

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

DECLARATION OF DR. DONALD ALPERT

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	QUALIFICATIONS	1
III.	LEGAL PRINCIPLES	4
	A. Person of Ordinary Skill in the Art.....	4
	B. Obviousness.....	4
	C. Claim Interpretation	6
IV.	MATERIALS CONSIDERED	7
V.	THE '636 PATENT	10
	A. Pertinent Disclosure	10
	B. Prosecution History.....	12
	C. Interpretation of the '636 Patent Claims at Issue	12
VI.	PERSON OF ORDINARY SKILL IN THE ART.....	13
	A. A POSITA's Ordinary Level of Skill	13
VII.	OVERVIEW OF CONCLUSIONS FORMED	13
VIII.	GROUND 1A: DALAL AND NG	14
	A. Dalal (AMD1004).....	14
	B. Ng (AMD1005)	19
	C. The Dalal-Ng Combination.....	21
	D. Claim Element Analysis.....	28
	1. Claim 1	28
	(i) <i>Element [1pre]: A compute node for use in a data processing system, the compute node comprising:</i>	<i>28</i>

(ii)	<i>Element [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;</i>	<i>29</i>
(iii)	<i>Element [1b]: a data store comprising records associated with a data processing operation;</i>	<i>32</i>
(iv)	<i>Element [1c]: a programmable logic component to execute the data processing operation in cooperation with the data store;</i>	<i>39</i>
(v)	<i>Element [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, ...</i>	<i>44</i>
(vi)	<i>Element [1d.ii]: wherein the node memory further comprises algorithm memory including resource-specific memory locations to support the data processing operation;</i>	<i>53</i>
(vii)	<i>Element [1e]: a data mover component to facilitate intra-node data communications between the programmable logic component and the node memory; and</i>	<i>56</i>
(viii)	<i>Element [1f]: a storage interface component to facilitate intra-node data communications between the programmable logic component and the data store;</i>	<i>58</i>
(ix)	<i>Element [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</i>	<i>63</i>
(x)	<i>Element [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.</i>	<i>68</i>
2.	Claim 2	<i>69</i>

(i)	<i>Element [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.</i>	69
3.	Claim 3	71
(i)	<i>Element [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.</i>	71
4.	Claim 6	73
(i)	<i>Element [6]: The compute node of claim 1 wherein the programmable logic component comprises a field programmable gate array (FPGA).</i>	73
5.	Claim 7	75
(i)	<i>Element [7]: The compute node of claim 1 wherein the compute node further comprises a processor to manage operation of the data mover component.</i>	75
6.	Claim 8	77
(i)	<i>Element [8]: The compute node of claim 7 wherein the data mover component comprises a device controller.</i>	77
7.	Claim 9	79
(i)	<i>Element [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.</i>	79
8.	Claim 10	81
(i)	<i>Element [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.</i>	81

9.	Claim 11	85
(i)	<i>Element [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.</i>	85
10.	Claim 12	89
(i)	<i>Element [12]: The compute node of claim 11 wherein the computer-executable code associated with the computationally expensive algorithm is incorporated in a bit stream.</i>	89
IX.	GROUND 1A: DALAL, NG, AND RAGHAVA	90
A.	Raghava (AMD1006).....	90
B.	The Dalal-Ng-Raghava Combination	92
C.	Claim Element Analysis.....	95
1.	Claim 4	95
(i)	<i>Element [4]: The compute node of claim 1 wherein the data comprise an Electronically Erasable Programmable Read Only Memory (EEPROM) device.</i>	95
2.	Claim 5	97
(i)	<i>Element [5]: The compute node of claim 4 wherein the storage interface employs an Open NAND Flash Interface (ONFI) protocol.</i>	97
X.	GROUND 2A: HEIL AND JUN.....	98
A.	Heil (AMD1010)	98

B.	Jun (AMD1011).....	108
C.	The Heil-Jun Combination.....	110
D.	Claim Element Analysis.....	116
1.	Claim 1	116
(i)	<i>Element [1pre]: A compute node for use in a data processing system, the compute node comprising:</i>	116
(ii)	<i>Element [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;</i>	117
(iii)	<i>Element [1b]: a data store comprising records associated with a data processing operation;</i>	124
(iv)	<i>Element [1c]: a programmable logic component to execute the data processing operation in cooperation with the data store;</i>	129
(v)	<i>Element [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, ...</i>	133
(vi)	<i>Element [1d.ii] wherein the node memory further comprises algorithm memory including resource-specific memory locations to support the data processing operation;</i>	138
(vii)	<i>Element [1e]: a data mover component to facilitate intra-node data communications between the programmable logic component and the node memory; and</i>	140
(viii)	<i>Element [1f]: a storage interface component to facilitate intra-node data communications between the programmable logic component and the data store; ..</i>	142
(ix)	<i>Element [1g]: wherein the storage interface component utilizes a plurality of communications channels to</i>	

	<i>transfer data between the programmable logic component and the data store; and</i>	145
(x)	<i>Element [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.</i>	148
2.	Claim 2	150
(i)	<i>Element [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.</i>	150
3.	Claim 3	154
(i)	<i>Element [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.</i>	154
4.	Claim 4	155
(i)	<i>Element [4]: The compute node of claim 1 wherein the data comprise an Electronically Erasable Programmable Read Only Memory (EEPROM) device.</i>	155
5.	Claim 6	156
(i)	<i>Element [6]: The compute node of claim 1 wherein the programmable logic component comprises a field programmable gate array (FPGA).</i>	156
6.	Claim 7	158
(i)	<i>Element [7]: The compute node of claim 1 wherein the compute node further comprises a processor to manage operation of the data mover component.</i>	158
7.	Claim 8	160

(i)	<i>Element [8]: The compute node of claim 7 wherein the data mover component comprises a device controller.</i>	160
8.	Claim 9	161
(i)	<i>Element [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.</i>	161
9.	Claim 11	162
(i)	<i>Element [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.</i>	162
10.	Claim 12	165
(ii)	<i>Element [12]: The compute node of claim 11 wherein the computer-executable code associated with the computationally expensive algorithm is incorporated in a bit stream.</i>	165
XI.	GROUND 1A: HEIL, JUN AND RAGHAVA.....	167
A.	The Heil-Jun-Raghava Combination	167
1.	Incorporation of Raghava’s ONFI Protocol with the Heil-Jun Combination	167
2.	Incorporation of Raghava’s Interface Card with the Heil-Jun Combination	170
B.	Claim Element Analysis.....	176
1.	Claim 5	176

(i)	<i>Element [5]: The compute node of claim 4 wherein the storage interface employs an Open NAND Flash Interface (ONFI) protocol.</i>	176
2.	Claim 10	177
(i)	<i>Element [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.</i>	177
XII.	CONCLUSION	178

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 comprise 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.

I, Donald Alpert, PhD, declare as follows:

I. INTRODUCTION

1. I have been retained as a technical expert by counsel on behalf of Advanced Micro Devices, Inc. (“AMD”/“Petitioner”). I understand that AMD is requesting that the Patent Trial and Appeal Board institute an *inter partes* review (“IPR”) proceeding with respect to U.S. Patent No. 12,271,636 (“the ’636 patent”) (AMD1001).

2. I have been asked to provide my independent analysis of the ’636 patent in light of certain prior art publications, and I have done so based on my personal knowledge and experience.

3. I am not, and never have been, an employee of AMD. I received no compensation for this Declaration beyond my normal hourly compensation based on my time actually spent analyzing the ’636 patent, the prior art publications cited below, and various issues related thereto. I will not receive any added compensation based on the outcome of any IPR or other proceeding involving the ’636 patent.

II. QUALIFICATIONS

4. In formulating my opinions, I have relied on my knowledge, training, and experience in the relevant field. My education and experience are described more fully in the attached curriculum vitae (Appendix A). For ease of reference, I have highlighted certain information below.

5. I am an independent consultant with Camelback Computer Architecture, LLC. My residence and place of business is at 7 Farm Road, Los Altos CA, 94024. I am over the age of eighteen, and I am a citizen of the United States.

6. I have over 45 years of academic and industrial experience in applying, designing, studying, teaching, and writing about microprocessors and computer systems. I received an Electrical Engineering Ph.D. degree in 1984 from Stanford University. I earlier received an Electrical Engineering B.S. degree from MIT in 1973 and an Electrical Engineering M.S. degree from Stanford University in 1978. I have taught classes in computer architecture at Stanford, Tel Aviv, and Arizona State Universities.

7. From 1976 to 1977, I worked at Burroughs Corporation, where I designed peripheral interface controllers, including those for serial data communications based on Intel 8080 microprocessor components. From 1980 to 1989, I was the lead architect for the design of three high-performance microprocessors at Zilog and National Semiconductor. Later, at Intel, I was the lead architect of the Pentium® Processor from 1989 to 1992 and of the 815 chipset from 1999 to 2000, both of which became the most widely used PC components of their time. The 815 chipset comprised two components: (1) a memory controller hub (MCH) that included a graphics controller and memory controller with interfaces to the CPU, 133 MHz SDRAM system memory modules, an optional, external graphics

controller and (2) an I/O controller hub (ICH) that included various I/O controllers (e.g., network, hard drive, USB) for system peripheral devices. Additionally, I served as co-manager for the Itanium processor design from 1993-1997.

8. I am a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), and served as the chair of the IEEE Technical Committee on Microprocessors and Microcomputers from 1999 to 2000. I was the keynote speaker at the first Cool Chips conference, dedicated to the study of low-power microprocessors and systems. I have given invited lectures at several universities, and published ten papers in various professional journals and conference proceedings. My paper entitled "Architecture of the Pentium Processor," was selected as best paper in IEEE Micro for 1993. I am a named inventor on over 30 U.S. patents that pertain to microprocessors, computer systems, and related technology.

9. I have reviewed the '636 patent, and I am familiar with the patent's subject matter, which is within the scope of my education and professional experience. Based at least on my background in academia and industry, I am familiar with the issues and technology relating to processors, chipsets, memory, and peripheral devices for computer systems, including those for scalable, multiprocessor systems and those with dedicated processing accelerators. I have personally analyzed, developed, and tested such computer components and systems.

10. A detailed list of my other professional activities, memberships, and speaking engagements is included in my CV.

III. LEGAL PRINCIPLES

11. In forming my analysis and conclusions expressed in this Declaration, I have applied the legal principles described in the following paragraphs, which were provided to me by counsel for the Petitioner.

A. Person of Ordinary Skill in the Art

12. I understand that the factors considered in determining the ordinary level of skill in a field of art include: the level of education and experience of persons working in the field; the types of problems encountered in the field; the teachings of the prior art regarding solutions to such problems; and the sophistication of the technology at the time of the alleged invention. I understand that a person of ordinary skill in the art (“POSITA”) is not a specific real individual, but rather is a hypothetical individual having the qualities reflected by the factors above and knowledge of all relevant prior art references, including the references I cite below.

B. Obviousness

13. I have been informed that a patent claim is invalid as “obvious” in light of one or more prior art references if it would have been obvious to a POSITA at the time of the alleged invention, taking into account (1) the scope and content of the prior art, (2) the differences between the prior art and the claims, (3) the level of

ordinary skill in the art, and (4) any so called “secondary considerations” of non-obviousness, which include: (i) “long-felt but unresolved need” for the claimed invention, (ii) commercial success attributable to the claimed invention, (iii) unexpected results of the claimed invention, and (iv) “copying” of the claimed invention by others.

