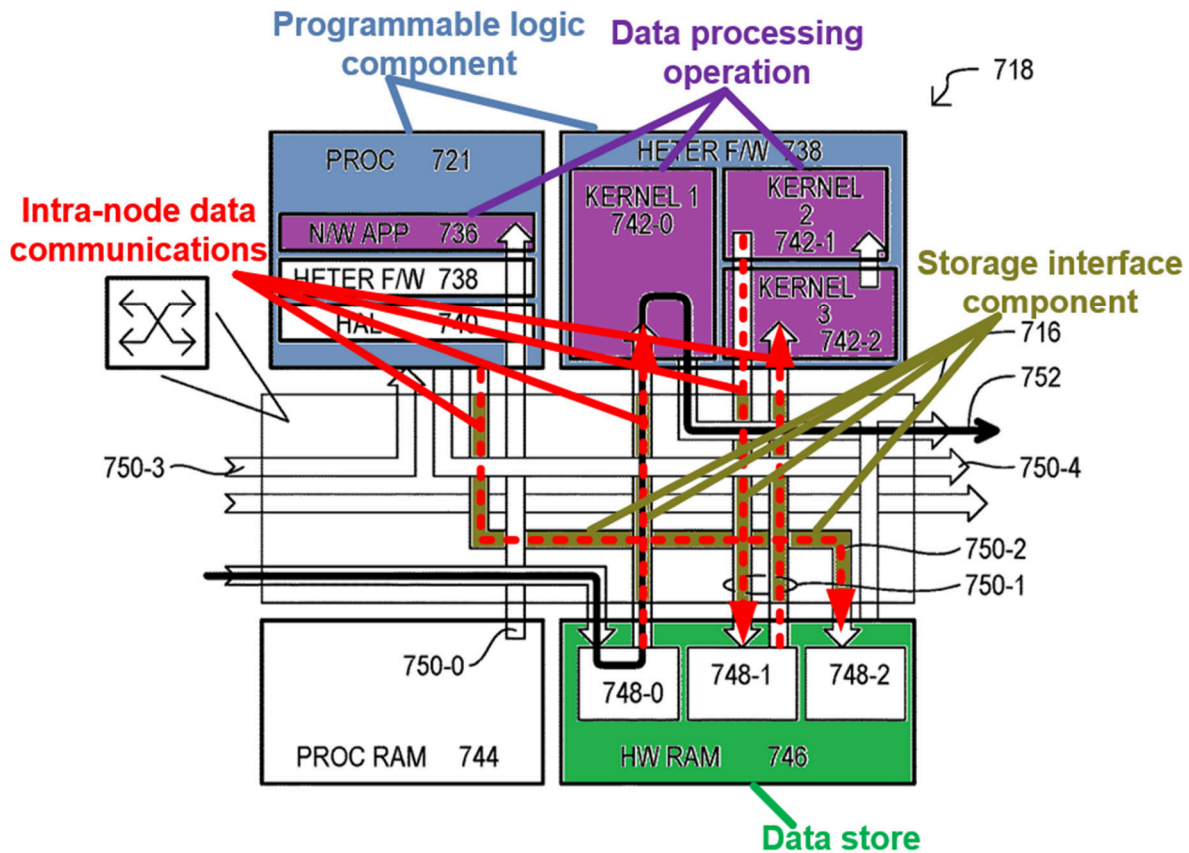


FIG. 7

AMD1004, FIG. 7.

Second, DNC also incorporates Ng's memory *interfaces*. AMD1005, [0035]–[0038], FIG. 3; AMD1003, ¶84. Ng's memory interfaces are coupled to a RAM and allow the neural network accelerator to read from the RAM and write to the RAM. AMD1005, [0036], [0038], [0040]–[0041], FIG. 3.

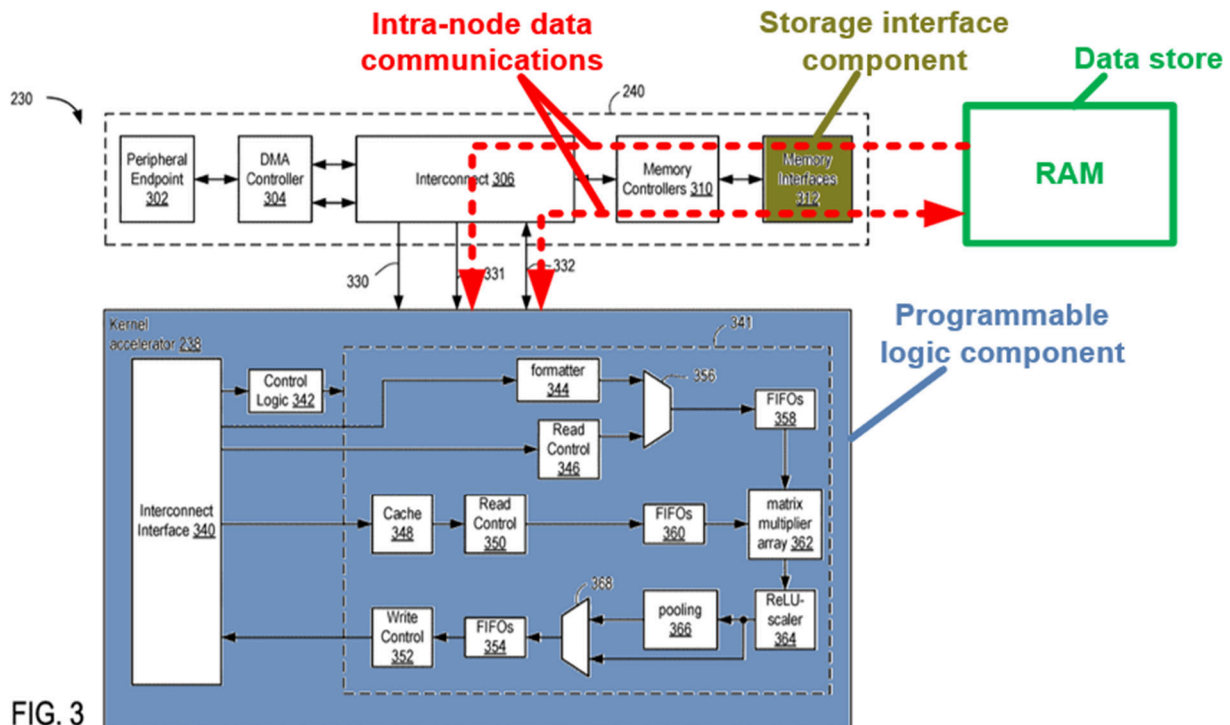


FIG. 3

AMD1005, FIG. 3.

In DNC, a POSITA would have found it obvious to implement Dalal's computing elements with a dedicated *storage interface component* to facilitate *intra-node data communications* over Dalal's data transfer fabric because doing so would have provided dedicated circuitry configured with one or more memory protocols of the data store (e.g., DDR RAM). AMD1004, [0080] ("second

memory...accessible by programmable logic (FPGA), such as static RAM (SRAM)”), [0085]; AMD1005, [0032], [0035], [0043]; AMD1003, ¶85; *supra*, §V.A.3. As depicted below in the modified version of Dalal’s Figure 7, the computing element in DNC would have included memory interfaces connected to the second memory (HW RAM) (***data store***), just as Ng discloses that “[t]he memory interfaces...are coupled to the RAM.” AMD1004, [0036]. AMD1003, ¶¶85–87

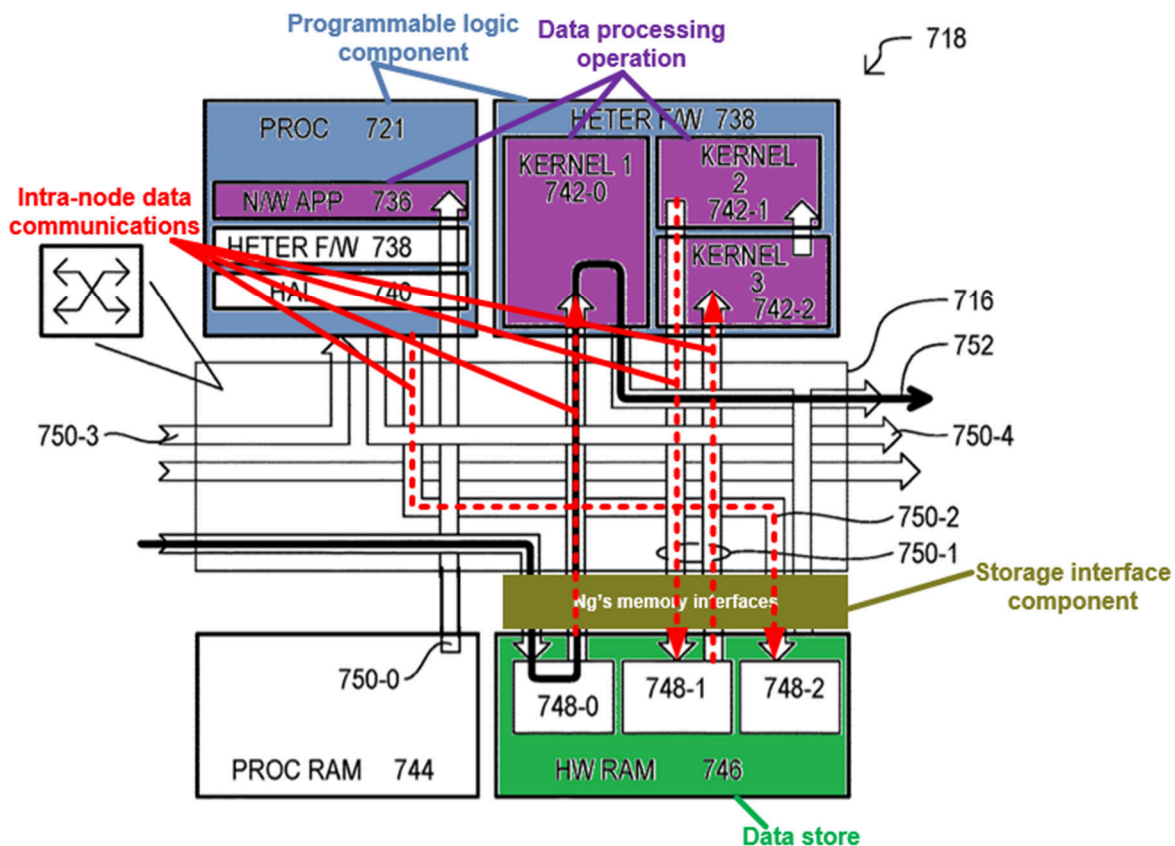


FIG. 7

AMD1004, FIG. 7 (modified).

Element_[1g]

DNC renders Element_[1g] obvious (1) based on Dalal's teachings alone, and (2) based on the combination of Dalal-Ng. AMD1003, ¶88.

First, Dalal's data transfer fabric provides different data transfer paths that include "connections between...processing circuits 721 (*programmable logic component*)...[and] second memory 746 (*data store*).” AMD1004, [0082], [0081] (“various possible connections”), [0084] (“providing connections as needed”), FIG. 7; *supra*, Elements_[1e]–[1f]; AMD1003, ¶89. As depicted below in Dalal's Figure 7, these connections include at least four (*a plurality*) of data transfer paths (*communications channels*) that *transfer data* between the processing circuits (*programmable logic component*) and the second memory (*data store*). *Id.*

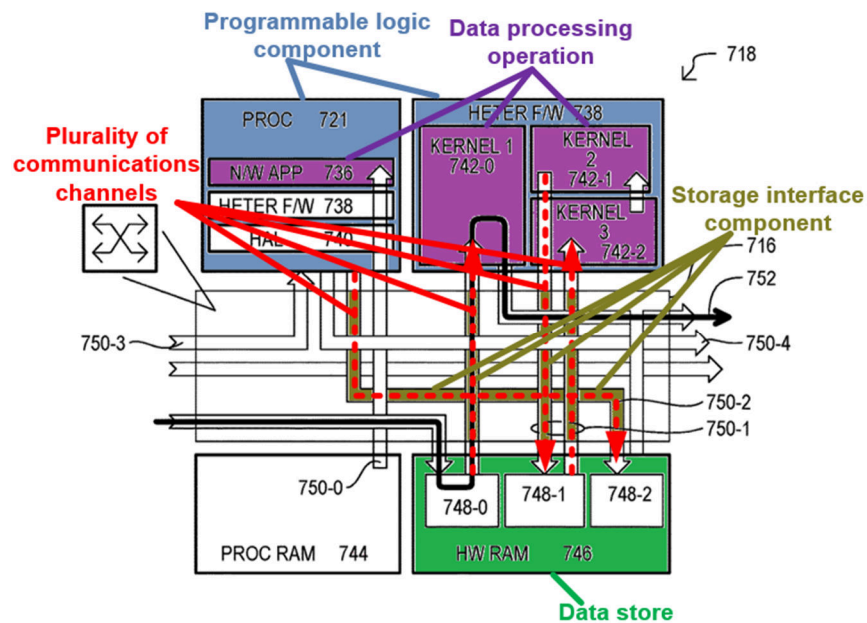
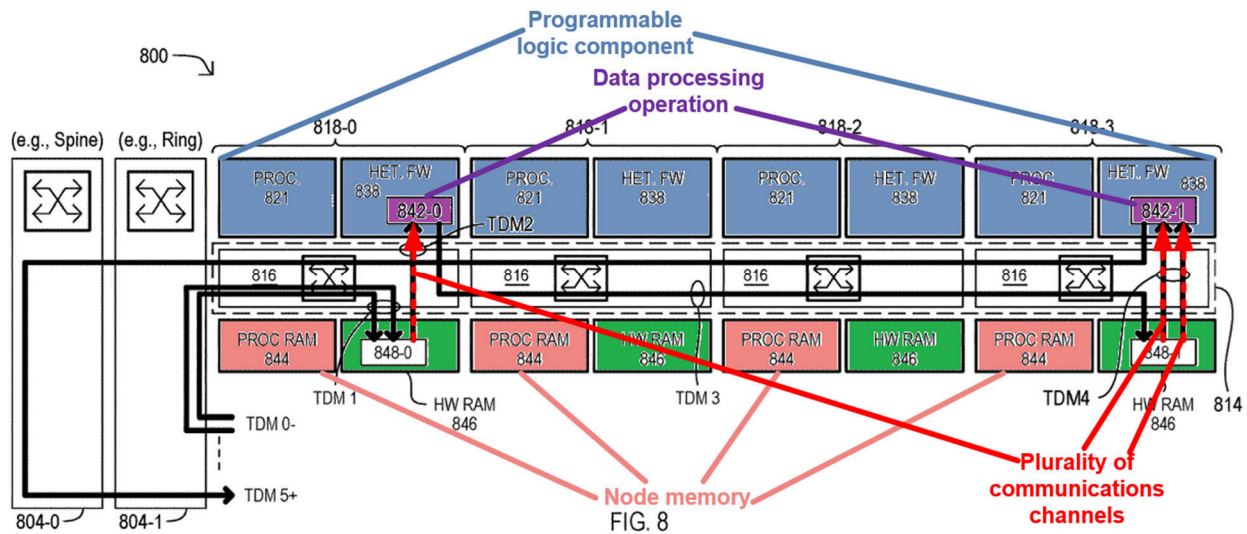


FIG. 7

AMD1004, FIG. 7 (modified).

Furthermore, Dalal's Figure 8 depicts a *plurality of communications channels* that transfer data between memory spaces (*data store*) and execution processes (*programmable logic component*) within an hwa module. AMD1004, [0094], [0095], FIG. 8; AMD1003, ¶¶90.



AMD1004, FIG. 8.

Second, as explained above, DNC also incorporates Ng's express teaching of memory interfaces (*storage interface component*) including a first memory interface used to read from the RAM and a second memory interface used to read from and write to the RAM (*plurality of communications channels*). AMD1005, [0035]–[0038], FIG. 3; AMD1003, ¶¶91–92; *supra*, §V.A.3. The modified version of Dalal's Figure 7 below, demonstrates that Dalal-Ng's memory interfaces would have utilized a *plurality of communications channels* that *transfer data*.

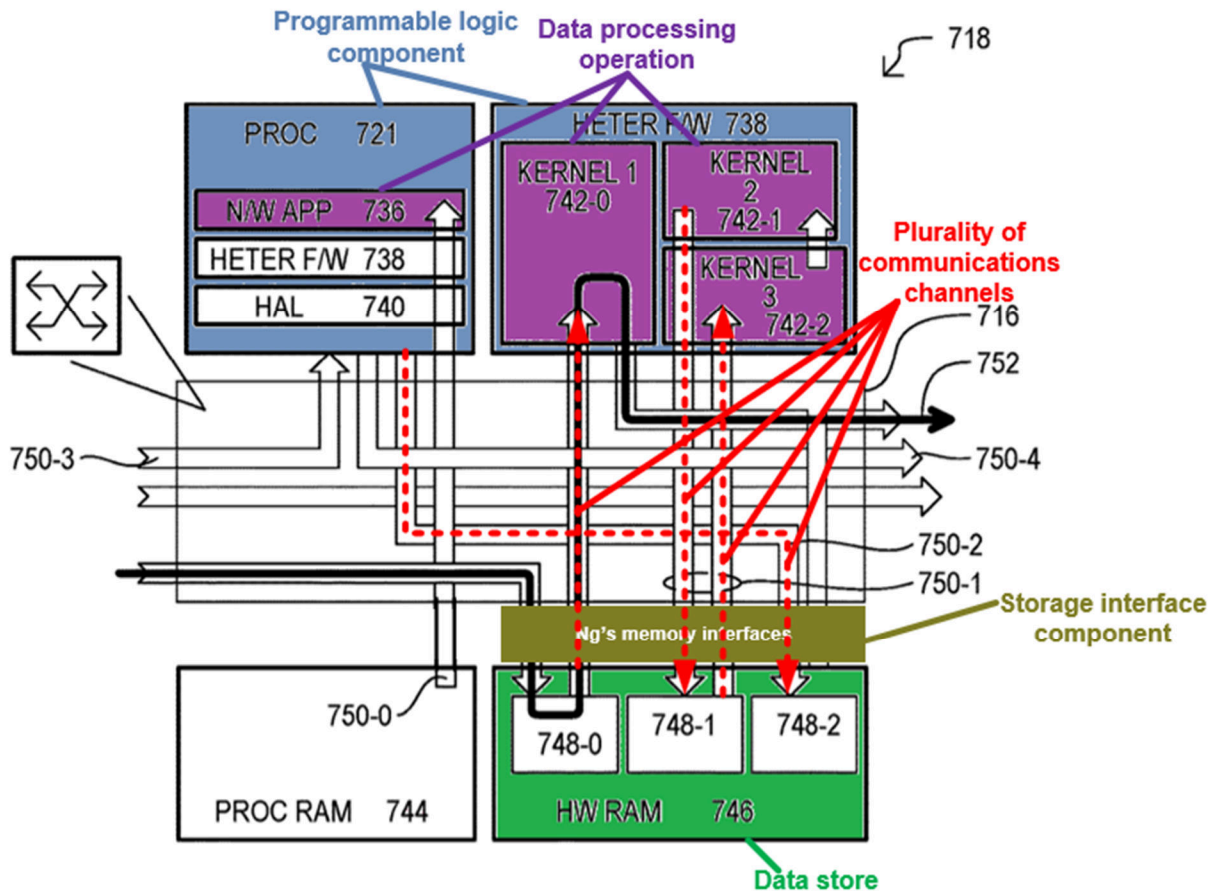


FIG. 7

AMD1004, FIG. 7 (modified).