14. While I do not know the exact date that the alleged invention of the ’636 patent was made, I do know that the ’636 patent’s earliest claimed priority date is February 6, 2019. For purposes of my obviousness analysis, I have applied a date of February 6, 2019 as the date of the alleged invention (“Critical Date”), although in many cases the same analysis would hold true even if the date of the alleged invention were earlier or later.¹

15. I have been informed that a claim can be obvious in light of a single prior art reference or multiple prior art references. To be obvious in light of a single prior art reference or multiple prior art references, there must be a reason that would have prompted a POSITA to modify or supplement the single prior art reference, or to combine two or more references, in a manner that provides the elements of the claimed invention. This reason may come from a teaching, suggestion, or

¹ Note, however, that I do not concede that the ’636 patent is entitled to its earliest claimed priority date.

motivation to combine, or may come from the reference(s) themselves, the knowledge or “common sense” of a POSITA, or from the nature of the problem to be solved, and this reason may be explicit or implicit from the prior art as a whole. I have been informed that, under the law, the combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results. I also understand it is improper to rely on hindsight in making the obviousness determination.

16. I understand that an obviousness determination also requires that a POSITA would have had a reasonable expectation of success. This concept has been explained to me as relating to the relative likelihood or predictability—from the perspective of a POSITA—of successfully modifying the prior art in a manner that would meet the claimed limitations of the patent being challenged (here the ’636 patent). I understand that the expectation of success need only be “reasonable” and, thus, does not require the absolute certainty gleaned from physically creating the proposed prior art modification.

C. Claim Interpretation

17. I understand that, for purposes of my analysis in this *inter partes* review proceeding, the terms appearing in the patent claims should be interpreted according to their “ordinary and customary meaning.” In determining the ordinary and customary meaning, I understand that the words of a claim are first given their plain

meaning that those words would have had to a person of ordinary skill in the art at the time of the alleged invention. I also understand that the structure of the claims, the specification and file history may be used to better construe a claim. Moreover, I understand that even treatises and dictionaries may be used, albeit under limited circumstances, to determine the meaning attributed by a person of ordinary skill in the art to a claim term.

IV. MATERIALS CONSIDERED

18. My analysis in this Declaration is based on my knowledge and experience. Based on my above-described qualifications in Section II, the Patent Trial and Appeal Board should consider me to be an expert in the field. Also, based on my experiences, I understand and know of the capabilities of persons of ordinary skill in this field.

19. As part of my independent analysis for this Declaration, I have considered the following: the '636 patent (AMD1001) and its prosecution history (AMD1002); the background knowledge/technologies that were commonly known to persons of ordinary skill; my own knowledge and experience gained from my work in the field; my experience in teaching and advising others in this field; and my experience working with others involved in this field. In addition, I have analyzed the following publications and materials:

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. US2016/0306668 (“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”)
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)
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”)
AMD1030	IEEE “The Authoritative dictionary of IEEE Standards Terms, Seventh Edition” (2000)
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”)
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” <u>2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)</u> (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”)
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”)

20. Note that my citations to non-patent literature throughout this Declaration reference the absolute page number added to the exhibit (as opposed to the original pagination of the document). For patent literature, I’ve used the column/line numbers or paragraph numbers. Note also that, unless I’ve indicated otherwise, all emphasis in any quoted text are ones I have added.

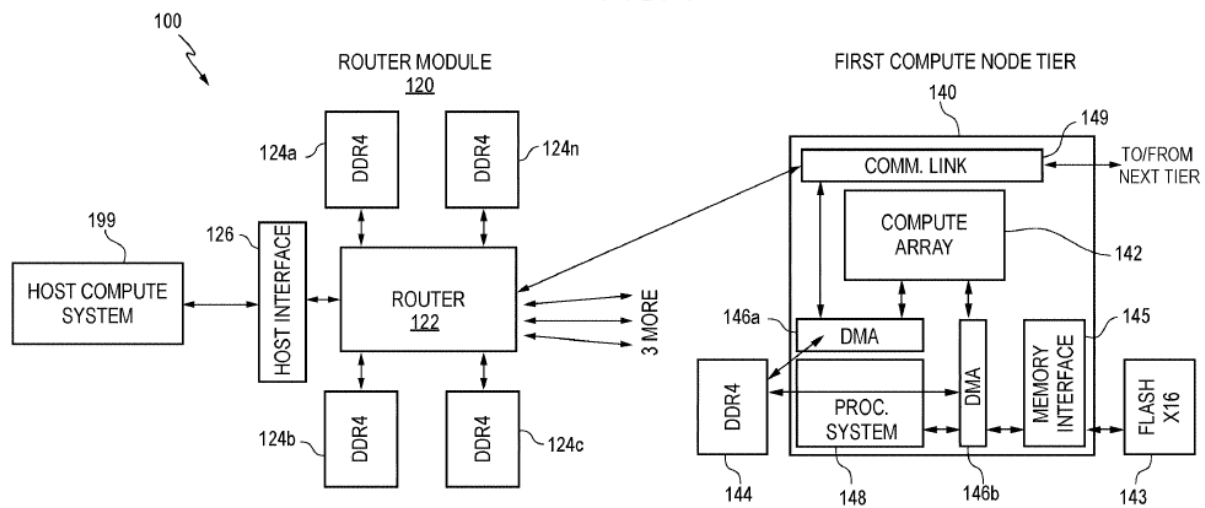
21. Although I cite and quote to selected portions of various references in this Declaration, the reader should understand that these citations are representative examples. A POSITA would have viewed each reference in its entirety and in combination with other references. Accordingly, I intend the references identified in this Declaration to be viewed as incorporated in their entireties.

V. THE '636 PATENT

A. Pertinent Disclosure

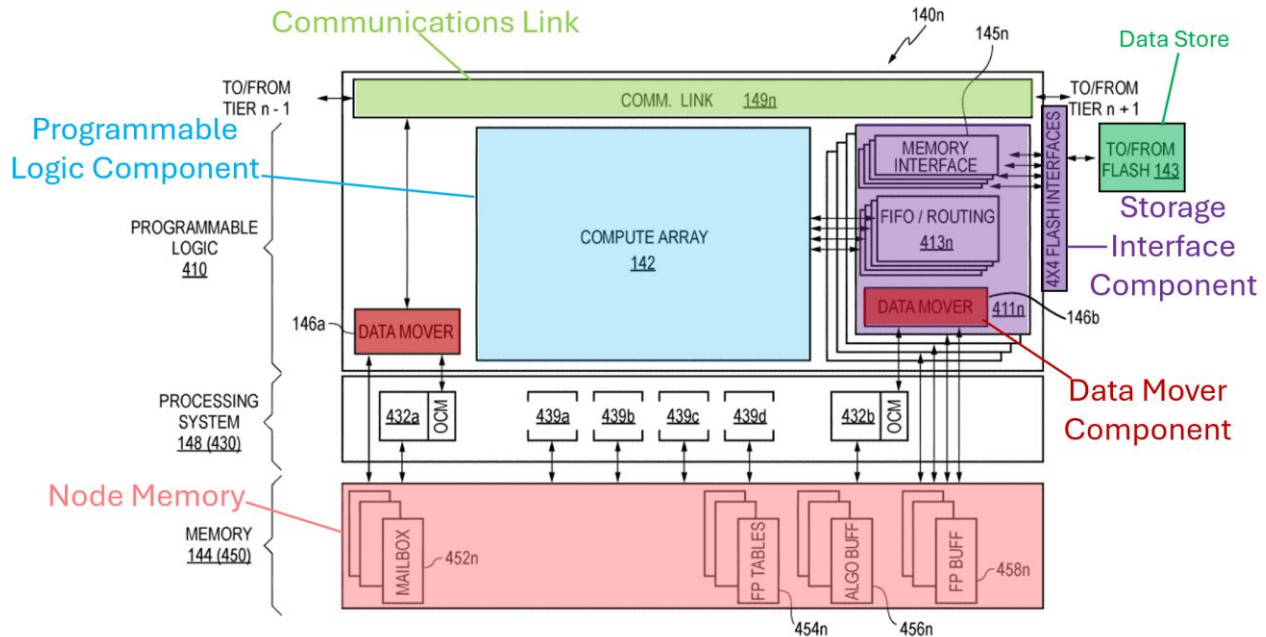
22. The '636 patent describes a “data processing system” that can include a plurality of nodes performing operations in an execution pipeline. AMD1001, Title, 7:60–8:11, FIG. 1 (reproduced below). An execution pipeline can include a plurality of nodes that are connected to each other in series, the first node being connected to a router. AMD1001, 5:11–45, 43:36–44:34.

FIG. 1



AMD1001, FIG. 1.

23. Every compute node (e.g., compute node 140n shown in Figure 4 below) in an execution pipeline described by the '636 patent includes a programmable logic component, data store, storage interface component, data mover component, node memory, and communications link, among other components. AMD1001, 10:41–11:45, 55:8–56:41; FIGS. 1, 4. These components are depicted below in Figure 4 of the '636 patent.



AMD1001, FIG. 4 (annotated).

24. A compute node initiates the execution pipeline by executing its operation for an algorithm and passing the result of the operation to the next node in the execution pipeline. AMD1001, 44:35–45:35, FIG. 22. Each of the additional nodes in the execution pipeline execute respective operations on input data received

from the preceding compute node and pass the respective results to the subsequent compute node. AMD1001, 44:35–45:35, FIG. 22.

B. Prosecution History

25. During prosecution of the '636 patent, the Examiner rejected all claims as anticipated by Whitehead (AMD1009) and found several terms indefinite. AMD1002, PP.46–71. In response, PO amended claim 1 to include Element [1d.ii] and Element [1h] arguing Whitehead lacked a similar “operational context” and failed to disclose “computationally expensive algorithms.” *Id.* PO cited “Big Data or artificial intelligence applications” as examples of “computationally expensive algorithms.” *Id.* However, as I have demonstrated below, neither the “operational context,” “computationally expensive algorithm,” nor any other claimed element is patentably distinct from the prior art cited in this Petition and the '636 patent never should have been allowed.

C. Interpretation of the '636 Patent Claims at Issue

26. It is apparent that the Challenged Claims are obvious under any reasonable interpretation advanced pursuant to the *Phillips* standard. There is not a need to resolve any express constructions to show that my analysis of Grounds 1A-2B demonstrates that the claims are obvious over the applied references. I have been asked to consider one interpretation of the claim term “programmable logic component” as a “logic component programmable by reconfiguring its internal

hardware connections, such as an FPGA.” This interpretation is supported by the intrinsic record and consistent with the ordinary meaning. *See, e.g.*, AMD1001, 15:55–56, 39:60–64, 20:43–50.

VI. PERSON OF ORDINARY SKILL IN THE ART

A. A POSITA’s Ordinary Level of Skill

27. Based on my knowledge and experience in the field and my review of the ’636 patent and its file history, I believe that a POSITA would have included 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. Additional experience could substitute for a formal degree or formal training (and *vice versa*).

28. My analysis and conclusions expressed in this Declaration are based on the perspective of a POSITA having this level of knowledge and skill. My education, technical expertise and personal knowledge discussed in Section II shows that I meet the qualifications of a POSITA.

VII. OVERVIEW OF CONCLUSIONS FORMED

29. This Declaration explains the conclusions that I have formed based on my independent analysis. To summarize those conclusions:

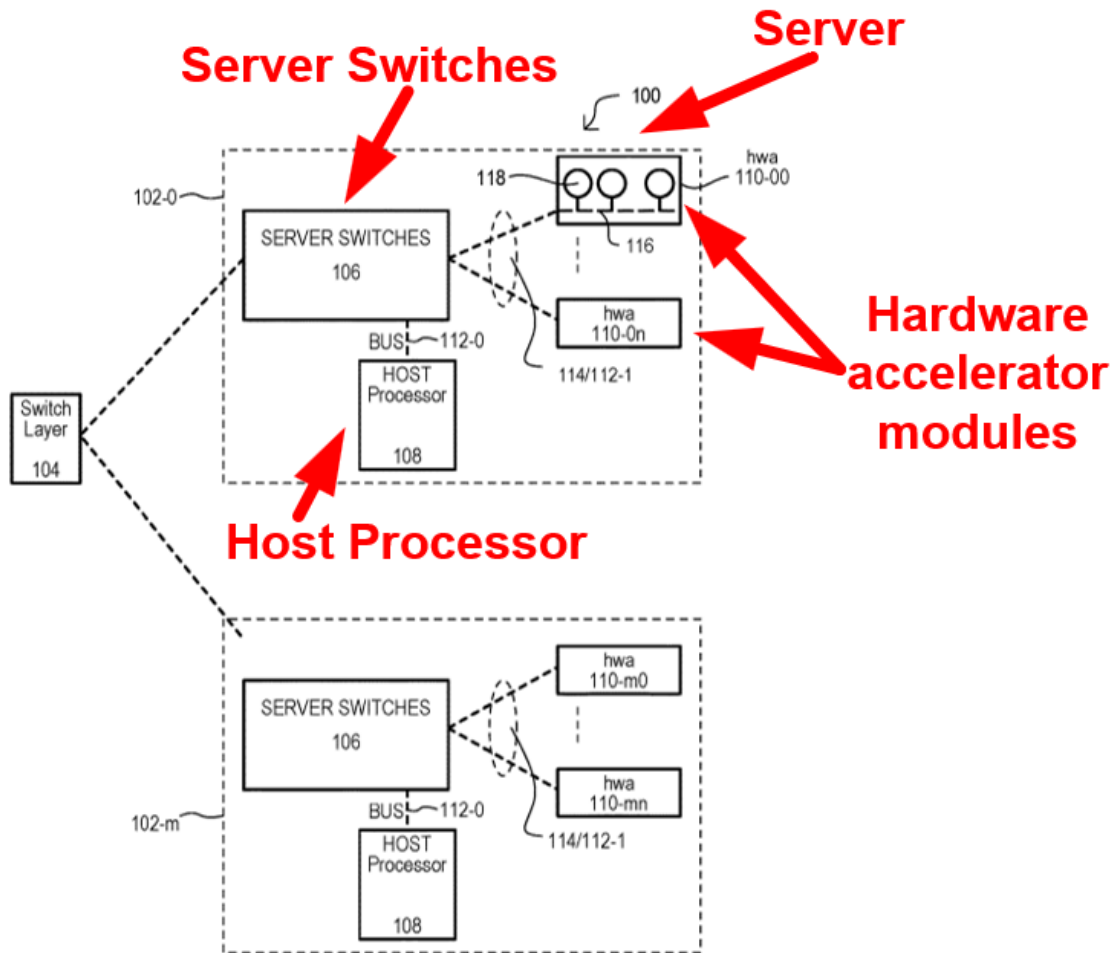
- Based upon my knowledge and experience, and my review of the materials listed above, I believe that Claims 1–3, 6–12 of the '636 patent are obvious in view of Dalal (AMD1004) and Ng (AMD1005). *See* Ground 1A at §VIII.
- Based upon my knowledge and experience, and my review of the materials listed above, I believe that Claims 4 and 5 of the '636 patent are obvious in view of Dalal (AMD1004), Ng (AMD1005), and Raghava (AMD1006). *See* Ground 1B at §IX.
- Based upon my knowledge and experience, and my review of the materials listed above, I believe that Claims 1–4, 6–9, 11, and 12 of the '636 patent are obvious in view of Heil (AMD1010) and Jun (AMD1011). *See* Ground 2A at §X.
- Based upon my knowledge and experience, and my review of the materials listed above, I believe that Claims 5 and 10 of the '636 patent are obvious in view of Heil (AMD1010), Jun (AMD1011), and Raghava (AMD1006). *See* Ground 2B at §XI.

VIII. GROUND 1A: DALAL AND NG

A. Dalal (AMD1004)

30. According to Dalal, servers can be connected to one another in a distributed computing environment. AMD1004, [0026] (“computing elements can be included in a network architecture to provide a heterogenous computing

environment...the computing elements can be...included in server systems”), *see also*, [0027]–[0048], FIGS. 1–2C. For example, Dalal’s servers can include “hardware accelerator modules” that are similar to the ’636 patent’s compute nodes, especially because each module includes “computing elements” that process data. AMD1004, [0026] (“the computing elements can be formed on hardware accelerator (hwa) modules”), *see also*, [0027], [0032]–[0036], [0038]–[0048]. The hardware accelerator modules receive packet data, process this data, and output the processed data using the computing elements. AMD1004, [0032]–[0036], [0040], [0042]–[0044], FIGS. 1, 2B–2C. In each of Dalal’s servers, there are “server switches” coupled to the hardware accelerator modules and a “host processor” so that the host processor is communicatively coupled to the hardware accelerator modules. AMD1004, [0032], FIG. 1.

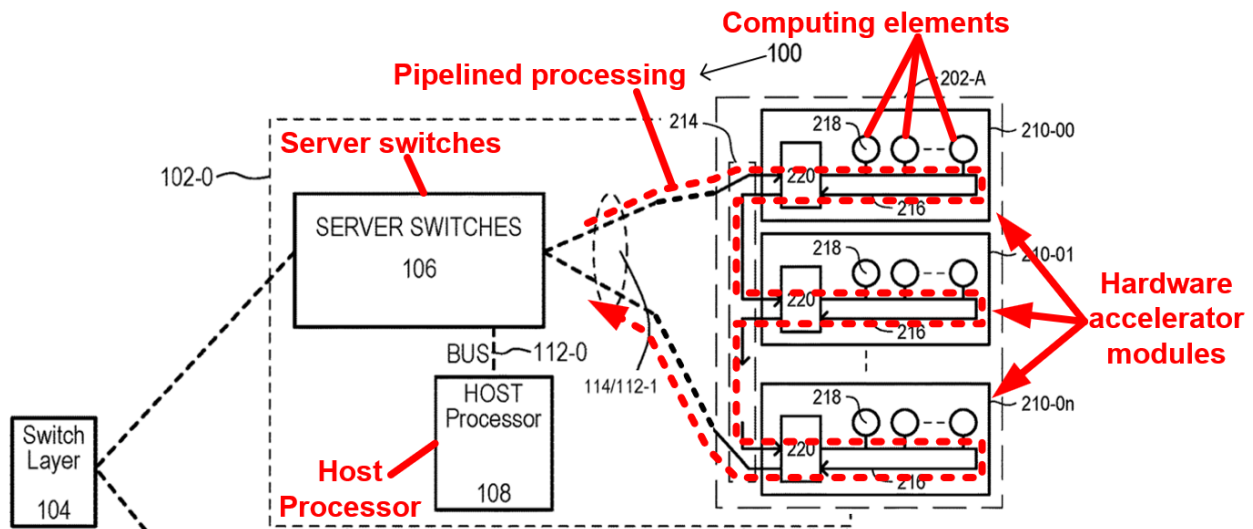


AMD1004, FIG. 1 (annotated).

31. Dalal explains that “FIG. 2A to 2C show various systems, which...can be particular versions of system 100 of FIG. 1,” meaning that each of the examples in FIGS. 2A to 2C represents a particular example of system 100. AMD1004, [0039]. Specifically, it is clear that the “multiple hwa modules 210-00 to -0n” of FIG. 2B are particular examples of the “one or more hwa modules 110-00 to -mn.” AMD1004, [0032], [0042], FIGS. 1, 2B. The below drawing is a combined figure

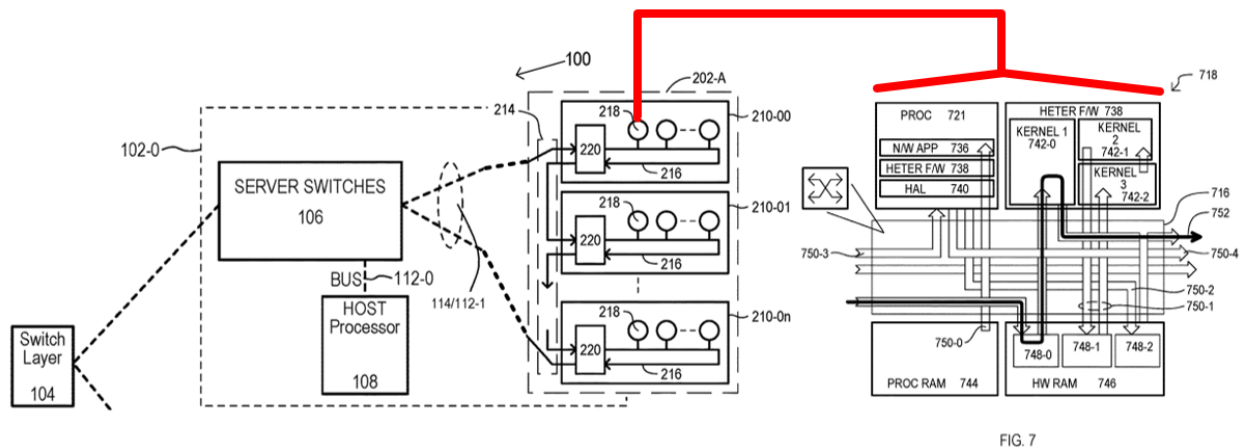
that integrates the system of Figure 1 with the “particular version” of Figure 2B, and this is consistent with Dalal’s disclosure. In the combined figure, the hwa modules 210 in box 202-A from FIG. 2B replace the hwa modules 110. The combined figure resulting from this substitution depicts an embodiment that is expressly disclosed by Dalal, because FIG. 2B simply illustrates a more detailed view of one “particular version” of FIG. 1’s system, namely a more detailed view of how the hardware accelerator modules are arranged. In FIG. 1, “hwa module 110-00” and “hwa module 110-0*n*” are joined with a dotted line. FIG. 2B includes solid lines indicating exactly how modules are connected to each other within a server.

32. Within one of Dalal’s servers, the hardware accelerator modules process data sequentially, one after the other, and this processing is achieved in part because the hardware accelerator modules are serially coupled to one another. AMD1004, [0042], [0045]–[0047], FIGS. 2B–2C. These serially coupled modules are depicted below the combined figure. AMD1004, [0042], [0045]–[0047], FIGS. 2B–2C. Dalal refers to sequential processing by serially coupled hardware accelerator modules as “pipelined processing.” AMD1004, [0035], [0036], [0042], [0045], [0047]. To communicate with other devices (e.g., other modules, server switches, and the host processor), a hardware accelerator module includes a “switching circuit” that can connect the module to a bus. AMD1004, [0040], [0041], [0044], [0048], FIGS. 2A–2C.



AMD1004, FIGS. 1, 2B Combination Figure (annotated).

33. The “computing elements” located in Dalal’s hardware accelerator modules each include several components that are used for processing data such as processing circuits (e.g., programmable logic), a first memory, and a second memory. AMD1004, [0075]–[0088], FIG. 7. Programs stored by the first memory can include “machine learning applications” that govern data processing across “multiple computing elements” in the hardware accelerator modules. AMD1004, [0078]. The second memory includes “predetermined memory spaces” that store data used by the processing circuits to run the program. AMD1004, [0076], [0078], [0080].