Element [1h]

Dalal's **data processing operation** involves execution processes that perform **computation** actions called by a program. AMD1004, [0078], [0033] (“an hwa module...can include a number of computing elements...which can execute operations on received data...”), [0075] (“**computation** element 718 can be included in an hwa module...and can execute **computation** operations”), [0086] (“compute element...can be...a **computational** FPGA”); *supra*, Elements_[1b]–[1d.ii];

AMD1003, ¶93. This program “can be part of a machine learning library...to enable machine learning applications to run across multiple computing elements,” which Applicant acknowledged is *computationally expensive* during original prosecution of the application that ultimately issued as the ’636 patent. AMD1004, [0078], [0115] (“data-intensive computing”); AMD1002, 163 (Applicant asserted that an “example” of “computationally expensive algorithms” includes “Big Data or artificial intelligence applications”); *supra*, §II.B.

Dalal’s *execution pipeline* represents a *distributed processing compute environment* that executes *data processing operation* associated with a *computationally expensive algorithm* because a hardware accelerator module (*compute node*) and an adjacent hardware accelerator module (*adjacent compute node*) perform computation actions corresponding to the operation. AMD1004, [0078], [0042]–[0043]; AMD1003, ¶94. Dalal expressly discloses that a “*distributed* system” includes “computing elements” of a hardware accelerator module connected to “other devices” located “locally” or “remotely.” AMD1004, [0093], [0115], [0126], [0174].

Element [2]

Dalal explains that within a server, PCIe buses, serving as a fabric, connect hwa modules to each other and to other components. AMD1004, [0032]–[0036], [0043], [0064], [0089]–[0099], [0123], [0170], FIGS. 1, 2B, 8, 12, 18; *supra*,

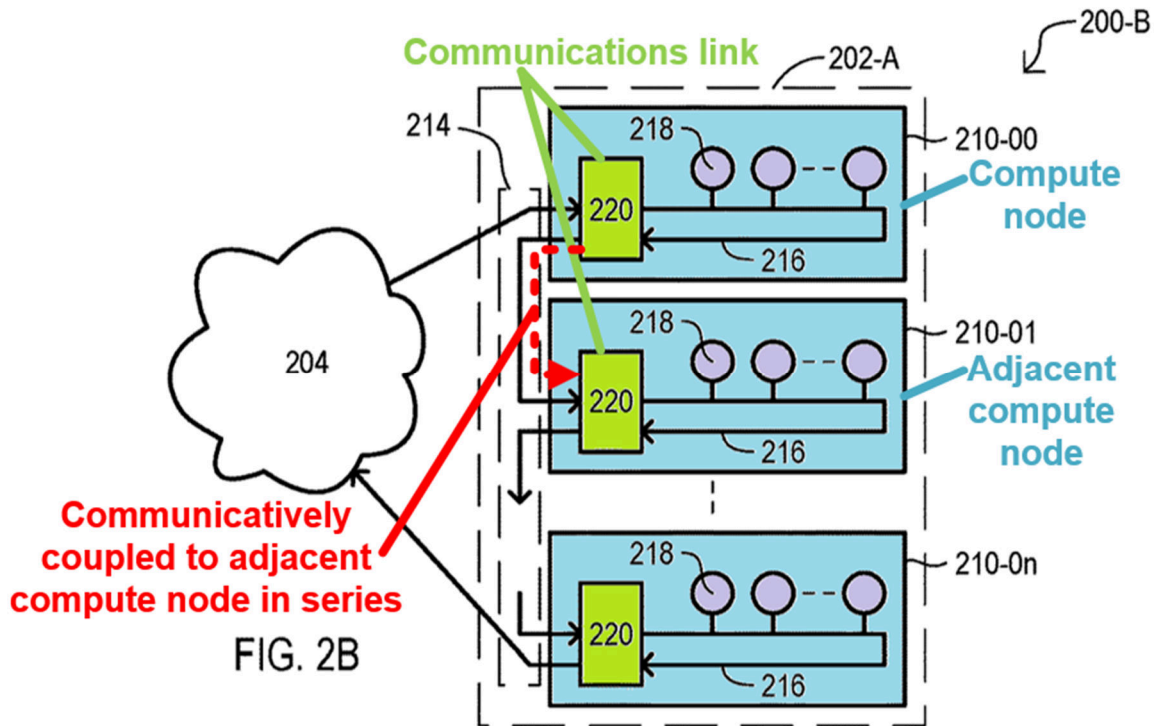
Elements_[1b]–[1c]; AMD1003, ¶95. Because the fabric is connected to the switching circuit (*communications link*) using the PCIe standard—a *serial communications protocol* that enables *bi-directional data communication*—it would have been obvious to a POSITA that the switching circuit (*communications link*) of a hwa module (e.g., 210-00) (*compute node*) employs a *serial communications protocol*. AMD1004, [0032], [0040]–[0048], FIGS. 1, 2B; *supra*, Element_[1a]; AMD1003, ¶95.

Multiple references confirm that PCIe is a *serial communications protocol* that enables *bi-directional data communication*. AMD1018, p.1 (“PCIe...is a *serial*...standard”), p.3; AMD1023, p.1; AMD1026, p.27; AMD1008, 3:30–55 (“*serial...bi-directional communication* between...PCIe devices”); AMD1003, ¶96. Ng’s disclosure further reinforces the obviousness of using a *serial communications protocol* such as PCIe. AMD1005, [0030]–[0043], FIGS. 2–4; AMD1003, ¶97; *supra*, §V.A.3.

Element_[3]

Dalal’s “hwa module 210-00” (*compute node*) is *communicatively coupled* to a “next hwa module 210-01” (*adjacent compute node*) *in series*. AMD1004, [0042] (“forwarded to a next hwa module”), [0045] (“serial flow”), [0047] (“processing...*in series*”), FIGS. 1, 2B; *supra*, Elements_[1pre]–[1a]; AMD1003, ¶98. As depicted below in Dalal’s Figure 2B, this communication occurs via the

switching circuit (*communications link*) of the hwa module 210-00” (*compute node*). *Id.*



AMD1004, FIG. 2B.

Element [6]

Dalal’s hwa modules have a series of computing elements, each including processing circuits that comprise a *programmable logic component*. AMD1004, [0033], [0076], [0077], FIG. 7; *supra*, Elements_[1pre], [1b]–[1c]. Ng describes a hardware accelerator that includes “*programmable* integrated circuits (ICs), such as *field programmable gate arrays (FPGAs)*,” and a POSITA would have implemented Dalal’s processing circuits (*programmable logic component*) with *FPGAs* as suggested by Ng because doing so would beneficially enable a developer

to customize the programmable logic component for specialized operations. AMD1005, [0026], [0032], [0058], [0081]–[0086], FIGS. 2–3, 9; AMD1003, ¶99; *supra*, §V.A.3. This is consistent with the fact that Dalal’s processing circuits include logic “derived from...**programmable** circuits,” and that the processing circuits are part of a computing element representing an “integrated circuit device, such as a computational **FPGA**.” AMD1004, [0077], [0087], [0033], [0065], [0080], [0096], [0104]; AMD1003, ¶100.

Element [7]

Dalal’s processing circuits within each hwa module (***compute node***) include “ARM **processors**...to control switch fabric 716,” including a **processor** that **manages operation** of the path 750-0 (***data mover component***) by “access[ing] first memory 744 through” the “path shown as 750-0.” AMD1004, [0079], [0085], [0029], cl.7; AMD1003, ¶101; *supra*, Elements [1pre], [1e].

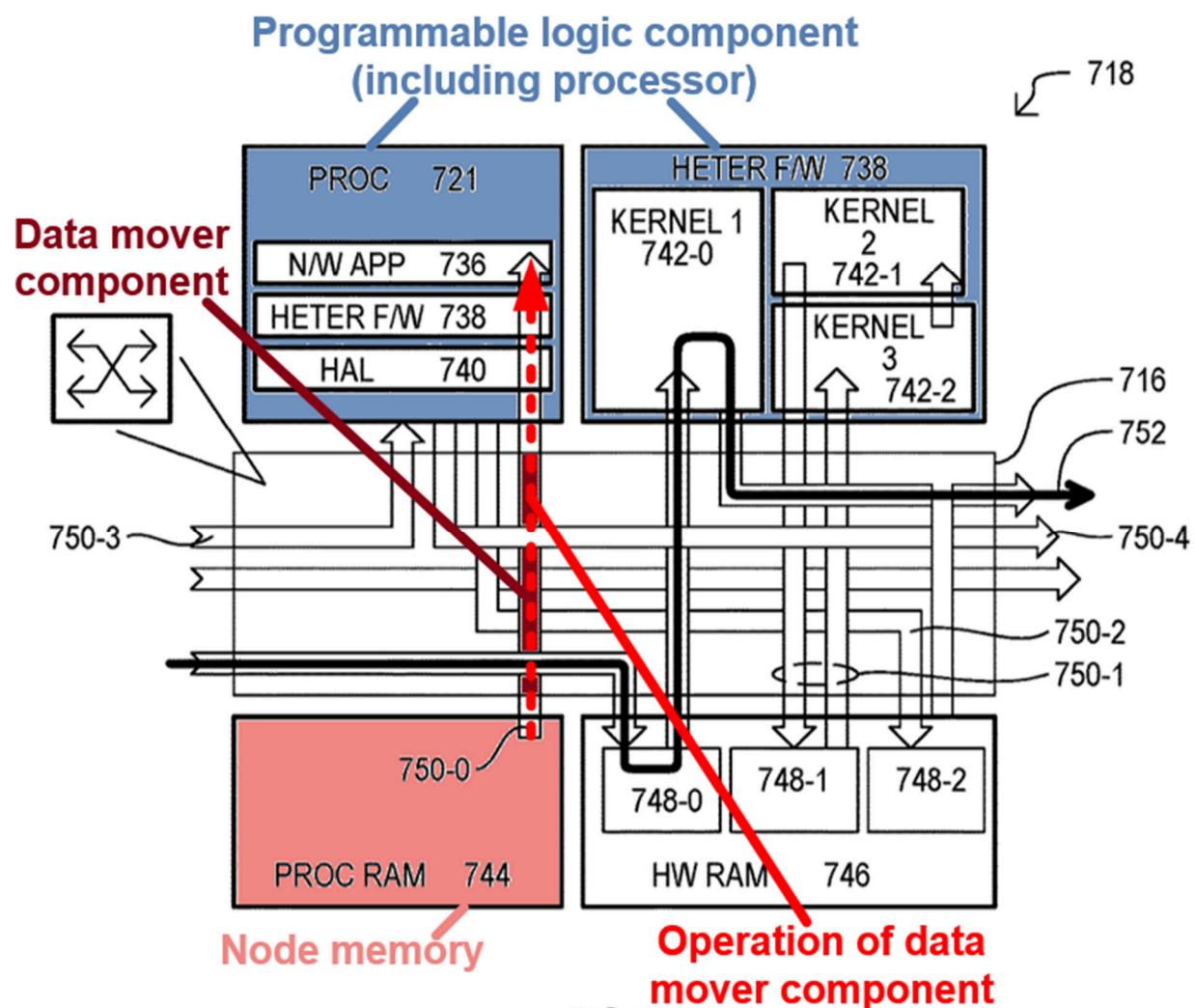


FIG. 7

AMD1004, FIG. 7.

DNC also incorporates Ng's supplemental teaching of a read control circuit and a write control circuit (*processor*) within a *programmable logic component* that facilitate communication between a *programmable logic component* and a memory via a *data mover component*. AMD1005, [0035]–[0038], [0040]–[0041], [0080], FIGS. 2–3; *supra*, §V.A.3; AMD1003, ¶102. A POSITA would have integrated

these circuits into Dalal's ARM processors to provide dedicated logic that is customized to manage data mover operations required for a data processing operation. *Id.*

Element [8]

The portion of Dalal's "data transfer fabric" (*data mover component*) connecting the processing circuits (*programmable logic component*) to the first memory (*node memory*) is a *device controller*. AMD1004, [0081]–[0086]; *supra*, Elements_[1pre], [1e], [7]; AMD1003, ¶103. This data transfer fabric is controlled by an ARM *processor* of the *programmable logic component*, similar to the '636 patent's *device controller*, which is operative to facilitate data communications either independently or in cooperation with control signals. AMD1004, [0085]; AMD1001, 16:45–61; AMD1003, ¶104. Incorporating Ng's supplemental teaching of memory *controllers* within the hardware accelerator *compute node* which provide the claimed *device controller* would have been obvious to a POSITA because this would have beneficially provided a way to manage processor-memory communications in every module. AMD1005, [0035]–[0037], [0043], FIG. 3; AMD1003, ¶105; *supra*, §V.A.3.

Element [9]

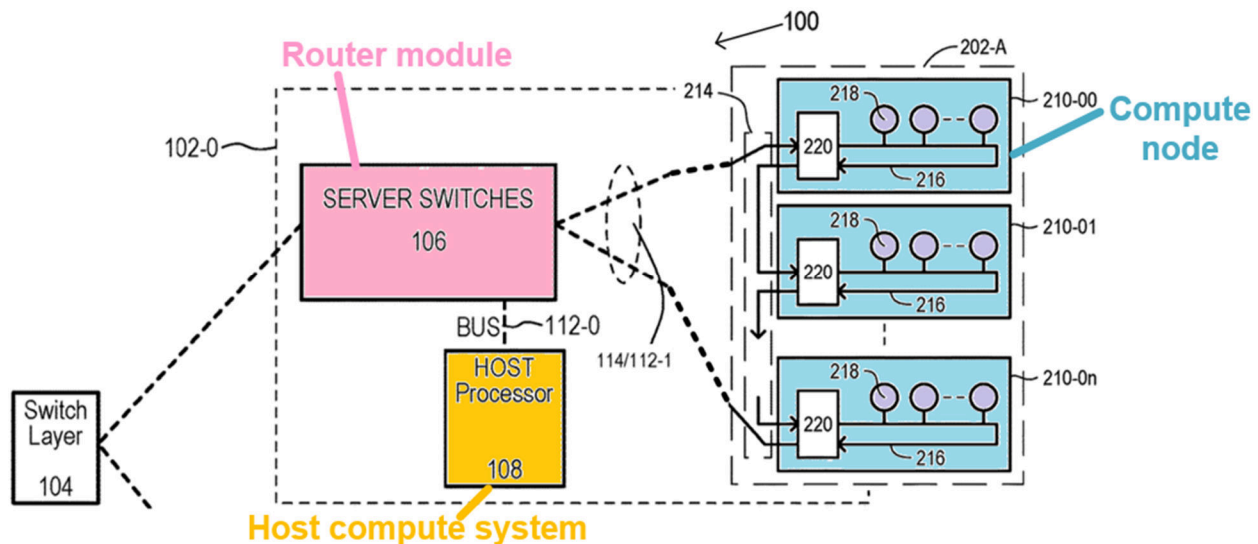
As described above, Dalal's host processor (*host computer system*) is connected to a switching circuit 220 (*communications link*) of an hwa module 210-

00 (*compute node*) that enables the hwa module to receive data from the host processor. AMD1004, [0032]; *supra*, Elements_[1pre]–[1a]; AMD1003, ¶106. DNC implements the host processor with Ng’s “host computer system” (*host computer system*) that executes *instructions* (e.g., input and weight matrices, and a work request) for a “neural network accelerator to perform neural network operations of each layer of the neural network.” AMD1005, cl.11, [0007]–[0008], [0021]–[0024], [0029], [0041]–[0050], FIGS. 4–5; AMD1004, [0032], [0070] (“host processor...can be any suitable processor device”), [0063], [0069], [0073], [0077], [0099]–[0103]; *supra*, §V.A.3; AMD1003, ¶¶106–107. In DNC, the *instructions* would influence *operation* of each hwa module (*each respective compute node*) in Dalal’s *execution pipeline* to efficiently execute a part of Ng’s operation such as a neural network layer. *Id.*

Element [10]

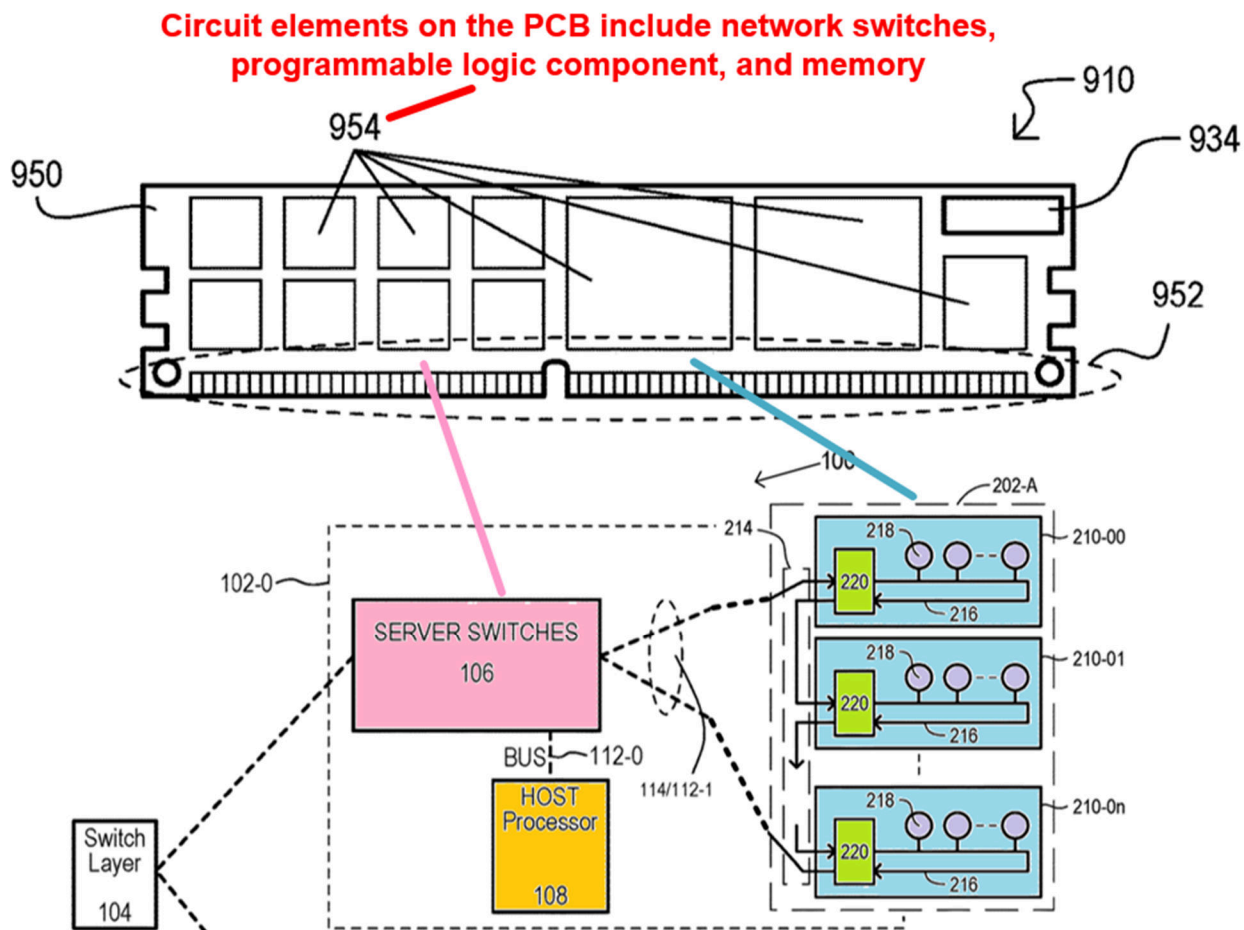
Dalal’s system includes server switches 106 (*router module*) that perform *routing* operations such as “forward[ing] packet data” to a hardware accelerator module (*compute node*) “based on information in the packet.” AMD1004, [0032]–[0036], [0096], [0123], [0170], FIG. 1; AMD1003, ¶108. As shown in Dalal’s FIGS. 1, 2B below, Dalal’s server switches (*router module*) are “connected” (*communicatively coupled*) to the host processor (*host compute system*) via a

peripheral component interconnect express (PCIe) bus. AMD1004, [0032]–[0036], [0123], [0170].