AMD1004, FIGS. 2B, 7 Combination Figure (annotated).

B. Ng (AMD1005)

34. Ng is similar to Dalal in many respects. For example, Ng describes a “hardware accelerator” having programmable logic and memory that executes a data processing operation. AMD1005, [0025]–[0027], [0029]–[0033]. Ng also describes a “host computer system” that communicates with the hardware accelerator to perform complex data processing operations. AMD1005, [0025]–[0042], FIGS. 1–4. The hardware accelerator includes a programmable integrated circuit (IC) such as a field programmable gate array (FPGA) that can communicate with memory and a host computer system. AMD1005, [0032], [0034], [0035]–[0039], FIGS. 2–4. The hardware accelerator processes data received from the host computer system and outputs the processed data to the host computer system. AMD1005, [0032], [0034], [0035]–[0039], FIGS. 2–4.

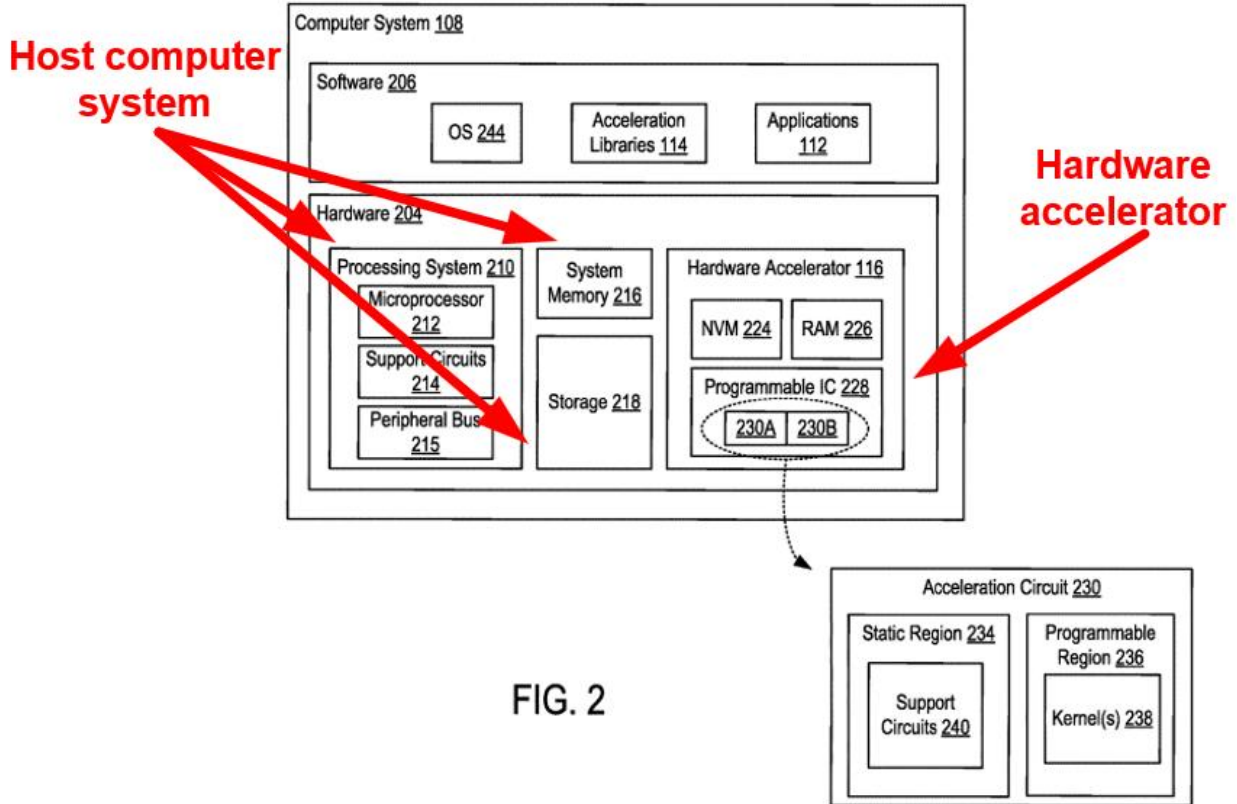
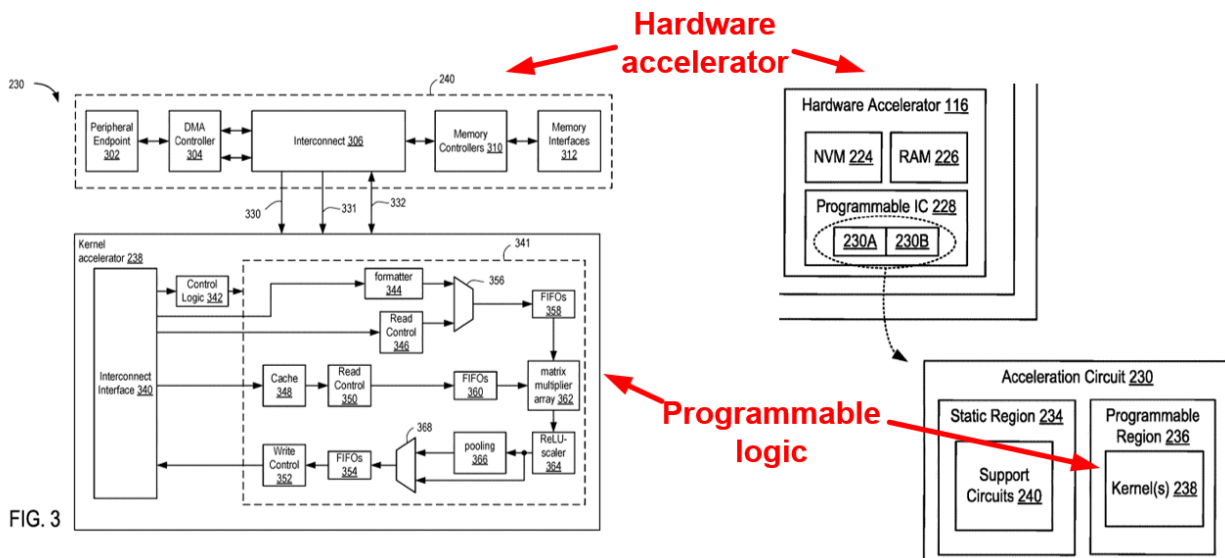


FIG. 2

AMD1005, FIG. 2 (annotated).

35. The hardware accelerator can perform complex data processing operations to execute multiple layers of a neural network sequentially, the output from one neural network layer serving as the input to the next neural network layer. AMD1005, [0007], [0008], [0021]–[0030], [0033], [0034], [0036]–[0039], [0042], [0044]–[0046], FIGS. 1–4. This is similar to the system of Dalal, which can perform machine learning operations using hardware accelerator modules. *Supra*, §VIII.A. One example of a machine learning operation is when the hardware accelerator performs convolution and matrix multiplication for each layer of the neural network

using custom sets of input data. AMD1005, [0021]–[0024], [0039]–[0041] [0044]. To perform these operations, the hardware accelerator includes a “programmable integrated circuit (IC)” having (1) a “static region” having “support circuits that transport data between memory and programmable logic, and (2) a “programmable region” including a “neural network accelerator” that processes the data. AMD1005, [0034], [0035], FIG. 3.



AMD1005, FIGS. 2, 3 (annotated).

C. The Dalal-Ng Combination

36. The Dalal and Ng references are analogous art to the '636 patent, for at least the reasons I have provided below. For example, the '636 patent is directed to “a system and method employing a distributed hardware architecture...in cooperation with an attendant data structure” that “facilitates 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. In the same field of endeavor as the ’636 patent, Dalal describes a “network architecture” including “hardware accelerator (hwa) modules” that are arranged to perform complex operations from a “machine learning library, such as TensorFlow.” AMD1004, [0026], *see also*, [0032], [0042], [0078]. Dalal’s hardware accelerator modules are therefore arranged to perform complex operations such as “machine learning,” which I noticed is similar to the system of the ’636 patent which performs machine learning operations using a similar architecture (e.g., TensorFlow). AMD1004, [0032]–[0036], [0040]–[0050], [0078], FIGS. 1, 2B–2C; *supra*, §VIII.A. In addition to Dalal, Ng is within the same field of endeavor as the ’636 patent and describes solutions relevant to one or more problems addressed by the ’636 patent because Ng describes how a host computer system can provide tailored instructions to a single hardware accelerator for executing every layer of a neural network. AMD1005, [0021]–[0022], [0024], [0039]–[0042], [0044]–[0046], FIGS. 2–4.

37. Executing a machine learning network layer-by-layer with a hardware accelerator is generally known in the art, as I describe below with respect to Dalal and Ng, and this is exactly the kind of operation that the ’636 patent describes as being optimal for an execution pipeline of compute nodes. AMD1001, 48:36–64 (“[o]ne example of a Level 2 algorithm as contemplated herein is a common inference algorithm for a neural network executing under TensorFlow™”), 49:59–

50:10 (explaining that a “tier 1” compute node can “execute required or desired computations for the first layer in the neural network” and “transmit computational results or output data to the [next] compute node” which “may begin processing operations at the next layer in the neural network”). A POSITA would have been motivated to modify Dalal by incorporating aspects of “host computer system,” a programmable “neural network accelerator,” “memory interfaces,” and a memory storing records such as matrices and work requests. AMD1005, [0025]–[0051]. Dalal’s host processors are implemented with the host computer system of Ng in the Combination, and each one of Dalal’s hwa modules are implemented with Ng’s hardware accelerator of Ng in the Combination. Making these implementations preserves Dalal’s pipelined architecture and supplements Dalal’s servers with aspects of the host-accelerator cooperation described in Ng.

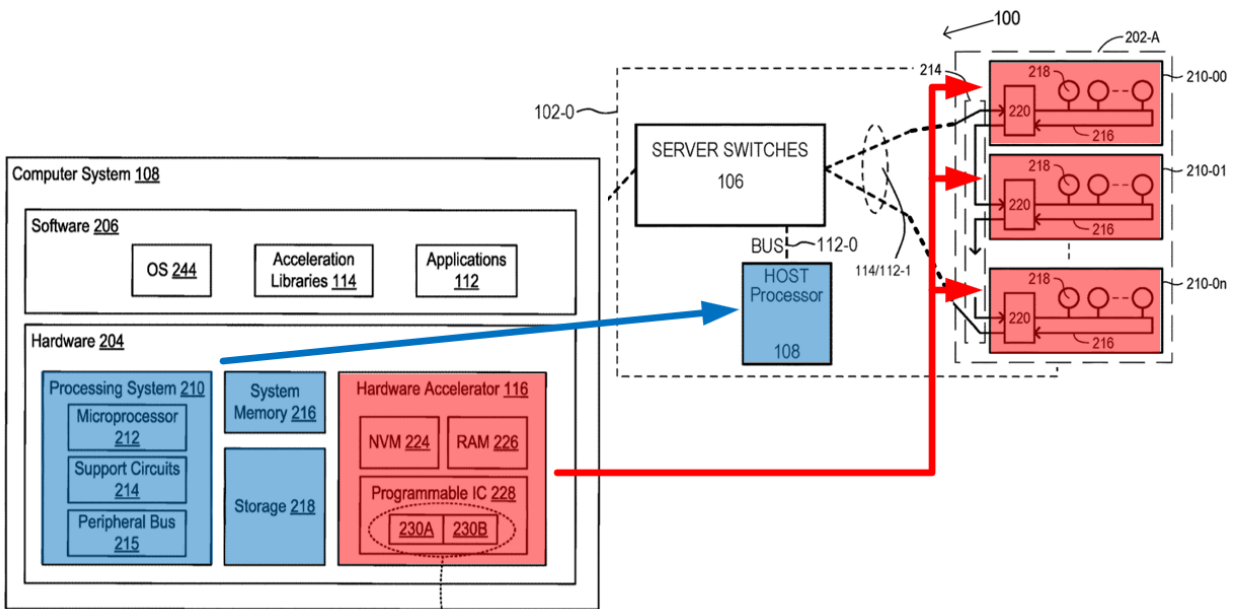


Illustration of a Portion of the Dalal-Ng Combination.

38. The following reasons, taken separately or collectively, support the Dalal-Ng Combination.

39. First, a POSITA would have implemented Dalal's host processor with Ng's teaching of a host computer system that with Ng's host computer system that actively coordinates with a hardware accelerator to effectively manage complex machine learning operations. For example, Dalal explains that hardware accelerators use "pipelined processing" to sequentially process packet data. AMD1004, [0035], [0036], [0042]. Ng describes an effective division of labor between the host and hardware accelerator, whereby the accelerator performs the computations necessary 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]. 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 allowing the host to provide instructions while reducing an amount of host-accelerator communication. AMD1004, [0063], [0077]; AMD1005, [0023].

40. Second, a POSITA would have understood that implementing Dalal's system with one or more of Ng's "design tools" such as a user interface for

programming the hardware accelerators would have beneficially allowed a developer to customize Dalal's pipelined processing to achieve specialized goals. AMD1005, [0027]. For example, Dalal and Ng both describe using TensorFlow to cause a hardware accelerator to execute machine learning tasks. AMD1004, [0078]; AMD1005, [0028], [0043]. Hardware accelerators are programmable to perform custom operations more efficiently than a general processor (e.g., a CPU or GPU) could perform these tasks because hardware accelerators include programmable logic that can be configured and reconfigured to perform custom tasks. AMD1005, [0006] ("an FPGA circuit can process data faster than a GPU and consume less power in the process"). Implementing Dalal's server with design tools, as described by Ng, would have allowed a developer to tailor hardware accelerators to a specific operation, which is more efficient and consumes less power than a general processor. AMD1004, [0063], [0077].

41. Third, configuring programmable logic of Dalal's hardware accelerator modules to perform computations such as "matrix multiplication" and "convolution" as described by Ng would have beneficially provided an effective way to execute machine learning operations using these modules. AMD1005, [0021], [0039], [0041], [0042]. A POSITA would have understood that matrix multiplication and convolution are common operations in machine learning tasks such as executing a deep neural network (e.g., a convolutional neural network (CNN)). Configuring

Dalal's hardware accelerator modules to perform convolution and matrix multiplication would have positioned these modules to efficiently process data to execute a neural network. At the time of the '636 patent, there was a strong motivation to optimize hardware accelerators for performing complex operations such as machine learning because machine learning is a specialized task that can be more efficiently handled by a hardware accelerator as compared with a CPU or a GPU.

42. Fourth, memory interface circuits manage communication between programmable logic and memory. Implementing Dalal's hardware accelerator modules with a memory interface circuit that operates according to a protocol of a coupled memory, as taught by Ng, 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.

43. 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. 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. A POSITA would have understood that because hardware accelerators are

often used in combination with other processors and computing elements under direction from a host, it would have made sense to implement Dalal's hardware accelerators with a host that permits a developer to configure the modules to perform a custom task.

44. As I explain below in my analysis of the Dalal-Ng combination, implementing Dalal's hardware accelerator modules with a cooperative host and interfaces for communicating with memory merely incorporates well known features that yield predictable results. It is apparent that supplementing Dalal's hardware accelerator modules in this way would not have disturbed the other aspects of Dalal's processing system, and in fact would have enhanced the system's ability to handle multi-step operations in a systematic manner.

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

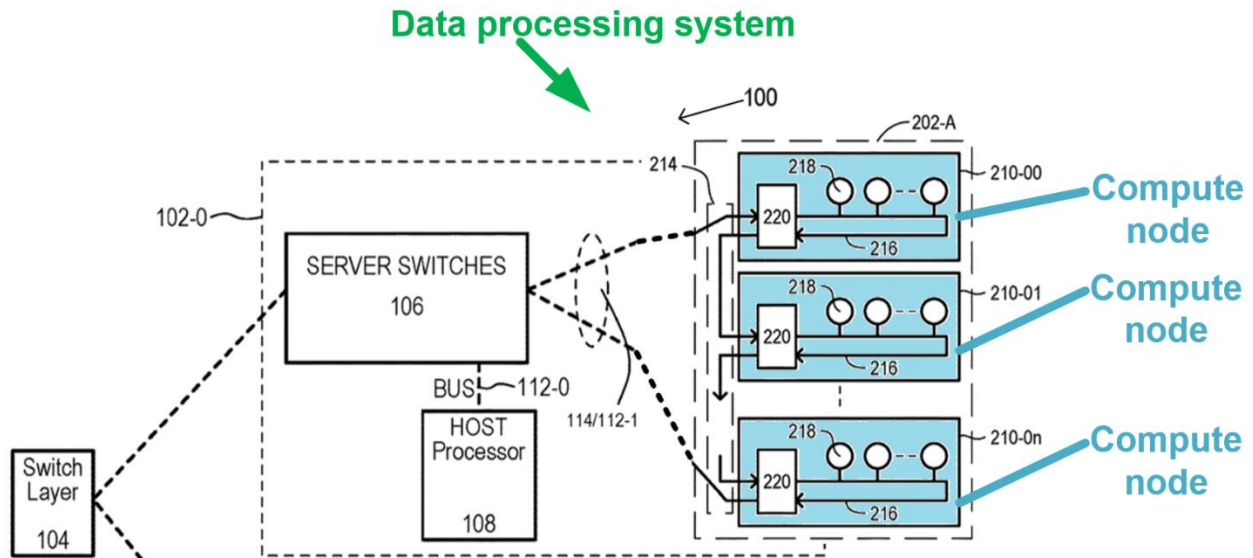
D. Claim Element Analysis

1. Claim 1

(i) *Element [1pre]: A compute node for use in a data processing system, the compute node comprising:*

46. The Dalal-Ng Combination renders obvious Element [1pre], even if the preamble is limiting. Dalal describes a **system** includes “a number of servers” connected to each other via a switching layer. AMD1004, [0031] (“[a] system 100 can include a number of servers...which can be networked together by one or more switching layers”), *see also*, [0032]–[0047], [0089]–[0092], [0096]–[0099], [0123]–[0125], [0170]–[0172], FIGS. 1–2C, 8, 12, 18. Each server includes “one or more hwa modules” (**compute nodes**) that can “execute operations on received **data**,” indicating that the modules are part of a **data processing system**. AMD1004, [0032], [0033], *see also*, [0035] (“servers...can receive packet **data** to be **processed**”), [0036] (“within hwa module...packet **data** can be **processed**”), [0075], [0089], FIG. 1. 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]. In the **data processing system**, each server includes hardware accelerator modules (**compute nodes**) 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 Figure (annotated).²

(ii) *Element [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;*

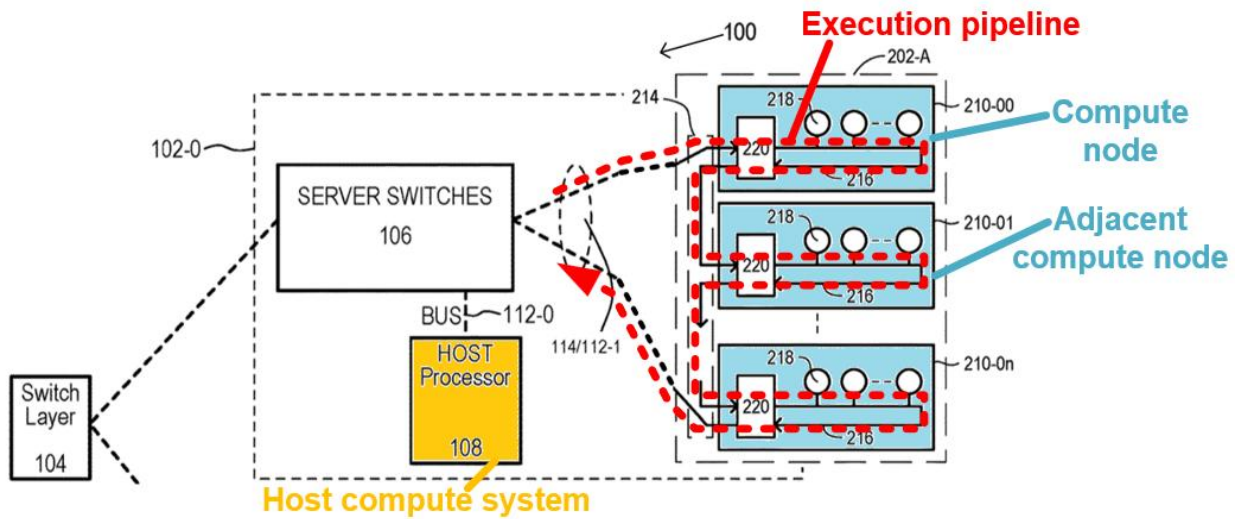
47. The Dalal-Ng Combination renders obvious [1a]. Each server includes “one or more *host* processors” representing a *host compute system*. AMD1004, [0032], *see also*, [0031]–[0038], [0040]–[0042], [0046]–[0047], [0063], [0069]–[0070], [0073], [0077], [0099], [0101]–[0103], [0123]–[0125], [0170]–[0172],

² Recall from §VIII.A that Dalal explains that FIG. 2B embodies a version of FIG.

1. AMD1004, [0039]. Thus, FIGS. 1 and 2B would have been combined in the manner illustrated here consistent with Dalal's teachings.