AMD1004, FIGS. 1, 2B Combination Figure.

Dalal’s server switches (*router module*) are “connected to host processor 108” and include “one or more network switches.” AMD1004, [0032], [0123], [0170], FIG. 1; *supra*, Elements_[1pre]–[1a]; AMD1003, ¶109. Dalal’s network switch is *combined with* and *integrated into* the same printed circuit board that also includes an hwa module (*compute node*). AMD1004, [0096], FIG. 9. Thus, a POSITA would have understood that Dalal’s printed circuit board is an *interface card* communicatively coupled to the *host compute system* into which *router module* and a *compute node* are *integrated* and *combined*. AMD1004, [0032], [0096]–[0097], FIG. 9; AMD1003, ¶110–112, *citing corroborating* AMD1006, [0038].



AMD1004, FIGS. 1, 2B Combination Figure; AMD1004, FIG. 9.

Element [11]

Dalal's processing circuits (*programmable logic component*) run programs (*computer-executable code*) stored in the first memory (*node memory*) to support a *data processing operation*. AMD1004, [0076]–[0079], FIG. 7; *supra*, Elements_[1b]–[1d.i]; AMD1003, ¶113. Like the '636 patent, Dalal's processing circuits use the *computer-executable code* of Dalal's programs (*information*) to “enable various execution processes...to perform computation actions” for

executing a *respective operation* associated with a *computationally expensive algorithm*. AMD1004, [0076], [0078], FIG. 7; AMD1001, 47:46–49. Dalal explains that received packet data is “processed by one or more elements 218 in hwa module 210-00” (*results of a respective preceding operation*), are “forwarded to a next hwa module 210-01, where it can be further processed” (*executes a respective operation*), and “[s]uch processing...continue[s] through multiple hwa modules” (passes *respective additional results* of the *respective operation* to a *next compute node* in the *execution pipeline*). AMD1004, [0042], [0094], FIGS. 2B, 8; AMD1003, ¶¶113–114.

DNC also incorporates Ng’s supplemental teaching of a “class of machine learning *algorithms*.” AMD1005, [0002], [0007], [0023]–[0024], [0042]. Such *computationally expensive* machine learning *algorithms* involve *operations* such as “matrix multiplication or convolution” whereby “the host instructs...to use the results data generated by the previous layer...as the input data matrix to the next layer.” AMD1005, [0021]–[0027], [0041]–[0047]; *supra*, §V.A.3; AMD1003, ¶115. Ng explains that the neural network accelerator (*programmable logic component*) “reads [a] work request...and initiates the specified neural network operations with the data referenced in the work request,” which reflects using information in computer-executable code to execute a respective operation, akin to Dalal’s processing circuits reading programs to perform computation actions. AMD1005,

[0063], [0045]–[0050]; AMD1003, ¶¶116–119. These teachings reinforce that it would have been obvious to implement a *compute node* to accept results of a *preceding operation* and pass results of the respective operation to a *next compute node*. *Id.*

Element [12]

Dalal discloses that programs are “loaded into first memory...for *execution* by processing circuits” and that “a program...can...enable machine learning applications to run.” AMD1004, [0078]–[0079], FIG. 7; *supra*, Element [11]. This confirms that every compute node (*e.g.*, hwa module) in Dalal’s execution pipeline uses a program having *computer-executable code* to execute a *computationally expensive* machine learning *algorithm*. AMD1003, ¶120. It would have been obvious to a POSITA that this *computer-executable code* is incorporated in a *bit stream*, because the data path 750-0 is time division multiplexed (TDM) and therefore uses a *bit stream* divided into channels for transferring data. AMD1004, [0033]–[0034], [0078], [0083], [0094]; AMD1003, ¶120, *citing corroborating*, AMD1040, 3:12–22 (describing “time division multiplexing” resulting in a bit stream having a “bit rate of f_2 ”); AMD1007, 1:23–30, 4:61–67. Furthermore, in DNC, the *information* includes “configuration parameters” which are uploaded to programmable logic in “configuration *bitstreams*” indicating that *the*

computationally expensive algorithm in DNC would have been *incorporated in a bit stream*. AMD1005, [0027], [0042]–[0043].

B. GROUND 1B: Dalal-Ng-Raghava Renders Claims 4, 5 Obvious

1. Raghava

Raghava’s system includes a “host processor coupled to a programmable acceleration coprocessor” that are “used to accelerate a variety of functions...[including] convolutional neural networks.” AMD1006, [0017]–[0039] FIGS. 1, 3–4; AMD1003, ¶121. The coprocessor includes one or more hardware accelerators including “external input-output (IO)” blocks that use an “Open NAND Flash Interface (ONFI)...for communicating with memory.” *Id.*

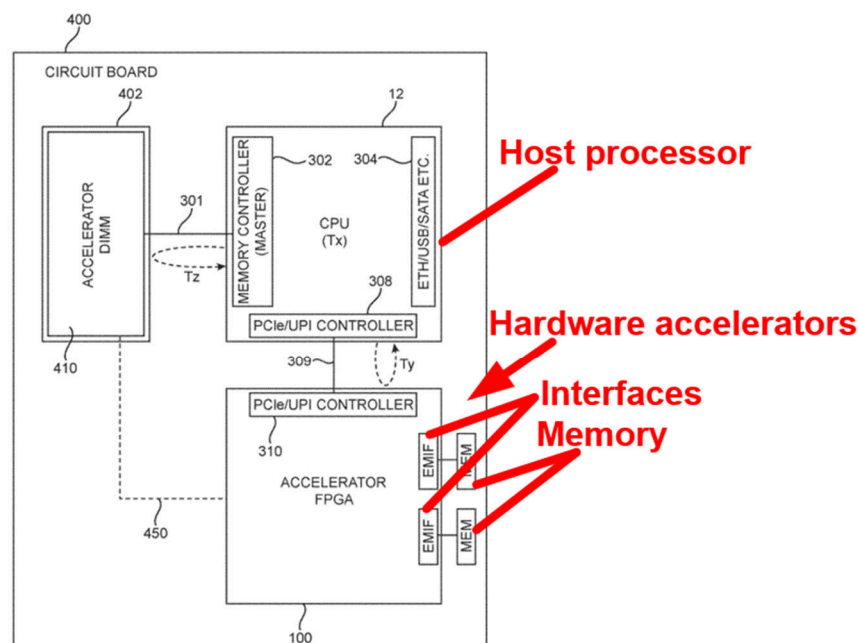


FIG. 4

AMD1006, FIG. 4.

2. Dalal-Ng-Raghava Combination

Dalal, Ng, and Raghava are analogous art to the '636 because all of these describe distributed systems where a host communicates with compute nodes having programmable logic to perform specialized operations. AMD1006, [0017]–[0022], [0030]–[0049], FIGS. 1, 3–4; *supra*, §§V.A.1-3; AMD1003, ¶122.

Combining the teachings of DNC with the NAND flash memory and ONFI storage interfaces of Raghava merely involves the application of a known technique to a known system to achieve predictable results, as demonstrated by the fact that Dalal and Ng both describe hwa modules including non-volatile memory (e.g., flash). AMD1004, [0033], [0067], [0096], [0105]; AMD1005, [0032], [0034], [0080]; AMD1003, ¶123. A POSITA would have reasonably expected success in combining Dalal and Ng because both describe a host that communicates with hardware accelerators to perform a data processing operation. AMD1004, [0032], [0078]; AMD1005, [0024]; AMD1003, ¶123.

The following reasons, taken separately or collectively, support the Dalal-Ng-Raghava Combination (DNRC). AMD1003, ¶124.

First, a POSITA would have understood that non-volatile memory such as flash provides multiple benefits including greater capacity and ability to maintain storage in the case of power loss. AMD1003, ¶125, *citing corroborating*, AMD1022, 1 (explaining that “non-volatile memory” such as “flash” has “notable

features such as non-volatility” and “fast write speed”); AMD1038, 1 (“large capacity flash memory”). Here, Dalal contemplates that hardware accelerator memories “can be of any suitable type” such as “RAM” or “flash memory.” AMD1004, [0067]. Implementing Dalal’s hwa modules with Raghava’s NAND flash memory would have beneficially increased memory capacity and removed power dependency on continued storage using a “suitable” type of memory. AMD1003, ¶125.

Second, because Raghava explains that “external IO blocks 312” “may include memory controllers for supporting a wide variety of external memory protocols” including “ONFI,” a POSITA would have been motivated to implement the storage interface component of DNC with aspects of Raghava’s IO blocks to ensure that any kind of memory connected to the accelerator is paired with a compatible interface. AMD1006, [0033], [0032]; AMD1004, [0067]; [0037]; AMD1003, ¶126.

Third, a POSITA would have understood that NAND flash memory and ONFI represent one of a finite number of predictable solutions for configuring memory of an hwa module. AMD1003, ¶127. Indeed, Dalal and Ng describe hwa modules that include flash memory. AMD1004, [0067]; AMD1005, [0032]; AMD1003, ¶127.

3. Claim Element Analysis

Element_[4]

The '636 patent explains that “all Flash is EEPROM (as a technical matter),” suggesting that any flash memory device includes an *EEPROM device*. AMD1001, 10:65–11:21; AMD1003, ¶128. Dalal’s second memory (HW RAM) corresponds to the *data store*. AMD1004, [0075], [0080]–[0082], [0094], cls.1, 11; *supra*, Element_[1b]; AMD1003, ¶128; *supra*, §V.B.2. For the reasons noted above, in the DNRC, the *data store* of *each respective compute node* in the *execution pipeline* would have been implemented with Raghava’s accelerator FPGA, which includes an *EEPROM*. *Id.*

In particular, Raghava discloses that an accelerator FPGA (*programmable logic component*) includes external input-output (IO) blocks (*storage interface component*) connected to a memory (*data store*) AMD1006, [0033]–[0034], [0014], [0046], [0074], cl.10, FIGS. 3–5; AMD1003, ¶129. Raghava’s IO blocks support the “Open NAND Flash Interface (ONFI)” and are “suitable interfaces for communicating with memory,” indicating that the memory includes NAND flash (*EEPROM device*). *Id.*

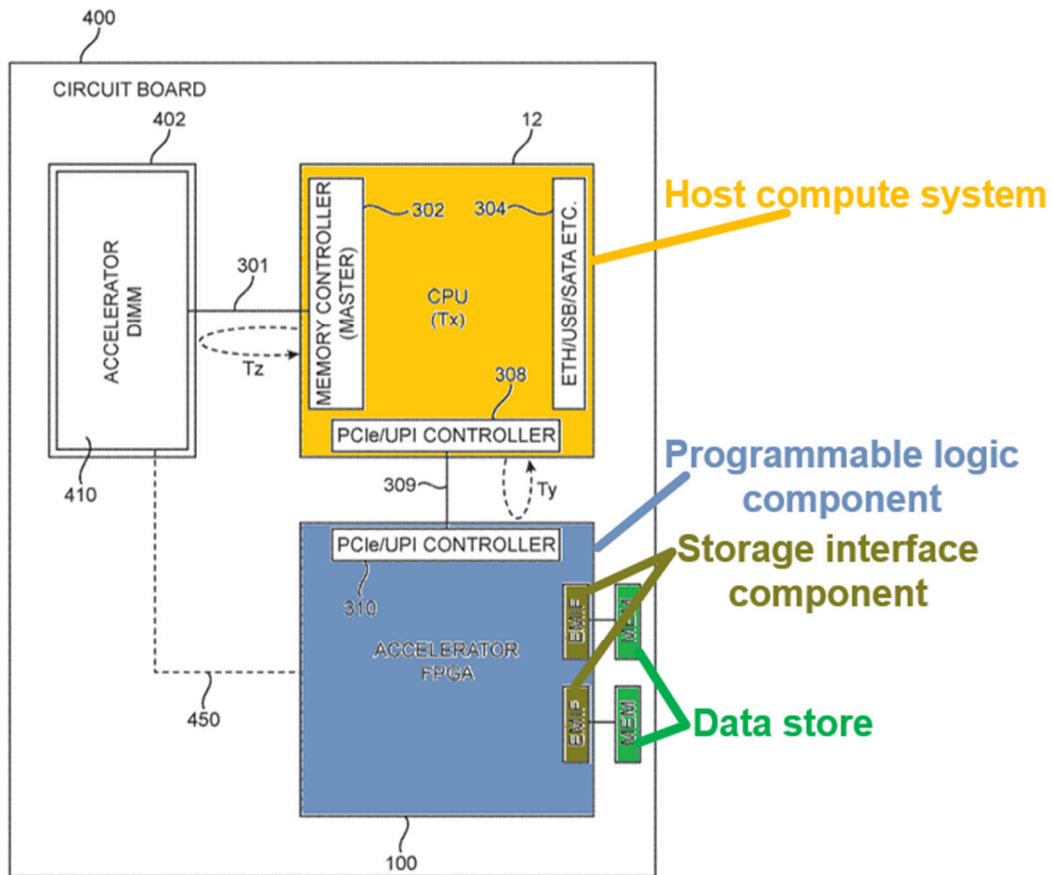


FIG. 4

AMD1006, FIG. 4.

Element_[5]

Dalal's data transfer fabric, or the combination of the data transfer fabric with Ng's memory interfaces corresponds to the *storage interface component*. AMD1004, [0075], [0079]–[0083], [0089]–[0090], [0094], cl.1, 18–20; AMD1005, [0035]–[0036], [0038]; *supra*, Elements_[1f]–[1g]; AMD1003, ¶130. For the reasons noted above, in DNRC, the *storage interface component* of each *respective compute node* in the *execution pipeline* would have been implemented with

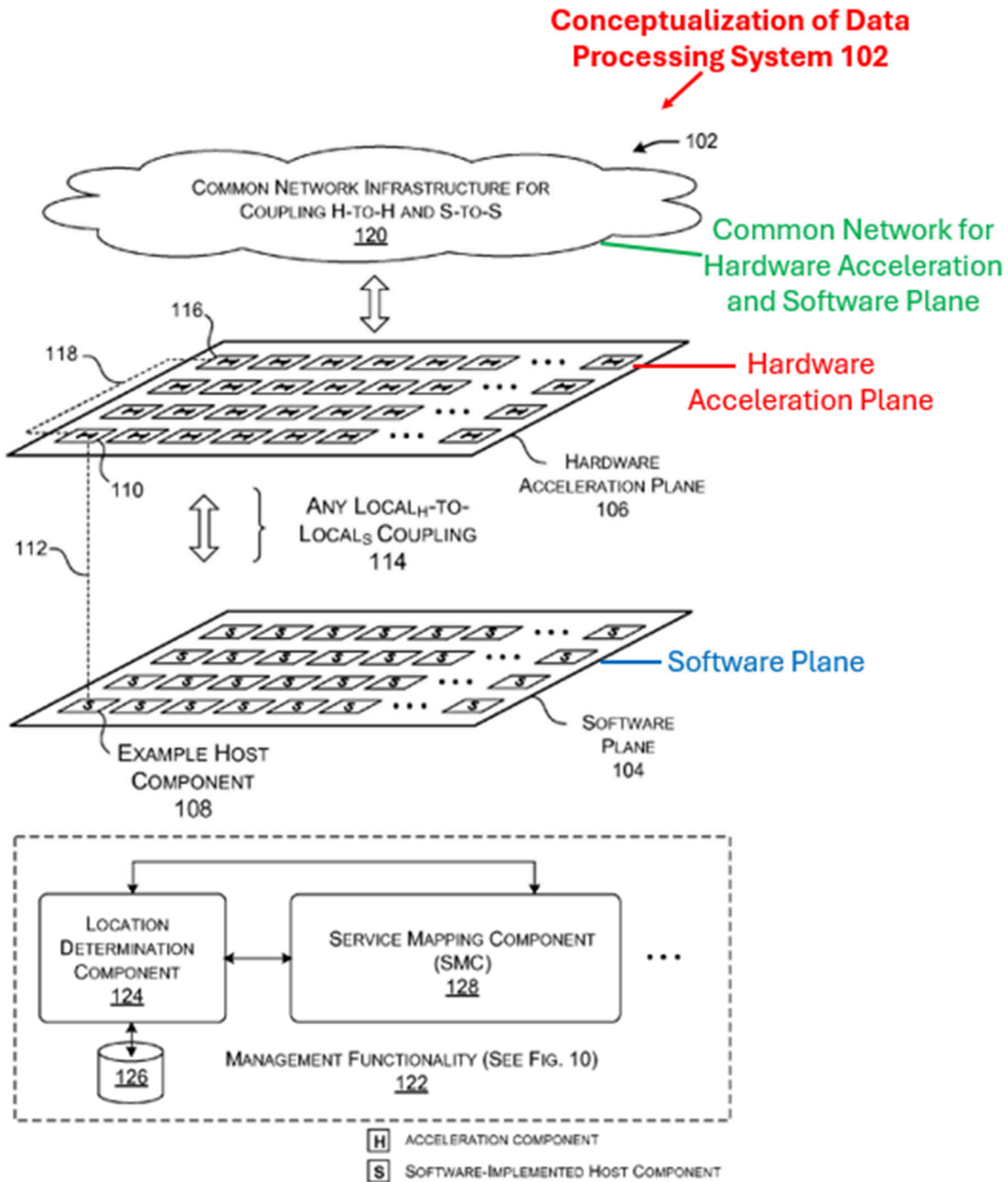
Raghava's input-output (IO) blocks that employ **ONFI**. AMD1006, [0033], [0014], [0046], [0074], cl.10, FIGS. 3, 5; *supra*, Element_[4]; AMD1003, ¶130; *supra*, §V.B.2. A POSITA would have found it obvious to make this combination because doing so would have provided nonvolatile storage at increased capacity. *Id.*

C. GROUND 2A: Heil-Jun Renders Obvious Claims 1–4, 6–9, 11–12

1. Heil

Heil discloses data processing systems for implementing multi-component services. AMD1010, [0003], [0040], [0053], [0218]–[0248]. The systems include multiple server unit components, each having a host component paired with an acceleration component. *Id.* These server unit components are communicatively connected to execute services. *Id.*; AMD1003, ¶132.