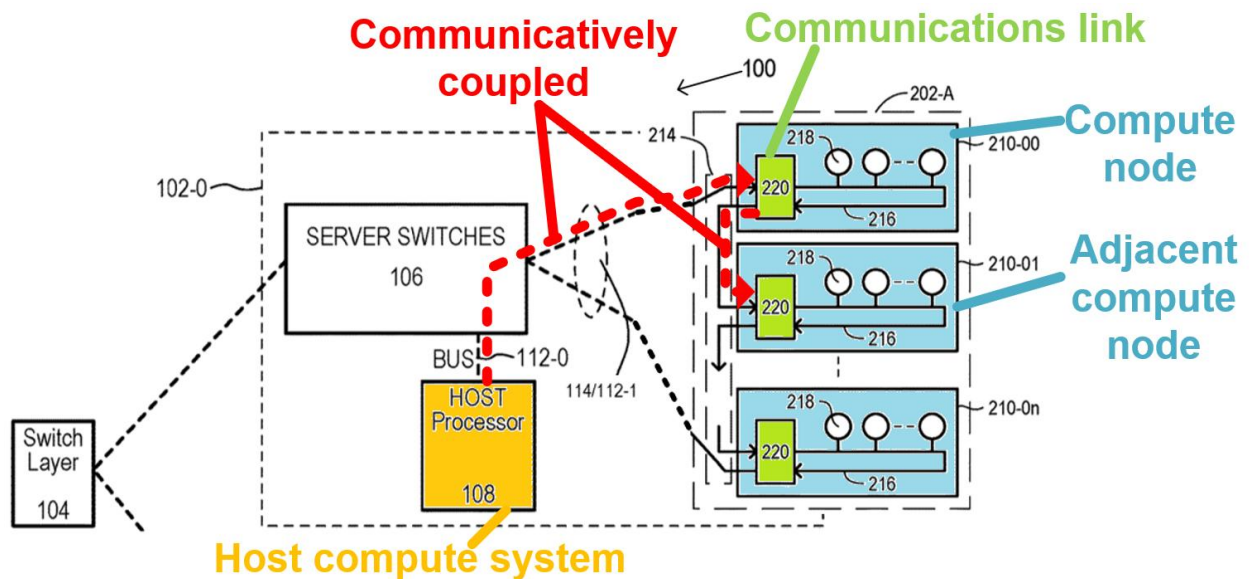
FIGS. 1–2C, 5–6, 12, 18, cl.1; *supra*, Element [1pre]. As shown in Dalal’s combination figure below, each server includes server switches 106 coupled to a host processor 108 (*host compute system*) and hwa modules 210 (*compute nodes*) via buses 112. AMD1004, [0031]–[0038], [0040], [0042], [0046], [0048],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

48. It is apparent in Dalal that the *plurality of compute nodes* are *serially connected* to one another in an *execution pipeline* because Dalal explains that within every 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).” AMD1004, [0042], *see also*, [0045] (“serial flow through multiple hwa modules (210-00 to -0n)”), [0047] (“processing by hwa modules in series”), [0035], [0036], [0046], FIGS. 1, 2B–2C. This is apparent in the combination figure below, which depicts each hardware accelerator module as being *serially coupled* to one or more adjacent hardware accelerator modules. 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], *see also*, [0035] (“*pipelined* processing...one stage of processing is passed on to the next stage”), [0045] (“processing *pipeline*”), [0049].



AMD1004, FIGS. 1, 2B Combination Figure (annotated).

49. 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. 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 “201-01” (*adjacent compute node*). AMD1004, [0032], [0035], [0040], [0042], [0048].



AMD1004, FIGS. 1, 2B Combination Figure (annotated).

(iii) *Element [1b]: a data store comprising records associated with a data processing operation;*

50. The Dalal-Ng Combination renders obvious [1b]. Dalal explains that every hardware accelerator module includes one or more “computing elements” that collectively execute a *data processing operation*. AMD1004, [0033] (“a number of computing elements...which can execute *operations* on received *data*”), [0035] (“such *processing* can be a pipelined processing”), [0037] (“a *processor* executing an *operation*”), *see also*, [0026]–[0029], [0034], [0036], [0040], [0042], [0046], [0075]–[0087], FIG. 7, FIGS. 1–2C, 7; *supra*, Element [1pre]. 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*, §VIII.C. More than one of the computing elements

within a hardware accelerator module (*compute node*) can have the architecture depicted in Figure 7, and an example of this is depicted in Dalal's Figure 8 which depicts more than one computing element having similar architecture. EX1008, [0089]–[0095], FIG. 8. It is not required for an execution pipeline of a hardware accelerator module to include more than one computing element, as Dalal explains that “a pipeline can be configured to use as few as one computing element of a hwa module.” EX1004, [0050]. Dalal explains that “[a] computation element...can include one or more processing circuits 721” that “run a program 736 for a networked application,” thereby confirming that each hwa module having one or more computing elements executes a *data processing operation*. AMD1004, [0076].

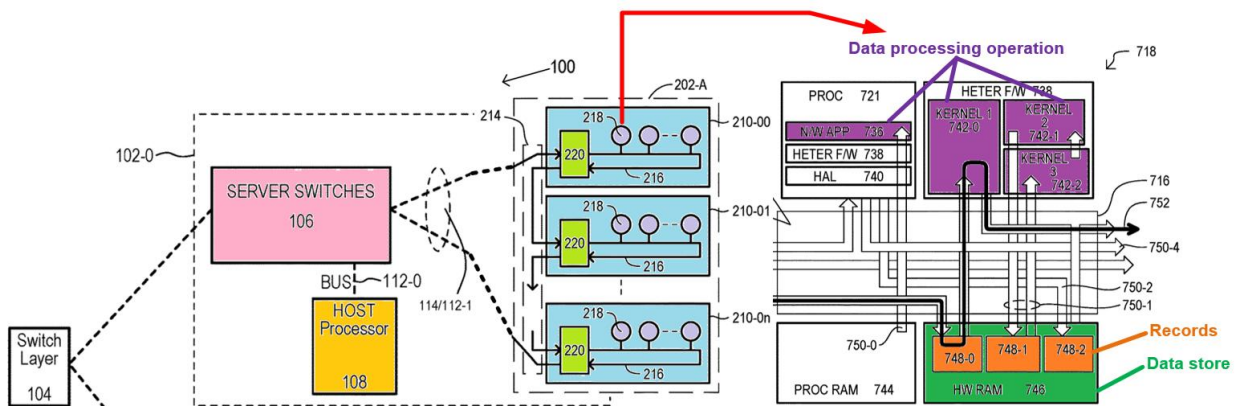


FIG. 7

AMD1004, FIGS. 1, 2B Combination Figure (annotated).

51. Dalal's processing circuits access *records* stored in predetermined memory spaces of a second memory 746 (*data store*) of each computing element to

run the program. AMD1004, [0081], *see also*, [0075], [0080]–[0082], [0085], [0094], cls.1–2. Each memory space provides a functional unit of memory for storing packet data, which represents a **record**. *Id.* A POSITA would have understood that this packet data represents a **record**, because packet data travels in a unit (i.e., a packet). 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**. AMD1004, [0075], [0080]–[0082], [0085], [0094].

52. Dalal expressly discloses that “data” is “processed” by the hardware accelerator modules in one or more “operations,” thereby supporting my opinion that the **compute node** executes a **data processing operation**. AMD1004, [0037] (“a processor executing an operation”), [0041] (“[d]ata processed by and/or generated within hwa module 210-00”), [0044] (“data processed”), [0048] (“data processed”), [0075] (“execute computation operations”), [0103] (“packet data processing”), [0119].

53. Every computing element within a hardware accelerator module includes “processing circuits” that “run a program...for a networked application,” further confirming that the hardware accelerator module executes a **data processing operation**. AMD1004, [0075]–[0078], FIG. 7. According to Dalal, this program “can be part of a machine learning library, such as TensorFlow...to enable machine

learning applications to run” and the program “can enable various execution processes...perform computation actions,” further confirming that the hardware accelerator modules execute a ***data processing operation***. AMD1004, [0078]. This is similar to a preferred embodiment of the ’636 patent whereby the execution pipeline implements an “inference algorithm for a neural network executing under TensorFlow™.” AMD1001, 47:46–48:7.

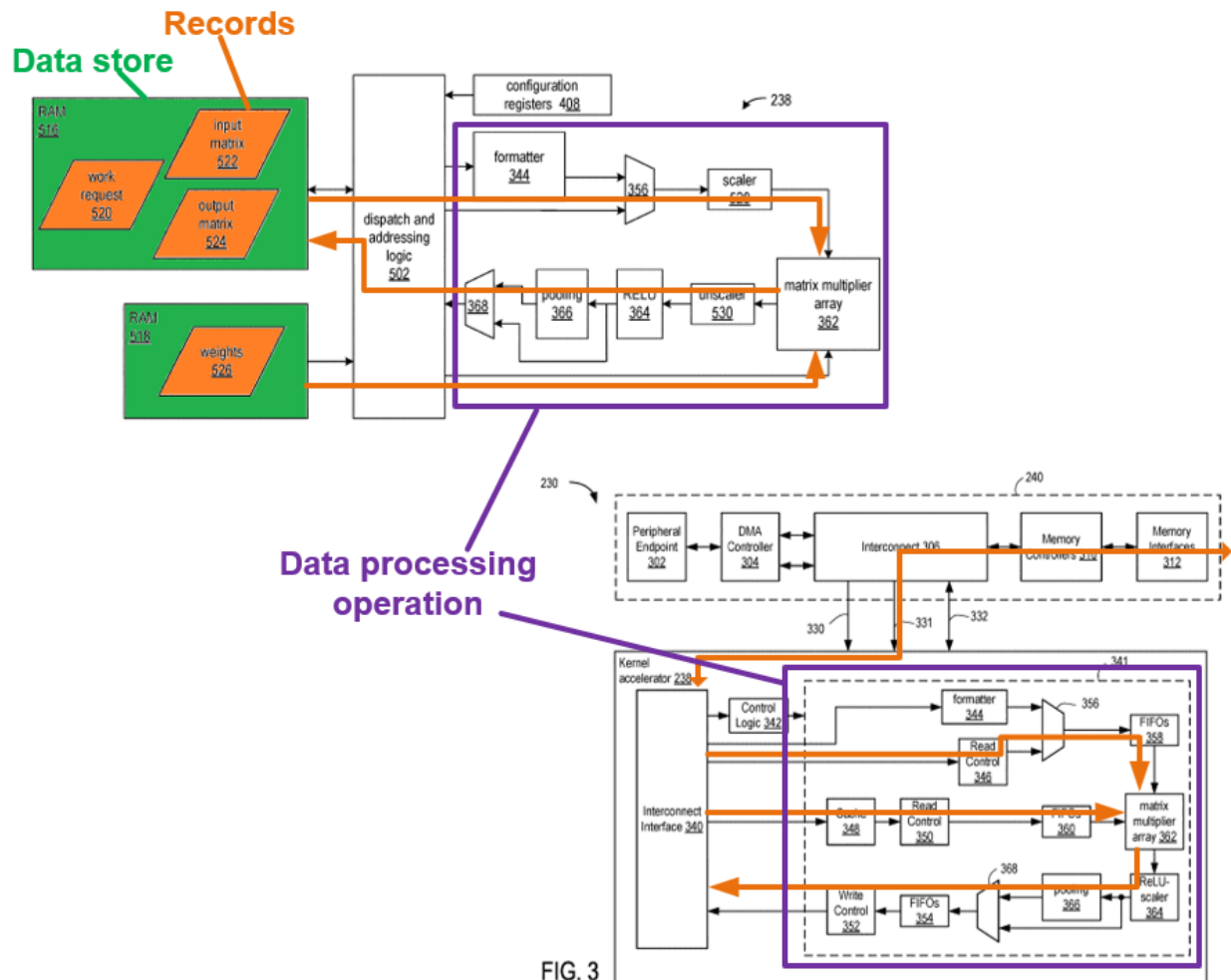
54. Dalal’s processing circuits access ***records*** stored in “predetermined memory spaces” in the “second memory” (***data store***) of each computing element to run the program for the networked application. AMD1004, [0081], *see also*, [0075], [0080], [0082], [0085], [0094], [0128], cl. 1, cl. 2. Each “memory space” provides a functional unit of memory for storing data (***record***). AMD1004, [0075], [0080]–[0082], [0085], [0094]; As described above, according to an IEEE technical definition, a “record” can include “[a] set of related data items treated as a unit,” which describes the data stored in Dalal’s predetermined memory spaces. AMD1030, 4. A POSITA would have also understood that second memory corresponds to the claimed ***data store*** comprising ***records*** because Dalal describes the second memory as “serving as ***data*** sources and sinks.” AMD1004, [0081], *see also*, cl. 1 (“at least one computing element is configured to transfer ***data*** to, or receive ***data*** from...the second memory circuits”), cl. 2 (“the second memory

circuits are configured to *store data* for processing by the at least one computing element”).