Heil's data processing systems (*e.g.*, 102 and 802 shown below) can be conceptualized as having a first logical network forming a software plane 104 with the host components and a second logical network forming a hardware plane 106 with the acceleration components. AMD1010, [0048]–[0058], [0089], FIGS. 1, 8. In Figure 1 (below), the two logical networks share a common network. *Id.*; AMD1003, ¶133.

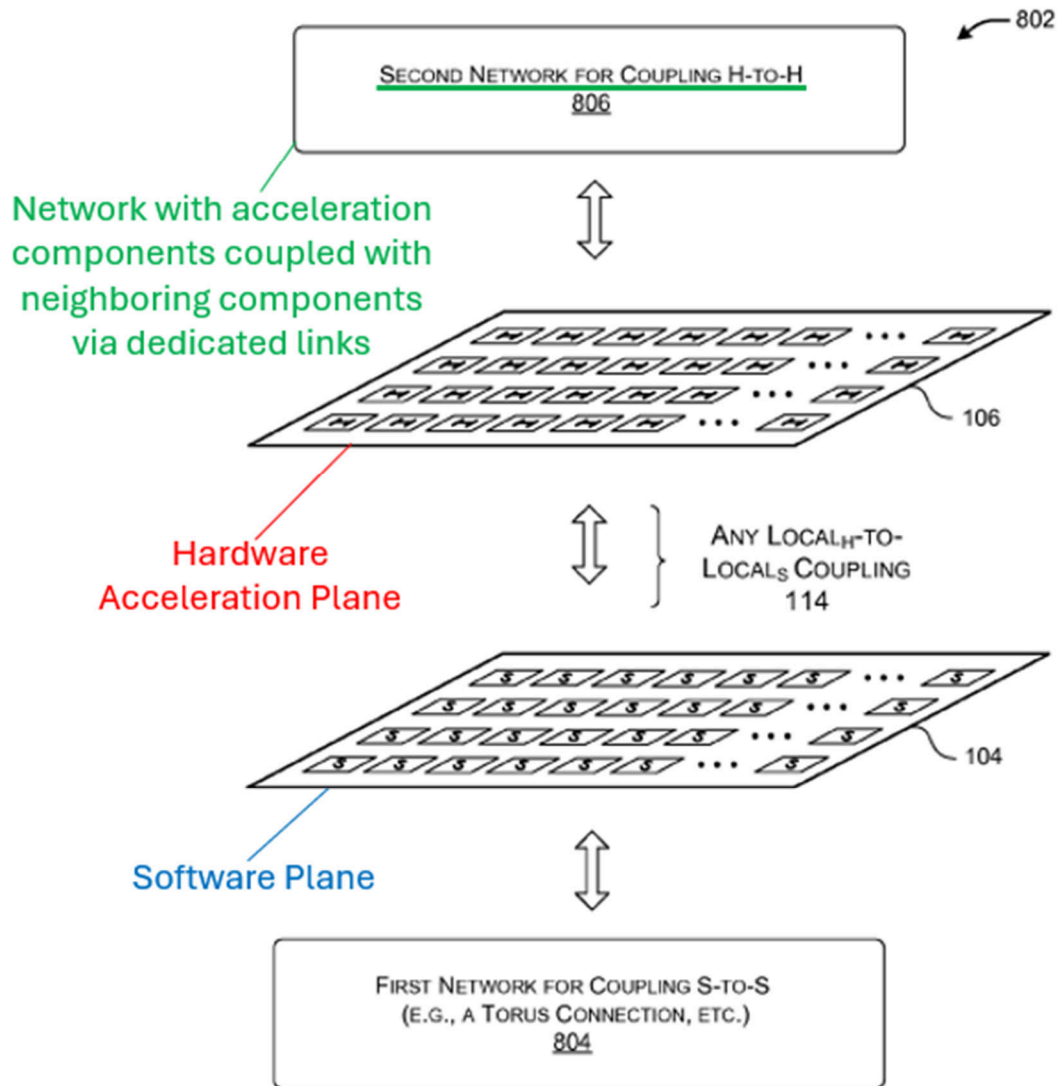


AMD1010, FIG. 1.

Heil's Figure 8 (below) is "[l]ike the data processing system 102 of FIG. 1" except it uses a second physical network for the hardware plane that is formed using

dedicated links to couple neighboring acceleration components. AMD1010, [0089], [0014], [0084]; AMD1003, ¶134.

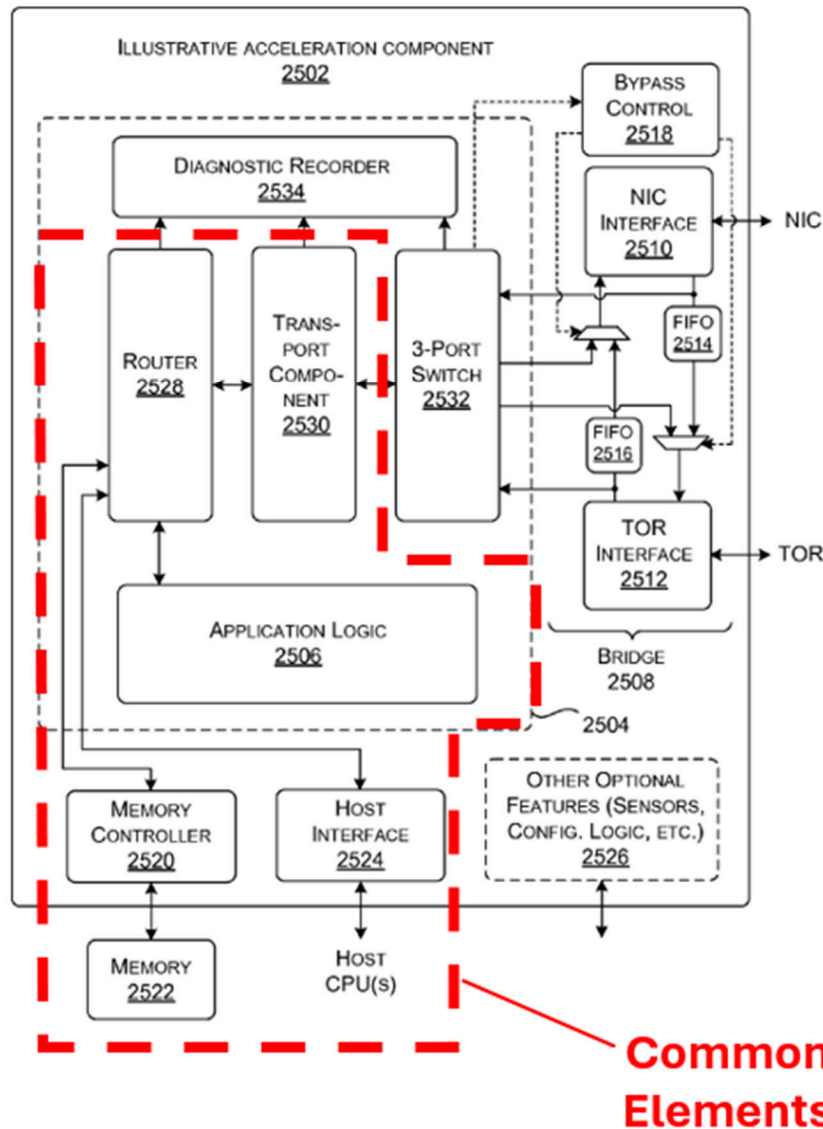
**Conceptualization of Data
Processing System 802**



AMD1010, FIG. 8.

Both Figures 1 and 8 represent Heil's acceleration components with "H" in hardware acceleration plane 106. Figure 25 (below) shows an illustrative example

of these acceleration components that includes application logic 2506, memory controller 2520, memory 2522, host interface 2525, router 2528, and transport component 2530. AMD1010, [0022], [0159]–[0168]. A POSITA would have understood or found obvious that at least these elements are common across acceleration components referenced in systems 102 and 802. AMD1003, ¶135.

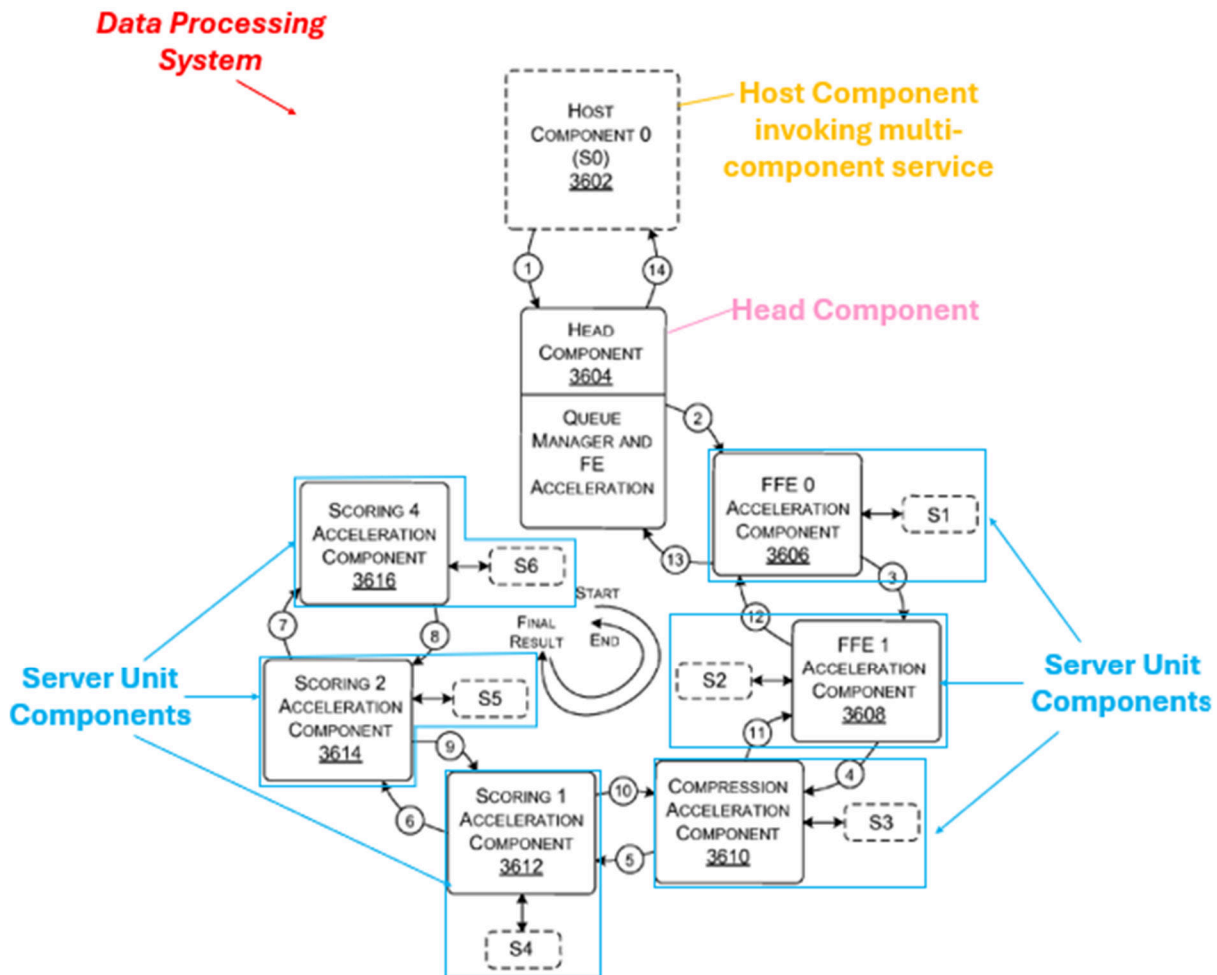


AMD1010, FIG. 25.

For example, while Heil states that Figure 25 is implemented for system 102, Heil also explains that “other data processing systems [*e.g.*, system 802] may use acceleration components having architectures which vary in one or more ways.” AMD1010, [0159]–[0162]. In view of this disclosure, a POSITA would have understood or found obvious that Heil’s acceleration components in system 802 forms the second network via dedicated links (*e.g.*, at TOR interface 2512) and would have still utilized the above identified common elements to implement core capabilities. AMD1003, ¶136 (describing the core capabilities of each common elements).

A POSITA would have also found it obvious to implement these common elements with system 802 because utilizing the same elements to fulfill the same core capabilities would have allowed system 802 to operate. *Id.*; *Bos. Sci. Scimed, Inc. v. Cordis Corp.*, 554 F.3d 982, 991 (Fed. Cir. 2009) (“Combining two embodiments...in a prior art patent does not require a leap of inventiveness.”). A POSITA would have also understood that implementing these common components across systems 102 and 802 would have provided similar benefits as the described outer/inner shell architecture does “across application scenarios” including negating the need to “recreate these common components” and reducing the risk that a “core inner or outer shell functions [is altered] in a manner that causes problems within the data processing system.” AMD1003, ¶¶137–138; AMD1010, [0159]–[0162].

Turning to an example of a multi-component service, Figure 36 (below) shows a service for assigning “a ranking score to a pairing of a query and a document.” AMD1010, [0218]–[0229], FIGS. 36, 40. A host component 3602 invokes the service by sending a request to head component 3604. *Id.* Server unit components execute the invoked service in a processing pipeline with coupled acceleration components (e.g., 3606–3616). *Id.*, [0068] (“pipeline”); AMD1003, ¶139.

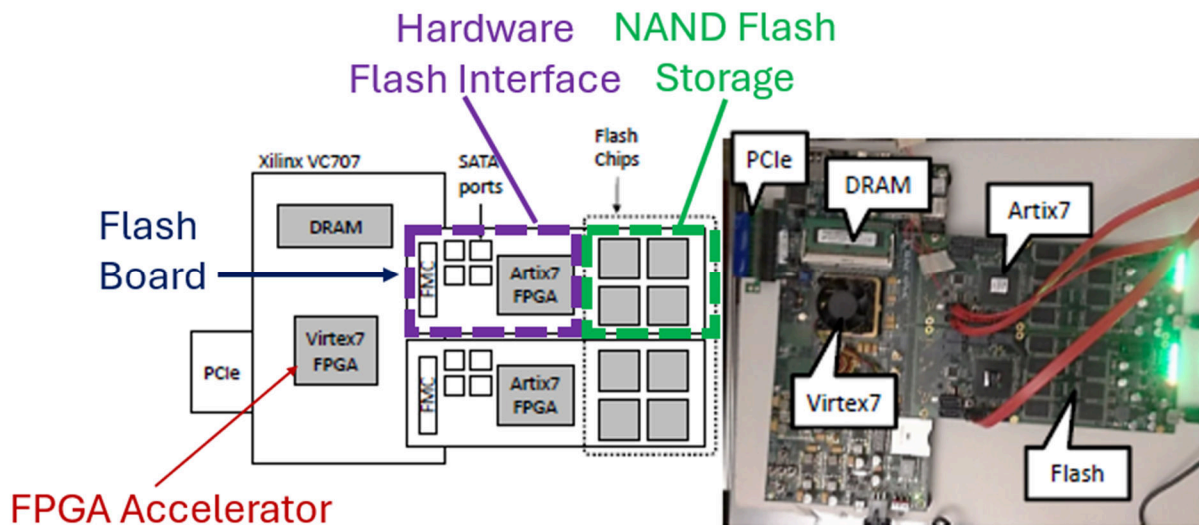


AMD1010, FIG. 36.

2. Jun

Jun qualifies as prior art at least because it is a technical paper published in the Proceedings of the 42nd Annual International Symposium on Computer Architecture, and publicly accessible at least by 2016, years before the Critical Date. AMD1013 ¶¶6-9 (explaining that Jun was made available to the public shortly after April 6, 2016).

Jun describes a data processing system that includes multiple FPGA-based compute nodes. AMD1011, pp.1–5, 7–9, 12–13. Jun’s nodes integrate NAND Flash Storage (Flash) with an FPGA processor device to execute operations on large data sets. *Id.* Jun explains that it is advantageous to use Flash in place of DRAM memory to effectively handle tasks on large datasets. AMD1011, pp.2–8, 12. Jun’s Flash is deployed in conjunction with a multi-channel hardware flash interface and connectors to the FPGA processor on a flash board. *Id.* The hardware interface and connectors on the flash board facilitate communication between Jun’s FPGA Accelerator and Flash. *Id.*; AMD1003, ¶¶140–141.



AMD1011, FIG. 10.

3. Heil-Jun Combination

Heil and Jun qualify as analogous art because they share the '636 patent's field of endeavor in data processing systems employing reconfigurable hardware. AMD1003, ¶142; AMD1001, 2:31–48, 3:8–28, 38:32–43; FIG. 17; *supra* §§VC.1–2. They are also reasonably pertinent to a technical problem shared with the '636 patent of accelerating processing by offloading tasks to reconfigurable hardware. *Id.*; AMD1010, [0051], [0224], AMD1011, pp.1–6. Indeed, Jun states it “has similar

goals in terms of reconfigurable hardware acceleration” to Heil’s FPGA/accelerator formed network. AMD1011, p.4⁴; AMD1003, ¶¶142–143.

Heil’s data processing system utilizes multiple server unit components each comprising a host component and an FPGA-based acceleration component. *Supra*, §V.C.1. As such, a POSITA would have been motivated to implement Jun’s flash board—including its Flash and hardware flash interface, and connectors—in place of Heil’s DRAM memory elements. AMD1003, ¶144.

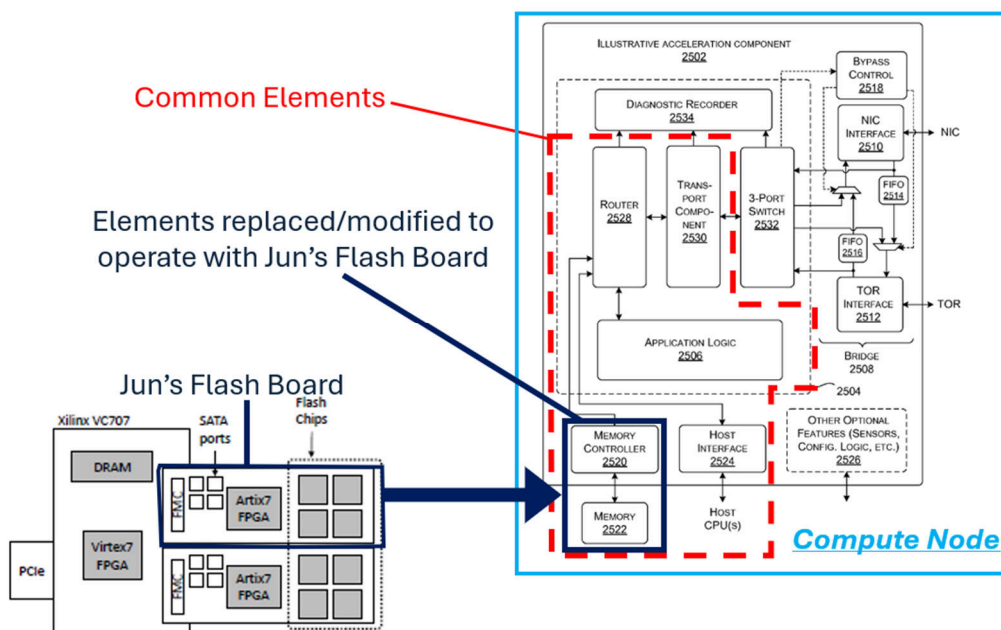


Illustration of the Heil-Jun Combination (HJC Illustration).