55. It is clear that the ’636 patent does not limit Element [1b] to any particular kind of memory or to any embodiment recited in the ’636 patent, and RAM is certainly not excluded from the scope of the Element [1b] *data store*. Multiple references from before the Critical Date use “data store” to refer to RAM or explain that data is “stored” in RAM. AMD1020, 11:1–11 (“data stored in RAM”); AMD1021, 12:46–63 (“data was stored” within a RAM). Dalal’s computing elements are not limited to using RAM as the second memory. AMD1004, [0033] (“volatile” or “nonvolatile” memory), [0067] (“module memories” of hardware accelerator modules “can be of any suitable type”), [0096], FIGS. 5, 9.

56. The Combination also incorporates Ng’s “RAM” (*data store*) that comprises *records* including matrices and work requests. AMD1005, [0041] (“[t]he input activation matrices can be read directly from the RAM...[w]eight matrices can be read from the RAM”), [0044] (“writes the weight matrices associated with the layers of the neural network to the RAM”), *see also*, [0007], [0008], [0024], [0032], [0034], [0036]–[0038], FIGS. 3–5. Ng’s *programmable* IC uses these *records* to perform a *data processing operation*, such performing matrix multiplication for each layer of a neural network using “input” and “weight” matrices stored in the RAM. AMD1005, [0041] (explaining that “the matrix multiplier array...performs

matrix multiplication operations for implementing a neural network” using the input activation matrix and the weight matrix stored in the RAM), [0052] (“the matrix multiplier 362 performs a matrix multiplication of matrices A1 and B1”), FIGS. 3–5.



AMD1005, FIGS. 3, 5 (annotated).

57. A POSITA would have found it obvious to implement Dalal’s hardware accelerator modules with the *data processing operation* described by Ng by storing *records* (e.g., matrices, work requests) in the memory spaces of Dalal’s second

memory, in part because Dalal's second memory is a convenient location to store data that is received from a previous node in the *execution pipeline*, processed, and passed to a next node in the *execution pipeline*. AMD1004, [0080]–[0081], FIG. 2B; AMD1005, [0024], [0047], [0094]; *supra*, §VIII.C. Ng's records would have been transmitted in the form of “packet data” as described by Dalal.

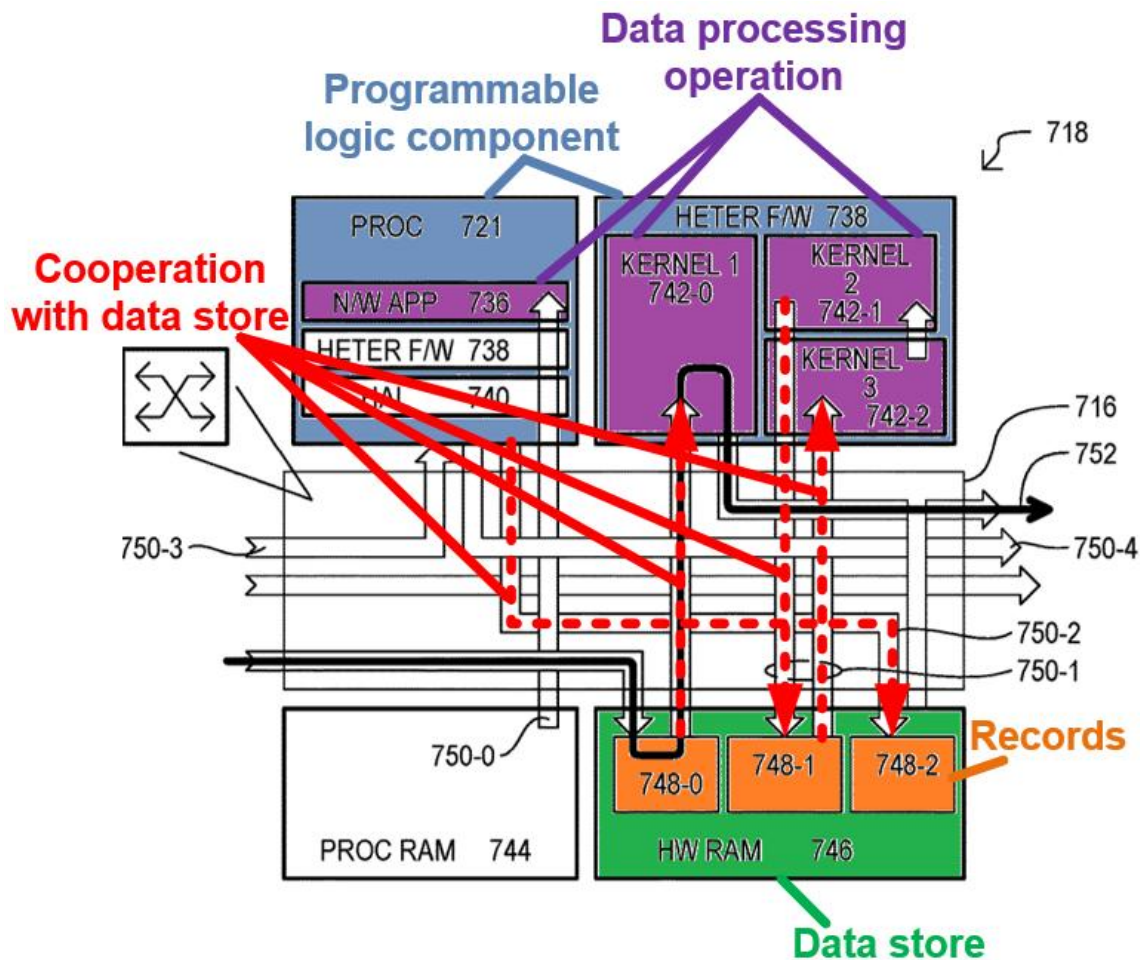
58. Ng explains that the results from the “previous layer” of the neural network are used as inputs for the “next layer” of the neural network in some cases, which is strikingly similar to Dalal's pipelined processing system where the output from a hardware acceleration module is input to the next module. AMD1004, [0042], [0045], FIG. 2B; AMD1005, [0024], [0047], [0051]; *supra*, §VIII.C. A POSITA would have been motivated to implement Dalal's pipelined processing system to perform the *data processing operation* of Ng for this reason, as Dalal offers serially coupled hardware accelerator modules for performing each step of a multi-stage process in order using the output from one stage as the input to the next stage.

59. The hardware accelerator of Ng is not limited to using volatile memory such as RAM. AMD1005, [0032], [0034], [0080], FIGS. 2, 8. For example, Ng discloses that the hardware accelerator includes “a non-volatile memory...and...RAM” in some embodiments and that the non-volatile memory

“can include any type...such as flash memory.” AMD1005, [0032], *see also*, [0034], [0080], FIGS. 2, 8.

(iv) *Element [1c]: a programmable logic component to execute the data processing operation in cooperation with the data store;*

60. The Dalal-Ng Combination renders obvious [1c]. The computing elements in Dalal’s hardware accelerator module each include “processing circuits” that collectively represent a *programmable logic component*. AMD1004, [0075], *see also*, [0076]–[0079], FIG. 7. For example, these processing circuits “can include an ARM type processor” that “can be integrated with other circuits” such as “*programmable logic*,” which supports my position that the processing circuits are a *programmable logic component*. AMD1004, [0077], *see also*, [0085] (“processing circuits...can be...ARM processors embedded with *programmable logic*”). Dalal explains that the processing circuits can include a field-programmable gate array (FPGA), which is a *programmable logic component*. AMD1004, [0065], [0070], [0087]. Also supporting my position that the processing circuits are a *programmable logic component* is that Dalal describes “*components*” of the “hwa module” as including the “processing circuits.” AMD1004, [0076], *see also*, [0081], [0082], [0096].



AMD1004, FIG. 4 (annotated).

61. 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] ("[p]rocessing circuits...can be configured to run a program"), [0078], [0079], FIG. 7. Furthermore, Dalal explains that the "programmable" logic of a data transfer fabric includes "connections" that are "configurable," which indicates that the *programmable logic* of the processing circuit is likewise programmable by

reconfiguring its internal hardware connections. AMD1004, [0081], *see also*, [0088] (“such linking can be reconfigured as needed for different applications”). My opinion here is supported by the fact that Dalal expressly discloses the “compute element 718 can be a single integrated circuit device, such as a computational FPGA” which has configurable hardware connections. AMD1004, [0087].

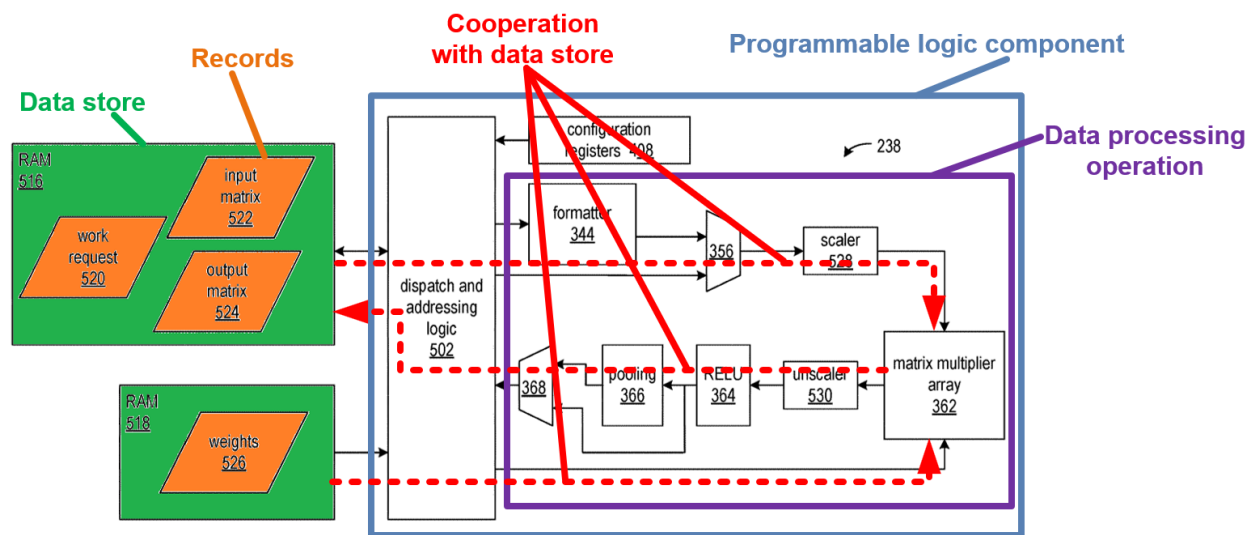
62. Dalal explains that a program run by the processing circuits “can be written on a framework” that “can enable various execution processes...to perform computation actions.” AMD1004, [0078], FIG. 7. According to Dalal, execution processes retrieve *records* (“packet data”) from the *data store* while running the program. AMD1004, [0078], [0081]; *supra*, Element [1b]. It is apparent, therefore, that the processing circuits *execute* the *data processing operation* in *cooperation* with the *data store*. AMD1004, [0078], [0081]; *supra*, Element [1b]. In describing interactions between the “compute array” and the “*data store*,” the ’636 patent explains that programmable logic manages “read/write or other access” to the data store. AMD1001, 16:1–7. Dalal’s cooperation is similar to this preferred embodiment of the ’636 patent because the processing circuits read and write data to the second memory. AMD1004, [0080]–[0082], FIG. 7.