⁴ Jun references AMD1014 describing Microsoft Project Catapult. AMD1011, pp.4, 14; *generally* AMD1014. This project refers to the same technology as Heil. AMD1003, ¶143.

The following reasons, taken separately or collectively, support the Heil-Jun Combination (**HJC**). AMD1003, ¶¶145–151.

First, a POSITA would have understood that implementing Jun’s flash board elements allows Heil’s data processing system to accelerate multi-component services on larger datasets. AMD1003, ¶146. Jun explains that hosting large datasets on DRAM-based acceleration components requires a large cluster of nodes. AMD1011, pp.2–4, 12. Replacing DRAM with Flash enables high-capacity data hosting on fewer nodes, reducing the required number of nodes or expanding the capacity of a given collection of nodes. AMD1011, pp.3–4; AMD1003, ¶146. A POSITA would have been motivated to implement HJC to accelerate services on datasets that exceed the DRAM capacity for the collection of nodes in Heil’s data processing system. *Id.*

Second, a POSITA would have recognized that implementing Jun’s flash board elements within Heil’s acceleration component facilitates a reduction in architectural complexity by allowing implementations of services to require fewer server unit components. AMD1003, ¶147. It would have been obvious to the POSITA that replacing Heil’s DRAM with Jun’s Flash would have allowed for a reduced number of server unit components because each NAND-Flash-based acceleration component can handle a larger dataset locally. *Id.*; AMD1011, Abstract. Reducing the number of server unit components provides at least the

following benefits: simplifies the system's physical wiring, reduces operational latency and network traffic, reduces maintenance requirements, and makes the system easier to implement and use. *Id.*

Third, it would have been obvious to a POSITA that implementing Jun's "flash board" elements in Heil's acceleration component would have facilitated a reduction in capital cost because Flash is a more cost-effective memory medium than DRAM. AMD1003, ¶148. As Jun explains, Flash Storage is "an order of magnitude cheaper price compared to DRAM." AMD1011, p.3, Abstract, 4, 12; AMD1003, ¶148 (*citing* AMD1037).

Fourth, a POSITA would have recognized that replacing Heil's DRAM-based storage with Jun's flash board elements would have facilitated improved power efficiency. AMD1003, ¶149. As Jun explains, Flash "consumes an order of magnitude less power" than the DRAM-based alternative. AMD1011, pp.3, 11; AMD1003, ¶149.

Fifth, a POSITA would have found it obvious to combine the teachings of Heil with Jun because the combination merely involves the application of a known technique to a known system to achieve predictable results. *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 417 (2007). Jun recognized a problem with existing DRAM-based acceleration components and described a straightforward technique to address it using Flash. AMD1003, ¶150. As such, combining the teachings of Heil and Jun

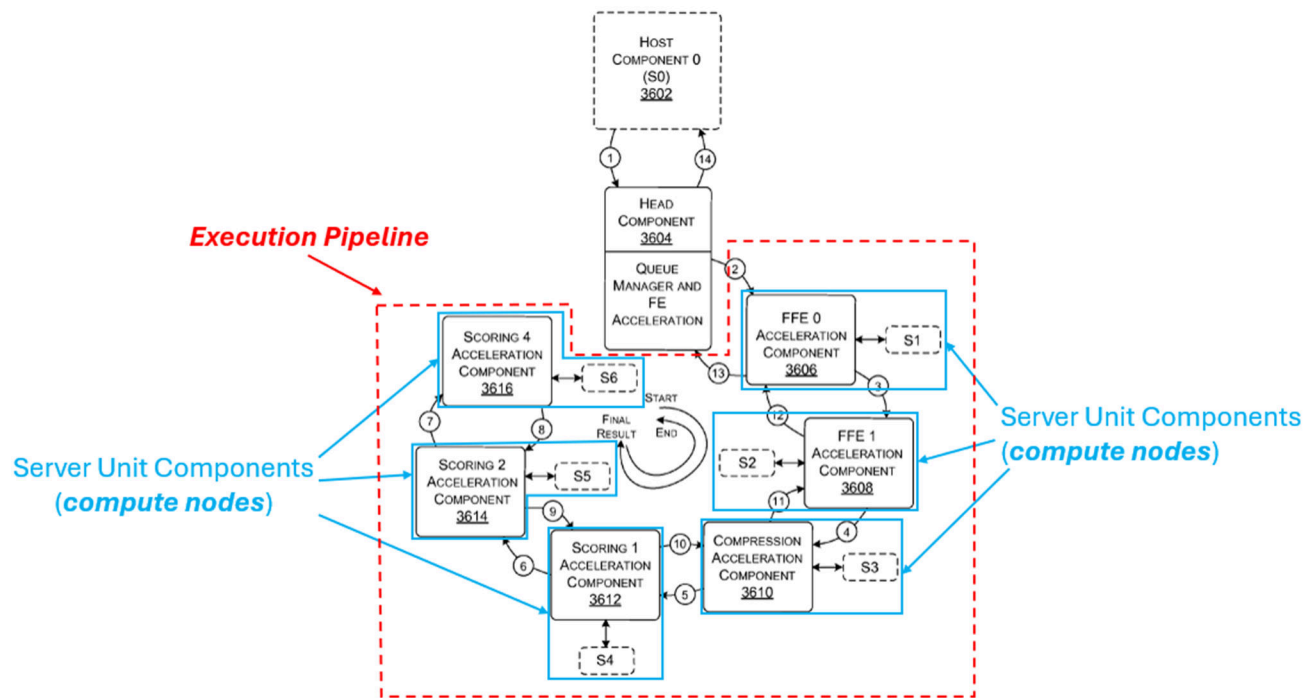
would have been an obvious simple substitution of known elements that a POSITA would have reasonably expected to succeed at least because both systems utilize similarly networked compute nodes that, in both references, utilize FPGA-based acceleration to perform similar processing tasks. *Id.*, ¶151.

4. Analysis

Element_[1pre]

Heil describes multiple server unit components (*compute nodes*) *for use in a data processing system* performing computations/data processing, each having a local host component paired with a local acceleration component. *Supra*, §V.C.1; AMD1010, [0040], [0053], [0089], [0218]–[0223], FIGS. 8, 36, 40; [0003], [0232], [0248]; AMD1003, ¶152.

In example Figure 36, each server unit component (acceleration component 3606 with S1, 3608/S2, 3610/S3, 3612/S4, 3614/S5, and 3616/S6) represents *a compute node*. AMD1010, [0218]–[0223] (also indicating that the S units are local host components), [0053] (“The host component in the server unit component is referred to as the ‘local’ host component”), [0074], [0084], [0089], [0229], FIGS. 8, 40; AMD1003, ¶153.

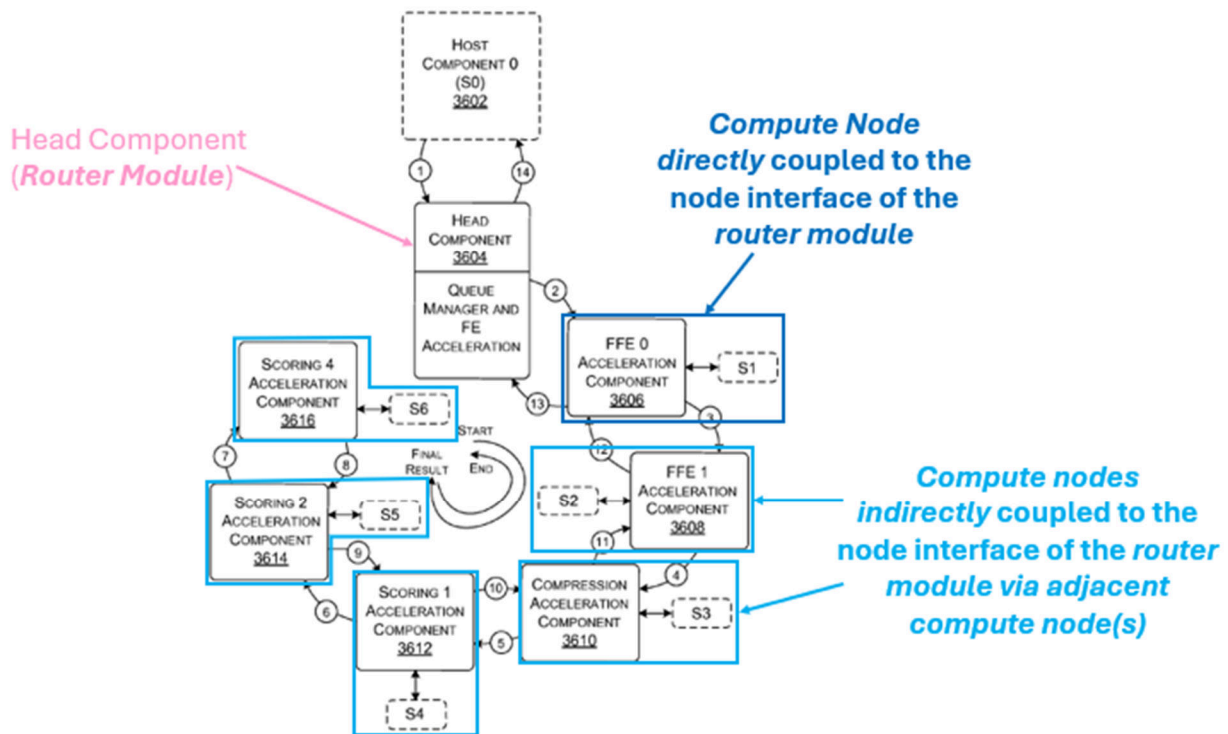


AMD1010, FIG. 36.

Element_[1a]

As discussed for Element_[1pre], Heil's server unit components each represent *a compute node*. Heil's host component 3602 (*host compute system*) invokes a service “by sending a request to a first acceleration component of the multi-component service.” AMD1010, [0219]–[0220], FIG. 36; *cf.* AMD1001, 9:17–46. Heil's first server unit component (*compute node*) includes an acceleration component 3606 that is *communicatively coupled* (1) to host component 3602 (*host compute system*) via head component 3604, and (2) to *adjacent* server unit components (*compute node[s]*) including acceleration component 3608 in the *execution pipeline*. *Id.*, FIG. 36, [0218]–[0223]. Each of Heil's acceleration

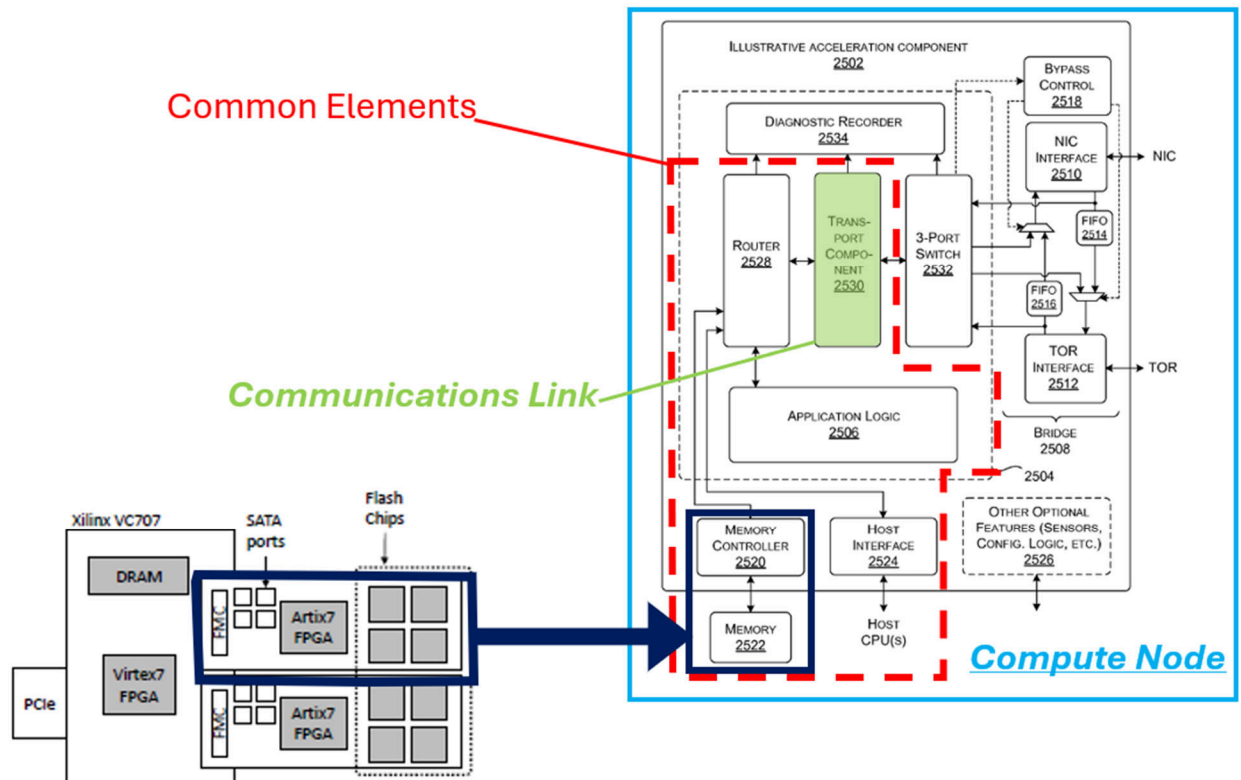
components are *communicatively coupled* to the neighboring acceleration components via dedicated links. AMD1010, [0089], [0221]–[0223], [0229], [0055], [0068], FIGS. 8, 36, 40; AMD1003, ¶¶154–155.



AMD1010, FIG. 36.

Heil discloses an example implementation of an acceleration component in Figure 25 (reproduced below). AMD1010, [0022], [0160]. As discussed at §V.C.1, a POSITA would have understood or found obvious that the acceleration component included in Heil’s downstream, pipelined server unit components (*compute nodes*) pictured in Figure 36 (above) would have included a transport component 2530 that is part of the example acceleration component shown in Figure 25. AMD1010, [0167]–[0168], [0181], FIG. 25; AMD1003, ¶156. The “transport component

2530...provide[s] a register interface to establish connections between nodes” and with other remote components, and thus functions as part of a **communications link** for communicating with adjacent acceleration components including the acceleration component 3608 of an **adjacent** server unit component (**compute nodes**) and to the **host** component 3602 (**host compute system**) (via the head component 3604). *Id.*; AMD1003, ¶156.



HJC Illustration.

In Heil, each acceleration component within a respective server unit component (**compute node**) is “**coupled** to east, west, north, and south neighboring acceleration components via appropriate cable links or the like” in a two-

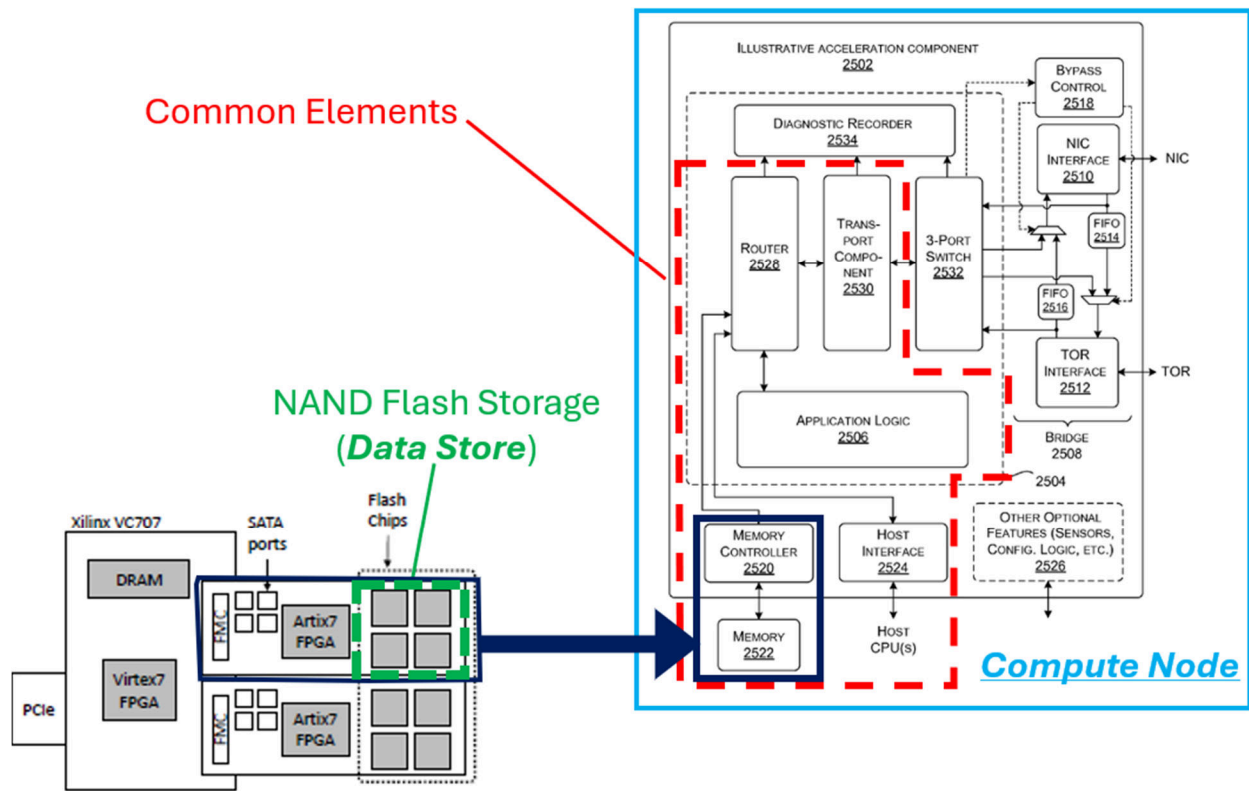
dimensional torus network. AMD1010, [0089] (“p×r torus network”), FIG. 8. Heil explains that these “dedicated links for connecting the acceleration components together” can have “any network topology” and that “[o]ther types of torus networks can alternatively be used having any respective sizes and dimensions.” *Id.*; AMD1003, ¶157.

Under Heil’s broad disclosure covering “any network topology,” it would have been obvious to a POSITA that Heil’s pipelined server unit components (*compute nodes*)—via their respective acceleration components—are *communicatively coupled to one or both of a host compute system and an adjacent compute node* in an *execution pipeline* for several reasons. AMD1003, ¶¶158–163. First, Heil’s Figure 36 illustrates a single dimension network topology where each node is connected only to two neighbors and the operations of the nodes are conducted serially, one after another. AMD1010, [0089]; AMD1003, ¶159. Second, Heil teaches using torus networks having “any” dimensions, and a 1-dimensional serial network topology (*e.g.*, ring with one node after another) would have been within this scope and obvious for implementing the service shown in Figure 36 (and other embodiments). *Id.*; AMD1010, [0089], [0084]; AMD1003, ¶160 (*citing* AMD1014, p.6 (demonstrating the example Figure 36 service being implemented on “one dimension of the torus.”)). Third, a 1-dimensional serial network topology would have direct functional alignment with the sequential pipeline and return path

as described with the example in Heil's Figure 36. AMD1010, [0089], [0218]–[0223], FIG. 36; AMD1003, ¶161. In fact, a 1-dimensional serial network would have been an efficient physical layout to support the example service shown in Figure 36. AMD1003, ¶161. Fourth, the POSITA would have recognized that a 1-dimensional serial network topology would have reduced the cabling complexity of the data processing system. *Id.* ¶162 (*citing* AMD1014, p.3 (demonstrating the “routability and cabling complexity” tradeoff)). Fifth, the POSITA would have appreciated that a 1-dimensional serial topology is one of a finite number of dimensional topologies for a torus network and its implementation with the example shown in Figure 36 would have resulted in a reasonable expectation of success. AMD1003, ¶163 (explaining the finite number of dimensional topologies and *citing* AMD1032).

Element [1b]

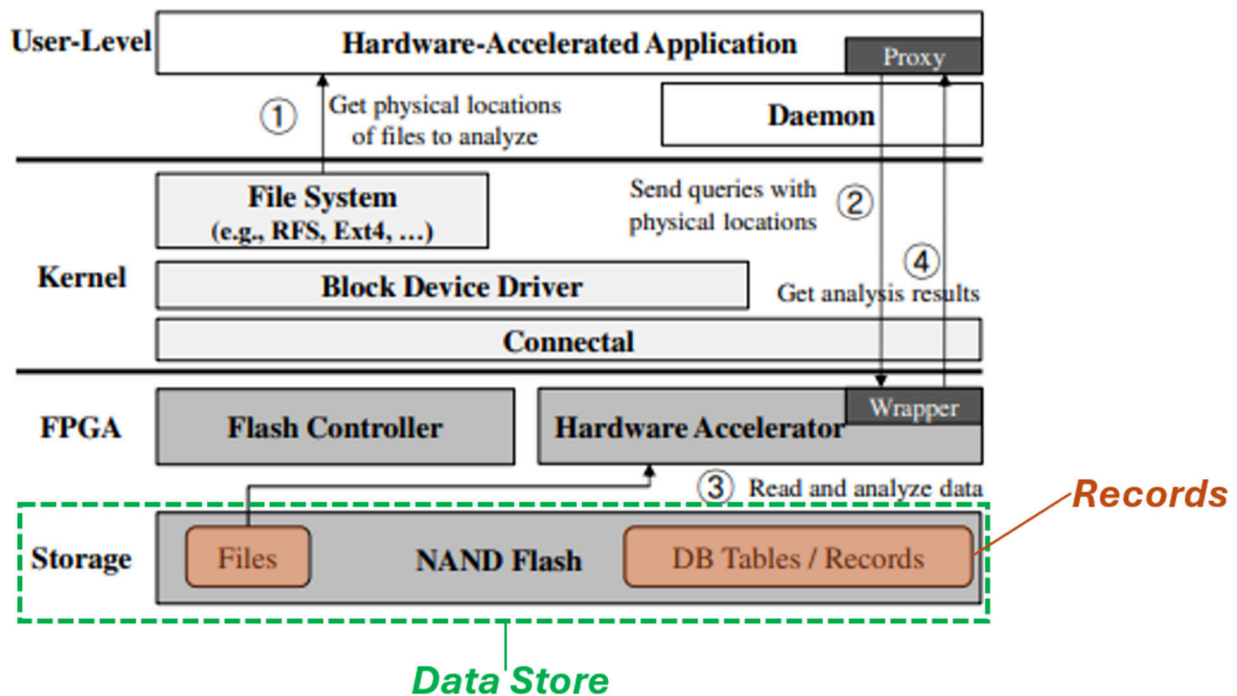
As discussed at §V.C.3, HJC integrates Jun's teachings to utilize Flash (***data store***) with Heil's acceleration components. AMD1003, ¶164.



HJC Illustration.

Jun provides teachings regarding the claimed *records associated with a data processing operation*. AMD1003, ¶¶164–165. Specifically, Jun’s Flash (*data store*) is shown as containing “Files” and “DB Tables/*Records*.” AMD1011, FIG. 8. Jun teaches that this information is read and analyzed by an FPGA-based hardware accelerator, like Heil’s acceleration components. AMD1011, 4, 7–8, 10, 12, FIG. 8. Also like Heil, the data stored in Jun’s Flash (*e.g.*, including the “Files” and “DB Tables/*Records*”) is associated with a *data processing operation*. AMD1011, pp.4 (“in-store processor to perform computation on the data”), 7 (“in-storage processors can directly read data from flash.”), 10 (“data is stored in flash

and the distance calculation is done by the in-store processor on the storage device.”), 12 (“...the accelerator, which uses these addresses to request for pages from the flash controller.”); AMD1001, 8:25–33 (“as is generally known...data [is] typically stored or maintained in files or datasets (or ‘records’ or ‘pages’)”).



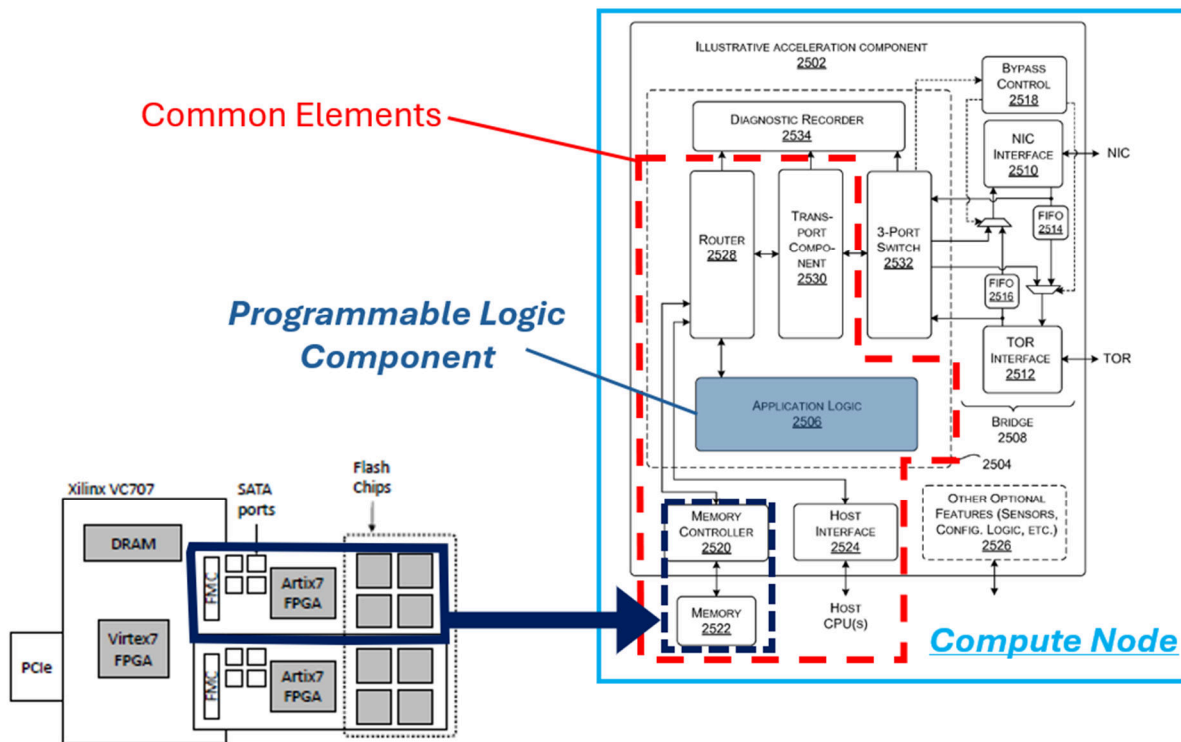
AMD1011, FIG. 8.

HJC incorporates Jun’s Flash (***data store***) with the acceleration component in each of Heil’s pipelined server unit components (***compute nodes***). *Supra*, §V.C.3. Moreover, a POSITA would have understood and found obvious to utilize Jun’s Flash for storing ***records associated with a data processing operation***, including, for example, one of the operations shown in the example of Heil’s Figure 36. AMD1010, [0218]–[0222], [0223], [0229], [0133]–[0140], [0157], [0161]–[0162],

[0178], [0248], FIGS. 36, 40; *supra*, Elements_[1pre]–[1a]; AMD1003, ¶¶166-168 (explaining that a POSITA would have also understood that incoming packets are stored in Jun’s Flash (as *records*) for data processing operations (including those discussed above) because Heil’s acceleration component depicted in Figure 25 includes a communication path between the router 2528 that receives the incoming data packets and the local memory 2522 (Jun’s Flash (*data store*) in the Combination) and *citing*: AMD1010, [0057], [0078], [0082], [0163]–[0164], [0167]–[0170], [0182]–[0193], [0218]–[0223], [0229], FIGS. 25, 36, 40; AMD1011; AMD1014).

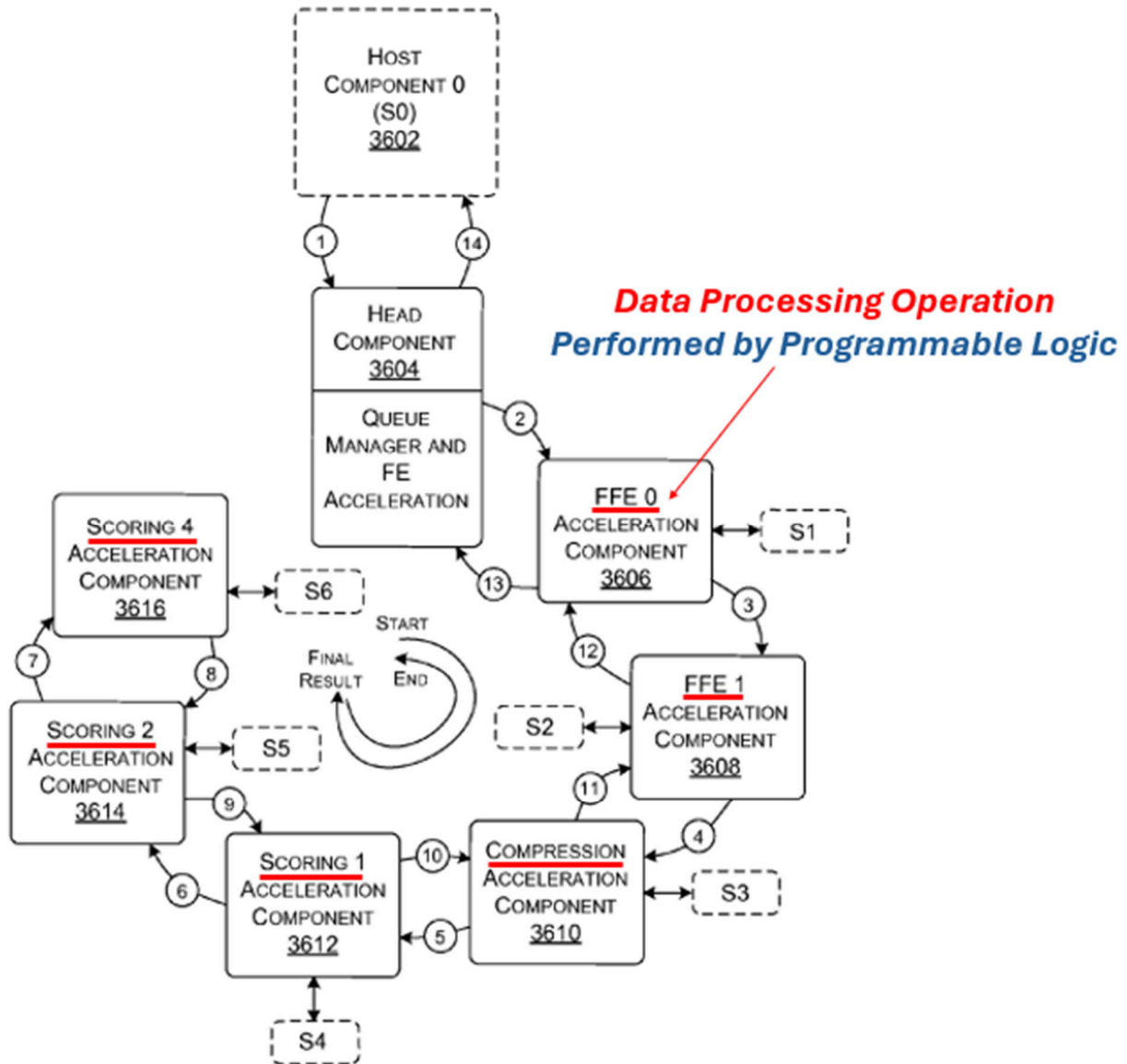
Element [1c]

Heil’s acceleration components within the pipelined server unit components (*compute nodes*) are implemented using an FPGA Device that includes reconfigurable application logic (*programmable logic component*). AMD1010, [0050], [0139] (“reconfiguring...the application logic”), [0157]–[0158] (“reconfiguration of the application logic”), [0160]–[0162], [0229], [0252], FIGS. 23–25; AMD1001, FIG. 4, 15:3–30, 37:61–67; AMD1003, ¶169. Because Heil’s application logic is implemented on an FPGA device, it is a logic component programmable by reconfiguring its internal hardware connections. *Id.*



HJC Illustration.

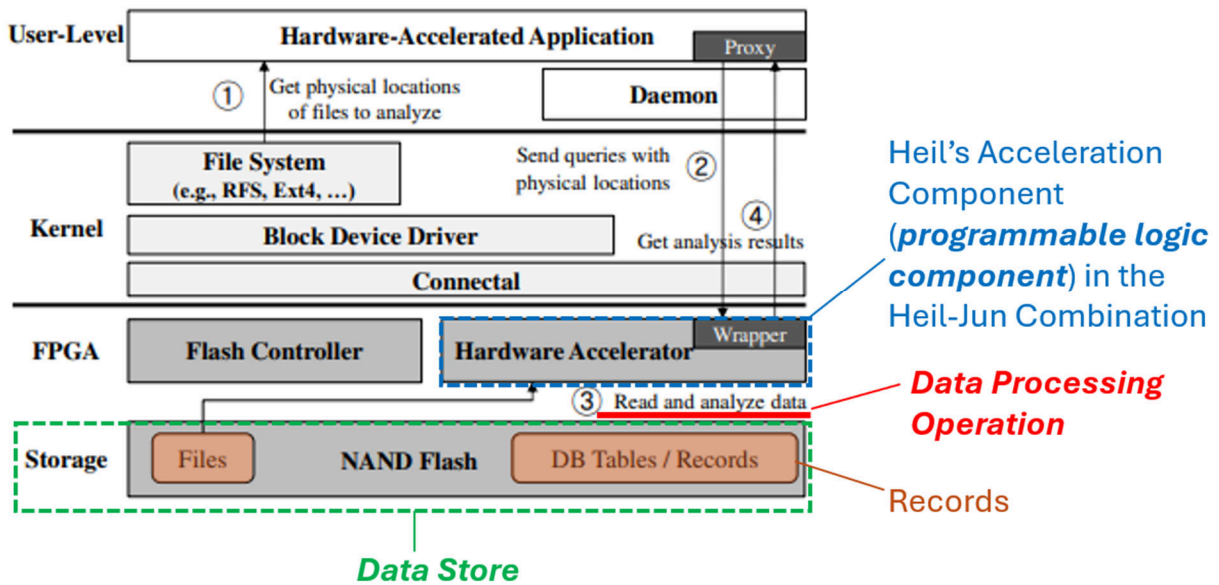
Heil teaches that each acceleration component, via its configured application logic, performs a respective data processing operation (*executes a data processing operation*). *Supra*, Elements_[1pre]–[1b]; AMD1010, [0222] (“perform more complex feature computations.”), [0223] (“generate a final ranking score”), [0229], FIGS. 36, 40; [0133]–[0140] (discussing “logic” on the acceleration component “for performing [a] service”), [0157], [0161]–[0162], [0178], [0248]; AMD1003, ¶170.



AMD1010, FIG. 36.

As discussed at Element_[1b], HJC discloses storing various *records* in Jun's Flash (*data store*) for performing the *data processing operation* assigned to the respective acceleration component of Heil's server unit components (*compute nodes*). Accordingly, HJC provides that the application logic (*programmable logic*

component) of each acceleration component operates *in cooperation with* Jun's Flash (*data store*) to execute its assigned *data processing operation*. AMD1003, ¶171.



AMD1011, FIG. 8.

Element [1d.i]

As discussed in §V.C.1 and for Element_[1pre], Heil's server unit component (*compute node*) includes a local host component and local acceleration component. As detailed below, Heil's local host component uses *data and instructions* stored in a main memory (*node memory*) to configure its companion local acceleration component, including the application logic (*programmable logic component*), for *supporting* the assigned *operation* and *facilitate communications* with the respective acceleration components of neighboring *nodes*. AMD1003, ¶172.

Heil's acceleration components include an application logic (*programmable logic component*) configured according to a configuration image/stream that is used to implement a service. AMD1010, [0155] ("Each configuration image contains logic...used to implement a corresponding service."), [0156]–[0157], [0124], FIGS. 21, 22; AMD1003, ¶173.

Heil also discloses that the local host component coordinates this configuration by "forwarding a configuration stream (corresponding to a configuration image) to the acceleration component." AMD1010, [0155]. A POSITA would have understood or found obvious that a configuration stream includes data defining the logic and instructions for configuring the logic. AMD1003, ¶¶174–175 (citing AMD1015 pp.2, 23, AMD1027 pp.8–10, 12, 38, 95, 118; AMD1046, 3:38–42 ("*instructions and data* (e.g., a configuration image)")); AMD1010, [0124], [0155]–[0158], [0171]–[0172]; *infra* Element_[11]. To transfer data to its counterpart local acceleration component, the local host component places the configuration image/stream (*data and instructions*) in the main (*node*) *memory*, so the *data and instructions* can be retrieved by the acceleration component using PCIe with DMA (direct memory access). AMD1003, ¶174; AMD1010, [0165], [0155], [0173]–[0175], FIGS. 25, 27, *infra* Element_[1g]. The local host utilizes the conventional approach of using main memory (*node memory*) to access data and instructions for performing its described functionalities including forwarding the

configuration image/stream. AMD1010, [0174], [0195] (identifying “RAM” as a storage resource for the host’s CPU to access “code, settings, data, etc.”), FIGS. 27, 31; AMD1003, ¶174.

Accordingly, it would have been obvious to a POSITA that each host component’s *node memory* includes *data and instructions* associated with deploying the configuration image *to support operation* of the *programmable logic component* (the acceleration component’s application logic) *in connection with the data processing operation*. AMD1003, ¶176; AMD1010, [0195], [0148]–[0152]; *supra*, Elements_[1b]–[1c] (describing example *data processing operations* performed by configured application logic including the example data processing operations shown in FIG. 36).

HJC renders obvious the *data and instructions to facilitate inter-node data communications via the communications link* at least because the acceleration component of each server unit component (*compute node*) receives “setup information” for establishing *inter node* connections. AMD1010, [0229], [0181] (“software process may set up the connections by statically allocating them before the transport component 2530 [(*communications link*)] can transmit or receive data”), [0248]. In Heil, the setup information received by the acceleration component is managed by a local management component implemented at the local host component within the same node. AMD1010, [0151] (“server unit component

stores its own dedicated local management component...implemented by a local host component”), [0100], [0148]–[0152], FIG. 10. Like for the configuration image, Heil’s host component transfers the setup information to its counterpart local acceleration component by placing the setup information—in the main (**node**) **memory**, so it can be retrieved using PCIe with DMA (direct memory access). AMD1003, ¶177; AMD1010, [0165], [0173]–[0175], FIG. 25, *infra* Element_[1e]. Accordingly, as would have been obvious to a POSITA, Heil’s software-implemented setup information (***data and instructions to facilitate inter-node data communications via the communications link***) is placed into the host component’s main (**node**) **memory** at least for facilitating transfer to the acceleration component. *Id.*; AMD1010, [0181], [0229], [0248]; AMD1003, ¶177. A POSITA would have also understood that implementing the local management component at the host component includes using ***data and instructions*** stored at the host component’s main (**node**) **memory**. AMD1010, [0174], [0195], FIGS. 27, 31; AMD1003, ¶177.

HJC also renders ***facilitating inter-node data communications*** obvious via the ***data and instructions*** associated with the configuration image in Heil’s local host main memory (**node memory**). For example, Heil’s acceleration component processes and flow-regulates data for transfer to the next pipelined node via the transport component (***communications link***). AMD1003, ¶178; *infra* Element_[11]; AMD1010, [0167]–[0168], [0191] (“the application logic 2506...regulate[s] the

flow of packets through the use of ‘back pressuring’”), [0218]–[0223], [0229]; FIGS. 25, 36, 40. This serialized *inter-node* communication would not have occurred effectively, unless each pipelined acceleration unit is appropriately configured. *Id.*; AMD1003, ¶178.

Element [1d.ii]

As discussed at Element_[1d.i], each of Heil’s pipelined server unit components (*compute nodes*) includes a main memory (*node memory*) for its local host component. Heil’s main memory includes “a kernel-pinned input buffer 2708 in main [*node*] *memory*.” AMD1010, [0173]–[0175], FIG. 27. Heil’s main memory (*node memory*) also includes output buffer 2716 that receives an output result from the acceleration component executing a data processing operation. *Id. cf.* AMD1010, 53:8:21 (explaining the ’636 patent’s algorithm memory can be used for “buffering a final intermediate result”). Like the buffers in the algorithm memory described in the ’636 patent, Heil’s kernel-pinned buffers (e.g., input buffer 2708, output buffer 2716) correspond to dedicated regions of memory that include *resource-specific memory locations*. AMD1010, [0173]–[0175], FIG. 27; *cf.* AMD1001, 40:25-41 (explaining that algorithm memory comprises resource-specific memory locations such as algorithm buffers and FP buffers). A POSITA would have also found this obvious at least because pinned memory spaces are specifically mapped to physical memory locations of a specific memory resource to

provide data transfer without CPU (e.g., Host Component) intervention. AMD1003, ¶179 (*citing* AMD1041, p.23 (defining pinned memory as “[m]emory pages that are to be maintained in real memory all the time” and explaining that a memory page must typically be pinned before DMA starts)); AMD1048, [0010]–[0012].

The kernel-pinned input buffer is an *algorithm memory* at least because it operates *to support the data processing operation* for the service, for example, by communicating instructions and data for a configuration image and setup information as discussed in Element_[1d.i]. AMD1003, ¶180; AMD1010, [0127], [0216] (“the multi-component service corresponds to an...*algorithm*”), [0218]–[0223]; *infra* Element_[1h]. Heil also discloses an example of utilizing the input/output buffers for a data processing operation to generate an output result. AMD1010, [0173]–[0175], FIG. 27; AMD1001, 18:50-65; 40:25-41 (“as is generally known”), 53:8:21; AMD1003, ¶180.

Element_[1e]

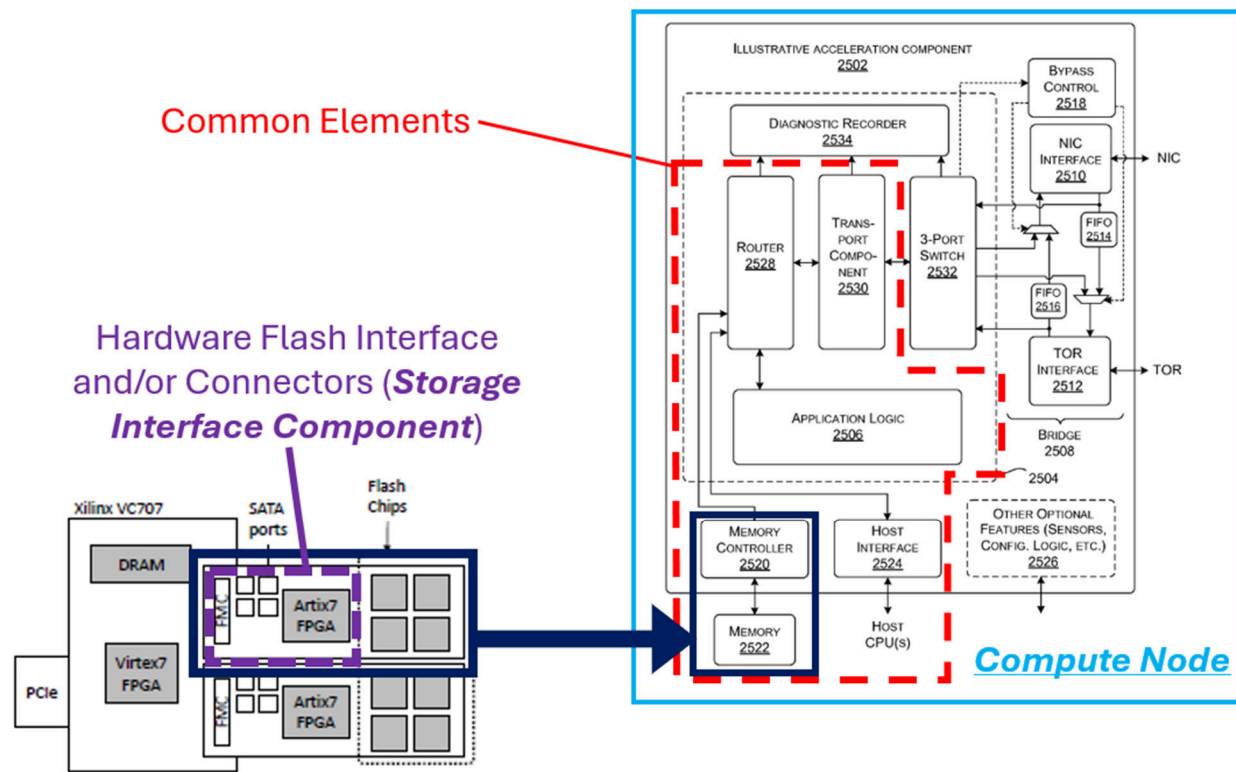
As discussed at Elements_[1d.i]–[1d.ii], Heil’s pipelined server unit components (*compute nodes*) include a main memory (*node memory*) for its local host component. Heil further discloses that each local host component may communicate with its local acceleration component “using PCIe in conjunction with DMA [(direct memory access)] transfer.” AMD1010, [0173]–[0175], [0165], [0089] (“the data processing system 802 includes...a local_H-to-local_S coupling 114

for connecting local host components to respective local acceleration components.”), FIG. 27. The components used for PCIe/DMA operate as a *data mover component to facilitate* data transfer within the server unit component (*intra-node data communications*) *between* the local host main memory (*node memory*) and the acceleration component’s application logic (*programmable logic component*). AMD1010, [0173]–[0175] (explaining PCIe/DMA communication from buffers from the local host main memory to the local acceleration component’s application logic), [0085], [0089], [0124], [0155]–[0156], FIGS. 6, 8, 22, 25, 27; *supra*, Elements_[1d.i]–[1d.ii]; *infra* Elements_[7]–[8]; AMD1003, ¶¶181–183 (*citing* AMD1014); AMD1001, 16:3-10.

Element_[1f]

As discussed, HJC incorporates Jun’s Flash (*data store*) to serve as the local memory for Heil’s acceleration component. *Supra*, §§V.C.2–3; Element_[1b]. The Flash is connected to a processor (*e.g.*, “Virtex 7 FPGA”) using a “FPGA Mezzanine Card (FMC) connector.” AMD1011, p.8. “The *communication between* the flash board and the Virtex 7 FPGA [processor] is done by a 4-lane aurora channel” and “SATA connectors.” *Id.* Moreover, Jun provides that “[t]he flash controller and Error Correcting Code (ECC) is implemented on [an] Artix chip” that “exposes a...hardware *interface*” to the Flash. AMD1011, pp.8, 4, FIGS. 2–4, 10; *infra* Element_[1g] (demonstrating the multiple channels to NAND Flash Chips).

Individually or collectively, the hardware interface and connectors on Jun's flash board function as a *storage interface component*. AMD1003, ¶184.



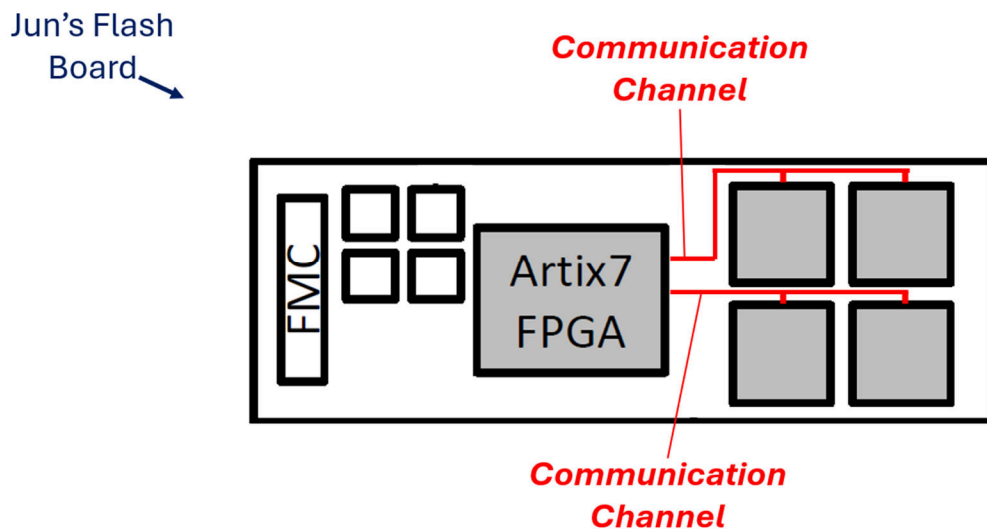
HJC Illustration.

In HJC, Heil's acceleration components incorporate Jun's flash board, which a POSITA would have understood, included a *storage interface component to facilitate intra-node data communication between Jun's Flash (data store) and the application logic (programmable logic component)*. See §V.C.3, Elements_[1b]–[1c], [1e]; AMD1003, ¶185; AMD1001, 11:19–21 (“data store interface 145 manages access...to data store 143, as is generally known in the art.”), 16:18–17:48; 39:65–40:8.

Element [1g]

As discussed, HJC incorporates Jun’s teachings of a flash board including Flash (***data store***) in conjunction with suitable connectors and hardware interface (***storage interface component***). *Supra*, §§V.C.2–3, Elements_[1b]–[1c], [1f]; AMD1003, ¶186.

Jun discloses and a POSITA would have found obvious that the flash board’s Artix-7 FPGA manages data transfer from the Flash utilizing ***multiple parallel communication channels*** to access NAND Flash Chips. AMD1011, FIG. 10 (below), pp.4 (“Flash devices...achieve high bandwidth by grouping multiple flash chips into several ***channels***...”), 6 (“data reads from flash chips on multiple buses in parallel”) 9; AMD1003, ¶¶187–188 (*citing* AMD1012). In HJC, Jun’s flash interface provides high-bandwidth Flash access to the application logic. *Id.*



AMD1011, FIG. 10 (modified).

HJC also renders Element_[1g] obvious via Jun's disclosure that "[t]he communication between the flash board and the Virtex 7 FPGA processor is done by a 4-lane aurora *channel*" and "SATA connectors." AMD1011, p.8, FIGS. 2 ("Flash *interface*"), 10. In HJC, Jun's 4-lane aurora *channel* and/or SATA connectors are *communication channels to transfer data* from Flash (*data store*) to application logic (*programmable logic component*) of Heil's acceleration component. AMD1003, ¶189; *supra*, §§V.C.2–3.

Element_[1h]

As discussed for Elements_[1b]–[1c], HJC's programmable logic component executes a data processing operation. In Figure 36, the multi-component service includes a pipeline utilizing multiple server unit components (*compute nodes*) with specialized acceleration/FPGA-based processing. AMD1010, [0218]–[0213], [0040] FIG. 36. Heil explains that this service "[t]raditionally...is performed entirely in software by applying a model produced by machine-learning." AMD1010, [0218], [0220]–[0223], [0216], [0127] ("search *algorithm*"). Heil further explains that "acceleration components (3606, 3608) perform more complex feature computations" (e.g., *the data processing operation*) that "involve mathematically combining feature values." AMD1010, [0222], [0224]. Accordingly, each of Heil's server unit components (*compute nodes*) executes an assigned data processing operation that *is associated with a computationally*

expensive (e.g., machine-learning based) *algorithm*. AMD1003, ¶190; AMD1010, [0229], FIG. 40. Notably, during prosecution of the application that led to the '636 patent, Applicant conceded that an example of “computationally expensive algorithms” includes “Big Data or artificial intelligence applications.” AMD1002, 163; *supra* §II.B; AMD1003, ¶190.

Heil teaches executing a multi-component service using a collection of distributed acceleration components (e.g., *deployed in a distributed processing compute environment to execute the computationally expensive algorithm*). AMD1010, [0218], FIGS. 8, 36; [0040], [0065]–[0069], [0071]–[0075]; [0083], [0144], [0151], [0198]–[0210], [0225]; AMD1003, ¶191.

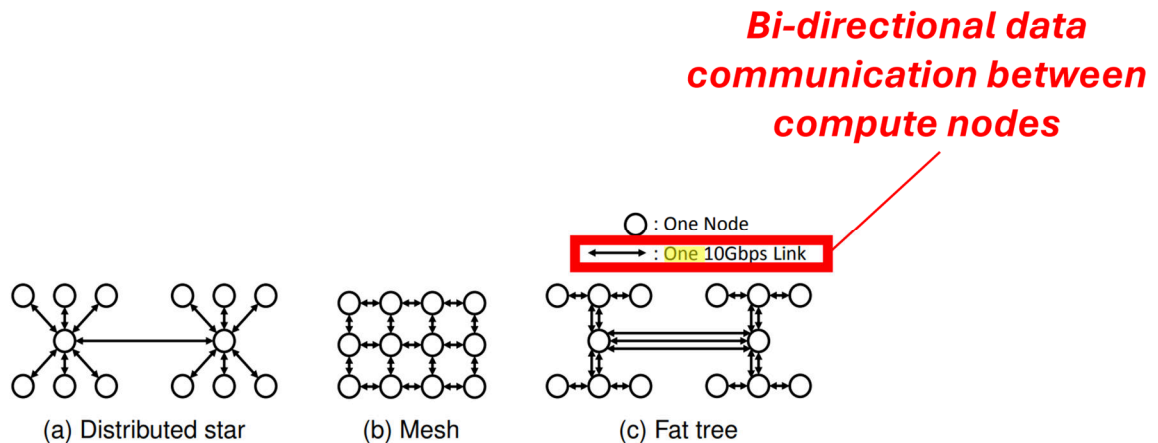
Element [2]

As discussed at Elements_[1pre]–[1a], Heil’s server unit component (*compute node*) includes an acceleration component with a transport component (*communication link*) for communicating with acceleration components of the neighboring server unit components. This inter-node communication takes place via dedicated “cable links.” AMD1010, [0089]. As a POSITA would have understood or found obvious that acceleration components communicating via “cable links” in a distributed computing environment routinely employ *serial communication protocols* to support high-speed *communication*. AMD1010, [0184] (describing an example employing a serial protocol (Ethernet)); AMD1003, ¶¶192–193 (*citing*

AMD1047, p.310 (“Ethernet is a serial protocol”); AMD1014, 4 (showing Heil’s torus network example with “[f]our high-speed *serial* links running *Serial*Lite III (SL3), a lightweight *protocol* for communicating with neighboring FPGAs”); AMD1011, p.4 (“The FPGAs form a network...via high-speed serial links...”), 9. A POSITA would have also recognized that this communication is bi-directional at least because Heil’s torus network architecture dictates *bi-directional communication* to function. AMD1010, [0180]–[0186] (describing sending and receiving packets and a failure handling component that “buffers all sent packets until it receives an acknowledgement (ACK) from the receiving node”), [0218]–[0223], FIG. 36 (demonstrating the processing operations flow in a first direction and the results are routed back to the requesting host in the opposite direction over the same network infrastructure); *supra* 1A; AMD1003, ¶194 (citing AMD1014, p.3 (“each inter-FPGA network link supports 20 Gb/s of peak *bidirectional* bandwidth.”)).

HJC also renders Element_[2] obvious via Jun’s teachings for using *bi-directional* “high-performance *serial* links” to form a network among FPGA-based devices. AMD1011, pp.5, 8 (“the link is implemented over the low-latency *serial* transceivers”), 9 (“each node can have an aggregate full duplex bandwidth”), FIGS. 5 (below showing *bi-directional data communication* between nodes), 9 (*Serial* Advanced Technology Attachment (“SATA”) cables); AMD1003, ¶195 (citing

AMD1001, 9:47–10:40 (explaining that communication between the host and router is *bi-directional* and listing SATA as an example for providing such communication). A POSITA would have understood that a serial protocol would be employed in a system using high-performance *serial* links with *serial* transceivers. AMD1003, ¶196 (explaining that a *serial protocol* (e.g., Serial ATA protocols) would have been used to transmit data over SATA cables and *citing corroborating* AMD1033). Appreciating the known low-latency benefits of high-speed data transfer, a POSITA would have been motivated to employ links using a *serial communication protocol*, as Jun did, *for enabling bi-directional communication between the* acceleration components in the Heil’s pipelined server unit components *(compute nodes) in the execution pipeline. Id.*



AMD1011, FIG. 5.

Element_[3]

As discussed for Element_[1a], Heil teaches the server unit component (*compute node*) *communicatively coupled to the adjacent* server unit component (*compute node*) *in an execution pipeline*. The server unit component is *communicatively coupled* to the *adjacent* server unit component *in series* via a transport component (*communications link*). AMD1003, ¶¶197–198; AMD1010, [0068], [0074], [0084], [0089], [0218]–[0223], FIG. 36.

Element_[4]

HJC incorporates Jun’s Flash (*data store*) for the server unit component (*compute node*). *Supra*, §§V.C.2–3, Elements_[1b], [1f]–[1g]. It was known in the art and a POSITA would have recognized that a Flash is an *EEPROM device*. AMD1001, 10:65–11:21 (“all Flash is EEPROM”); AMD1003, ¶199 (*citing* AMD1034).

Element_[6]

Supra, Element_[1c]. AMD1003, ¶¶200–201 (*citing* AMD1014).

Element_[7]

As discussed for Element_[1e], Heil’s server unit components (*compute nodes*) utilizes PCIe/DMA elements (*data mover component*) to move data from the main memory of the host component to the acceleration component. AMD1010, [0165]. In particular, Heil’s local host component 2702 “forward[s] information to

its local acceleration component 2704” using host logic. AMD1010, [0173]–[0175], FIG. 27. The “host logic 2706 places data to be processed into a kernel-pinned input buffer 2708 in main memory associated with the host logic 2706” and “instructs the acceleration component 2704 to retrieve the data.” *Id.* Heil’s host logic is implemented on a *processor* that instructs (*e.g., manages operation of*) *the data mover component* (components for PCIe with DMA). *Id.*, [0195] (“host component 3102 can include one or more processing devices 3104...which may implement one or more hardware threads.”), [0077], [0048], [0051], [0054], [0077], [0133], [0248]; AMD1003, ¶¶202–203 (*citing* AMD1014).

Element [8]

As discussed for Elements_[1e], [7], Heil’s *data mover component* at the server unit component (*compute node*) utilizes PCIe/DMA elements (*data mover*) to facilitate communications between a local host and a local acceleration component. As would have been obvious to a POSITA, PCIe/DMA functionality routinely involves a *device controller* to help manage the transfer process. *Supra* Elements_[1g], [7]; AMD1003, ¶204 (*citing* AMD1014, FIG. 3 (“DMA Engine”); AMD1039, pp.3–5, FIGS. 3–5 (demonstrating a “*device controller*” for PCIe DMA); AMD1010, [0165], [0173]–[0175], FIG. 27. A POSITA would have appreciated that a DMA *controller* would have been used in Heil’s system to receive

instructions and notify the host logic for providing received data to the host component's main memory. *Id.*

HJC also renders Element_[8] obvious via Jun's teachings for a DMA Engine (*device controller*) as part of a PCIe-based host interface, and it would have been obvious to integrate this teaching with Heil to facilitate data transfer from the local host's main memory using PCIe. AMD1011, pp.4–6, 6, 9, FIGS. 7, 9–10; *cf.* AMD1001, 15:59–16:10; AMD1003, ¶205.

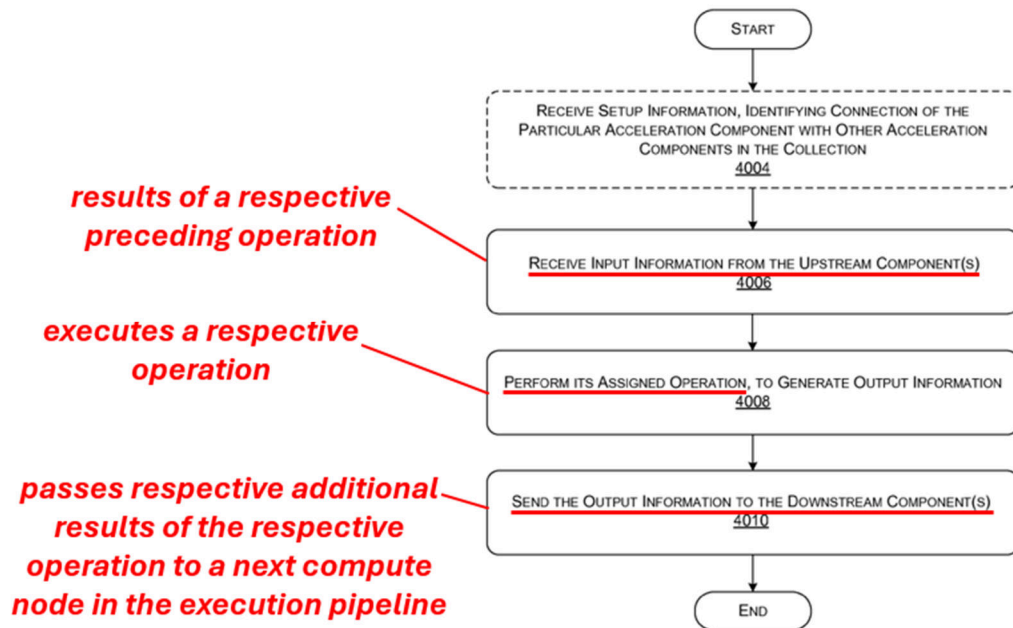
Element_[9]

As discussed for Elements_[1pre]–[1a], the server unit component's (*compute node*) transport component (*communications link*) *enables* the server unit component to receive a request from the host component (*host compute system*) (e.g., via a head component) to invoke the service . AMD1010, [0167]–[0168], [0181], [0218]–[0223], [0239], FIGS. 36, 40; *supra*, §V.C.1. In the FIG. 36 example, this request includes “various items of information, such as one or more hit vectors which describe the locations of the query terms within the document under consideration” and “also specif[ies] a collection of software-generated feature values.” AMD1010, [0220], [0056], [0070]–[0071], [0225]–[0227], [0240]–[0241], [0248]. A POSITA would have understood that Heil's request invoking a service includes instructions that would have been received by the server unit component (via the head component) in order to utilize the relevant items of information in the

request. *Id.*; AMD1003, ¶206. Thus, the *instructions* included in the request *influence the operation of the* server unit component (*compute node*). AMD1010, [0219]–[220], [0229], FIGS. 36, 40.

Element [11]

As discussed, Heil teaches pipelined server unit components (*compute nodes*) that each include an acceleration unit with application logic that executes *a respective operation*. *Supra* Elements_[1a], [1c]; AMD1010, [0229], FIG. 40. Specifically, Heil’s “acceleration component receives input information from the upstream component(s)...performs its assigned *operation* to generate output information...[and] sends the output information to the downstream component(s).” AMD1010, [0229], FIG. 40; AMD1003, ¶207.



AMD1010, FIG. 40.

As discussed at Element_[1h], the operation assigned to the server unit component (*compute node*) is *associated with the computationally expensive algorithm*. AMD1010, [0216]–[0218]; *supra*, Element_[1h]. And Heil discloses a process for configuring an acceleration component to execute an operation for a service using a “configuration stream/image.” *Id.*, [0124], [0155]. The configuration stream/image specifies the logic to be “programmed” into a recipient acceleration component and incorporates *computer-executable code associated with the* service (the *computationally expensive algorithm*). AMD1010, [0124], [0155], [0049], [0157], [0161]–[0162]; AMD1003, ¶¶208–209 (*citing* AMD1027, 39); AMD1001, 5:10–6:34, 19:40–20:15, 35:28–43, 36:45–55, 44:59–45:25 (confirming that the *computer-executable code* in the ’636 patent is incorporated in a bit stream for FPGA configuration); *infra* Element_[12].

Element_[12]

As discussed for Element_[11], Heil’s *computer-executable code associated with the computationally expensive algorithm* includes a configuration stream/image. Heil’s configuration stream for an FPGA Device is *incorporated in a bit stream*. AMD1010, [0124], [0155]; AMD1003, ¶¶210–211; AMD1014, pp.2 (“bitstream reconfiguration”), 4–5. A POSITA would have understood or found obvious that a configuration *stream/image* in the context of FPGA devices refers to

a *bit stream*. AMD1003, ¶210 (*citing* AMD1015, p.2 (“a binary file called a configuration file or bitstream”), AMD1016, Abstract, AMD1017 p.3.

D. GROUND 2B: Heil-Jun-Raghava Renders Obvious Claims 5, 10

1. Heil-Jun-Raghava Combination

Heil, Jun, and Raghava qualify as analogous art to the ’636 patent because they share the ’636 patent’s field of endeavor in data processing systems employing reconfigurable hardware. AMD1003, ¶212; AMD1006, Abstract, [0004], [0015]–[0017], [0040]–[0049]; *supra*, §§V.B.1, V.C.1-3. They are also reasonably pertinent to a technical problem shared with the ’636 patent of accelerating processing by describing solutions that offload tasks to reconfigurable hardware. *Id.*

(a) Raghava’s ONFI

As discussed, HJC incorporates Jun’s NAND-based Flash in conjunction with suitable connectors and hardware interface (*storage interface*) for the pipelined server unit components (*compute nodes*). *Supra*, §§V.C.3; V.C.4, Elements_[1b], [1f]–[1g]. A POSITA would have implemented a standard protocol like **ONFI**, as taught by Raghava, at *each* server unit component (*compute node*). AMD1003, ¶¶213–214 (*citing* AMD1012 showing a BlueDBM Node (the same as Jun’s) with an “**ONFI**-compliant FPGA-based flash controller.”); *Supra* §§V.B.1–2, V.B.3, Elements_[4]–[5] (describing Raghava’s teachings for the ONFI protocol);

AMD1006, [0033], [0046]. A POSITA would have employed Raghava's **ONFI protocol** to support communication between Jun's NAND Flash and Heil's acceleration component and achieve known benefits including improved transfer rate, power consumption, and/or standardized interoperability. AMD1003, ¶¶215–216 (*citing* AMD1035, AMD1012).

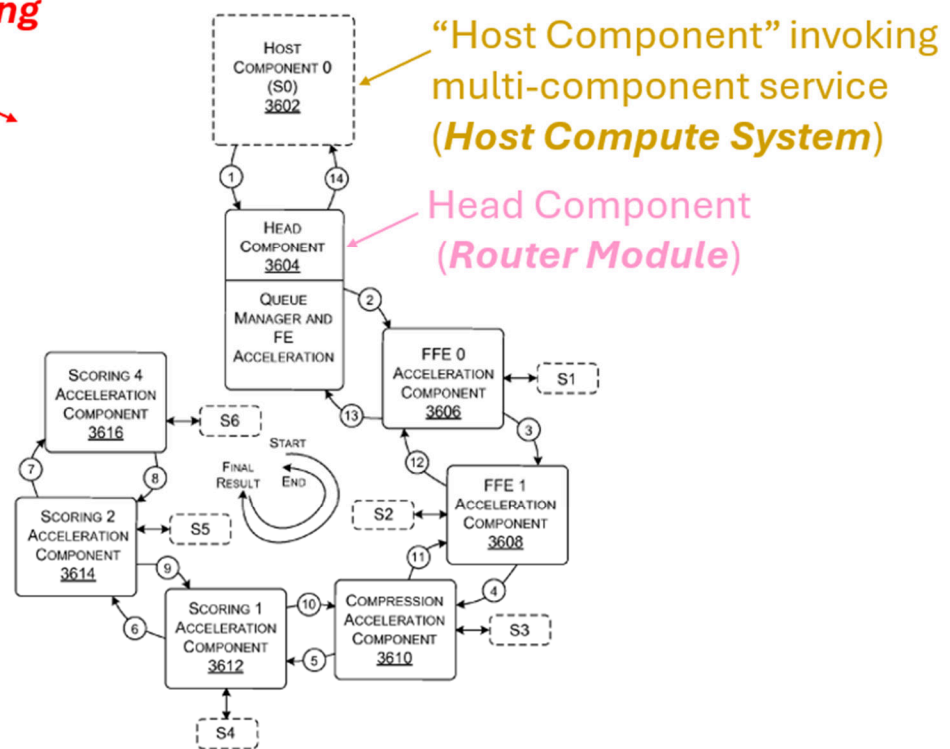
Additionally, the combination of HJC's server unit components (**compute node**) including NAND Flash with Raghava's **ONFI** teachings for accessing the same type of storage interface would have been obvious because it represents the predictable application of a known protocol to achieve the expected results for communicating with NAND Flash. AMD1003, ¶217 (*citing* AMD1012). A POSITA would have an expectation of success for implementing a standard **protocol** like Raghava's **ONFI** on the known components (Jun's **storage interface**) to achieve an expected result of enabling access to NAND Flash at least because ONFI is a standard protocol that is specifically designed for interoperability with different hardware components to facilitate communication and in particular with functionally compatible devices like Jun's storage interface and NAND Flash. *Id.*

(b) Raghava's Interface Card

It would have been obvious to a POSITA that Heil's head component corresponds to a **routing module** at least because the head component **routes** communications between the requesting host and the server unit components

(including the **compute node**) implementing the service. *Supra* §V.C.4, Elements_[1a]–[1pre]; AMD1003, ¶¶218–219; AMD1010, [0068], [0219]–[0223], [0236], [0251], [0167]–[0168], [0176]–[0186], FIGS. 25, 27–28, 36, [0074], [0089], [0095], [0203]. Like the **router module** described in the '636 patent, Heil's head component is FPGA-based and facilitates communication between the requesting host and server unit components, including by controlling the flow/queue of requests/commands. AMD1010, [0048]–[0050], [0160], [0225]–[0227], [0252]; AMD1001 9:47–10:24; *Supra* §V.C.4, Elements_[1a]–[1pre].

**Data Processing
System**



AMD1010, FIG. 36.

Heil discloses a host component and remote acceleration components “implemented on a single physical component.” AMD1010, [0084]–[0090] (“implemented on a single die”). Consistent with this disclosure, a POSITA would have implemented a single component *combining* Heil’s first server unit component (*compute node*) (e.g., 3606 with S1 in Figure 36) and remote head/acceleration component (*router module*) (e.g., 3604 in Figure 36). *Supra*, §§V.C.1, V.C.4, Elements_[1a]–[1b]. Indeed, Raghava demonstrates that integrating this arrangement into a single accelerator DIMM card (*interface card*) was well known. AMD1006, [0040]–[0045], FIG. 5; *supra*, §§V.B.1, V.C.4, Element_[1a]. Specifically, Raghava’s *interface card* is coupled to a host CPU and uses a memory controller and request decoding/routing circuit to route host requests to FPGA-based hardware acceleration units. AMD1006, [0040]–[0045], FIG. 5; AMD1003, ¶220.

It would have been obvious to combine and integrate Heil’s head component (*router module*) and the first server unit component (*compute node*) in an *interface card communicatively coupled to the host compute system*, based on Heil’s teaching to incorporate remote acceleration components (such as the head component/*router module*) with a server unit component (*compute node*) on a single component with Raghava’s teachings for a router and acceleration unit arrangement on an interface card coupled to the host. AMD1003, ¶221. The following reasons, taken separately or collectively, support such a Heil-Jun-Raghava Combination.

First, a POSITA would have been motivated based on Raghava's teachings that the accelerator DIMM *card*, including both *router module* and acceleration-based *compute node*, further supports Heil's goals to orchestrate multi-component services by offloading computationally intensive tasks to parallel computational resources. AMD1003, ¶222; AMD1006, Abstract, [0004], [0015] ("Offloading computationally intensive functions onto the accelerator DIMM card is oftentimes more efficient...due to the highly parallel nature of the programmable device."), [0017]; [0040]–[0044], [0048]–[0049]; AMD1010, [0051], [0224] ("leverage parallel computational resources").

Second, a POSITA would have understood that integrating Heil's head component (*router module*) and server unit component (*compute node*) within a single component as Raghava's accelerator DIMM (*interface*) *card* would have reduced latency in Heil's data processing system. AMD1003, ¶223. For example, as taught by Raghava, such an arrangement would have allowed the host component to initiate a service and execute Heil's pipeline with less latency. *Id.*; AMD1006, [0036]–[0037], [0054]–[0055].

Third, a POSITA would have been motivated to implement such a combination based on Raghava's teachings that "the inclusion of acceleration circuitry in the DIMM card/slot improves the energy efficiency of applications." AMD1006, [0053]; AMD1003, ¶224.

Fourth, combining HJC's head component (*router module*) and server unit component (*compute node*), each having (FPGA-based) acceleration, with Raghava's accelerator DIMM cards also having FPGA-based acceleration would have been obvious because it represents the predictable application of a known multi-stage architecture to a high-performance interface to achieve the expected results of reduced latency for offloading complex processing to specialized/FPGA-based resources. AMD1003, ¶225. A POSITA would have an expectation of success for implementing known components (*e.g.*, Heil's head component (*router module*) and server unit component (*compute node*)) in a known arrangement on a known interface card as taught by Raghava to execute complex processing operations at FPGA-based acceleration components. AMD1003, ¶225 (*citing* AMD1014).

2. Analysis

Element_[5]

The Combination discussed in §V.D.1.(a) satisfies Element_[5]. AMD1003, ¶226. *Supra* §§V.B.1, V.B.3, Elements_[4]–[5].

Element_[10]

The Combination discussed in §V.D.1.(b) satisfies Element_[10]. AMD1003, ¶227.

VI. PAYMENT OF FEES

Petitioner authorizes the Patent and Trademark Office to charge any fees to Deposit Account No. 06-1050.

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

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

Petitioner, Advanced Micro Devices, Inc. is the real party-in-interest.

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

Petitioner is not aware of any disclaimers, reexamination certificates or petitions for *inter partes* review for the '636 patent. The '636 patent is the subject of civil action: *Fermat International, Inc. v. Advanced Micro Devices*, 7:26-cv-00011 (WDTX), filed January 13, 2026.

C. Lead And Backup Counsel—37 C.F.R. §42.8(b)(3)

Petitioner provides the following designation of counsel.

Lead Counsel	Backup counsel
Kenneth W. Darby, Jr., Reg. No. 65,068 Fish & Richardson P.C. 60 South Sixth Street, Suite 3200 Minneapolis, MN 55402 Tel: 202-783-5070 Fax: 877-769-7945 Email: IPR54788-0026IP1@fr.com	David M. Hoffman, Reg. No. 54,174 Usman A. Khan, Reg. No. 70,439 Cameron A. Ubel, Reg. No. 77,081 Joseph W. Bauer, Reg. No. 81,218 Fish & Richardson P.C. 60 South Sixth Street, Suite 3200 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 IPR54788-0026IP1@fr.com
(referencing No. 54788-0026IP1).

Respectfully submitted,

Dated May 12, 2026

/Usman Khan/
Kenneth W. Darby, Jr., Reg. No. 65,068
David M. Hoffman, Reg. No. 54,174
Usman A. Khan, Reg. No. 70,439
Cameron A. Ubel, Reg. No. 77,081
Joseph W. Bauer, Reg. No. 81,218
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
T: 202-783-5070
F: 877-769-7945

(Control No. IPR2026-00360)

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 13,938 words, which is less than the 14,000 allowed under 37 CFR § 42.24.

Dated May 12, 2026

/Usman Khan/
Kenneth W. Darby, Jr., Reg. No. 65,068
David M. Hoffman, Reg. No. 54,174
Usman A. Khan, Reg. No. 70,439
Cameron A. Ubel, Reg. No. 77,081
Joseph W. Bauer, Reg. No. 81,218
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
T: 202-783-5070
F: 877-769-7945

Attorneys 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 May 12, 2026, a complete and entire copy of this Petition for *Inter partes* Review, Power of Attorney, and all supporting exhibits were provided by Federal Express, to the Patent Owner by serving the correspondence address of record as follows:

Ethos Legal PC
30 Gruver Road
Pipersville, PA 18947

/Diana Bradley/
Diana Bradley
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
bradley@fr.com