63. As an additional and alternative basis for finding obvious Element [1c], the Combination also incorporates Ng’s supplemental teaching of a hardware accelerator including a *programmable logic component*. AMD1005, [0025]–

[0027], [0029]–[0039], [0042]–[0044], [0047]–[0051], FIGS. 1–5. Ng’s hardware accelerator is similar to the hardware accelerator modules of Dalal, because in both cases the accelerator includes programmable logic for performing a specialized task such as a machine learning algorithm, which would have motivated a POSITA to implement Dalal’s system with aspects of Ng’s hardware accelerator. AMD1004, [0076]–[0085]; AMD1005, [0025]–[0051]; *supra*, §VIII.C.

64. It is clear that Ng’s hardware accelerator includes “*programmable* integrated circuits (ICs)” that can include “field *programmable* gate arrays (FPGAs)” in some embodiments. AMD1005, [0026], *see also*, [0032] (the “hardware accelerator...includes a *programmable* IC” that “can be an FPGA”), [0034] (“the *programmable* IC...is configured with an acceleration circuit” that “can be implemented using a static region...and a *programmable* region” and “[t]he *programmable* region...can include one or more neural network accelerators”), [0035], FIGS. 2–3. Ng explains that internal hardware connections of this *programmable* IC can be configured (or reconfigured) to perform a specialized task, and this would have motivated a POSITA to implement Dalal’s hardware accelerator modules with Ng’s *programmable logic component*. AMD1005, [0027] (describing “programming the hardware accelerator” using “configuration bitstreams for FPGAs”), [0034] (“*programmable* IC...is configured with an acceleration circuit”), [0035] (“reconfigurations of the *programmable* region”), [0080] (“*programmable*

logic...includes *logic* cells...that can be configured to implement general logic functions”); *supra*, Element [1b] *supra*, §VIII.C. 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. Because Dalal and Ng both describe a hardware accelerator including *programmable logic*, a POSITA would have been motivated to implement Dalal’s execution pipeline comprising series-connected compute nodes with Ng’s *data processing operation* so that the processing circuits are programmed to perform the operation as suggested by Ng. *Supra*, §VIII.C.



AMD1005, FIG. 5 (annotated).

65. A POSITA would have understood at the time of the '636 patent that a *programmable logic component* such as Dalal-Ng's "FPGA" includes internal

hardware connections that are reprogrammable by configuration data (e.g., bitstreams), and multiple references from before the time of the '636 patent confirm this. AMD1027, 31 (“[a] programmable logic device contains a memory array that holds the configuration data that determines the programmed function of the device”); AMD1006, [0023]–[0030] (explaining that an FPGA includes “functional blocks” that having “programmable interconnect circuitry” therebetween).

- (v) *Element [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,*

66. The Dalal-Ng Combination renders obvious [1d.i]. As I described above in connection with my analysis of Elements [1b]–[1c], the processing circuits as described by Dalal include a *programmable logic component* that runs “programs” to perform a *data processing operation*. AMD1004, [0075], [0080]–[0082], FIG. 7; *supra*, Elements [1b]–[1c]. In each computing element of a hardware accelerator module, the programs are “loaded into a first *memory* 744” AMD1004, [0075], [0079]–[0082], FIG. 7. It is apparent that the first memory of each computing element within a hardware accelerator module (*compute node*) collectively represent a *node memory*. AMD1004, [0075], [0079]–[0082], FIG. 7. As depicted below in Dalal’s Figure 7, the programs support *operation of the programmable logic component in connection with the data processing operation*,

in part because the programs are executed “by processing circuits” to perform execution processes and computation actions. AMD1004, [0078], [0079], *see also*, [0076] (“written on a framework...for execution across heterogenous platforms”), [0077], FIG. 7 (referring to execution processes as “kernel 1” through “kernel 3”); *supra*, Element [1d]. My opinion that that the programs support *operation* of a *programmable logic component* in connection with a *data processing operation* is supported by the fact that Dalal expressly discloses that a “computation element” of the hardware accelerator module “can execute computation operations.” AMD1004, [0075].

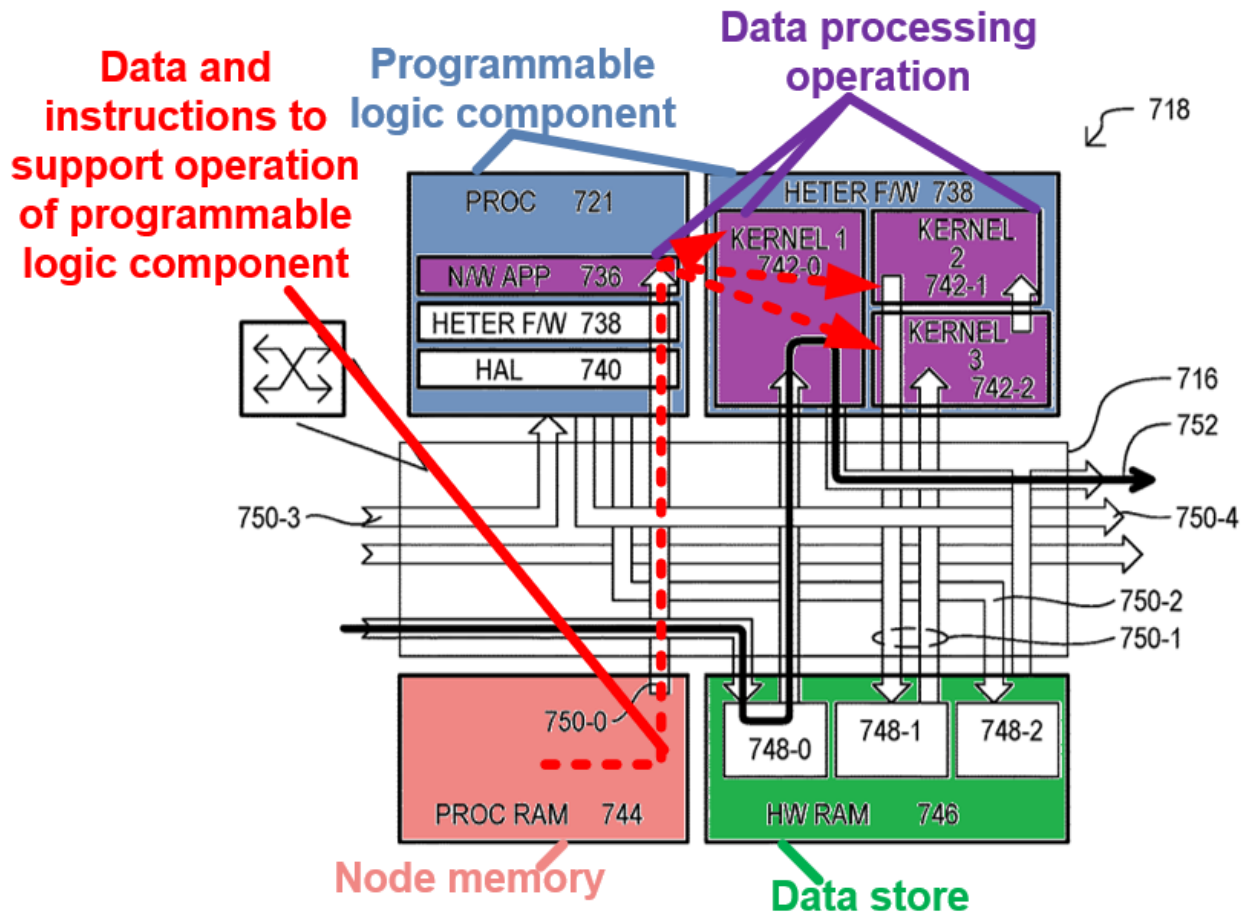


FIG. 7

AMD1005, FIG. 7 (annotated).

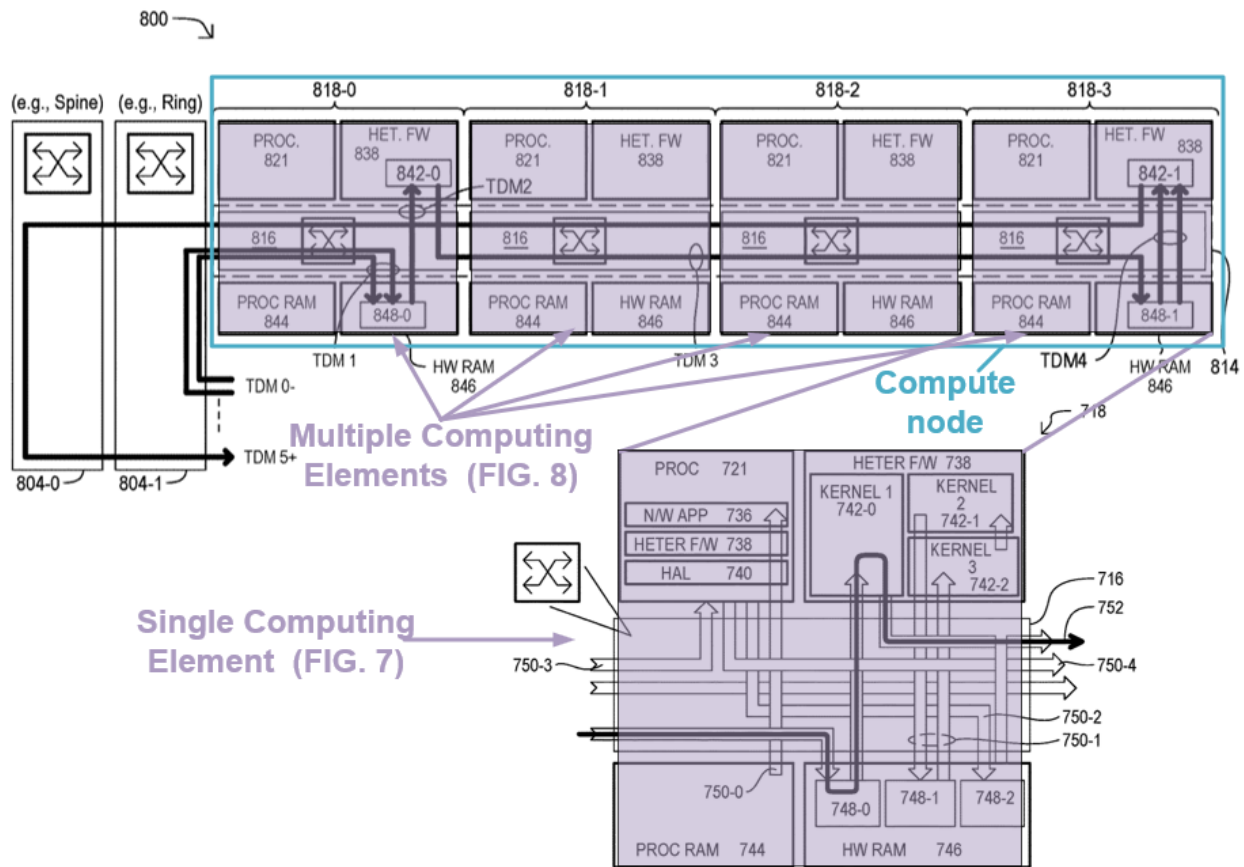
67. The first memory is “store[s] *instructions* for execution by the processing circuits,” meaning that these *instructions* are *supporting operation* of the *programmable logic component*. AMD1004, cl. 11. Dalal also explains that a data transfer fabric 716 provides “*data* transfer paths between components” including connections between the processing circuits and first memory , confirming that *data* is stored in the first memory and transferred over the data transfer paths. AMD1004, [0082], *see also*, cl. 1 (“a data transfer fabric configured to enable data

transfers between the processing circuits and the first and second memory circuits”), [0081], [0083], FIG. 7. It is clear that one of these paths “750-0” transfers **data** and **instructions** from the first memory (**node memory**) to the processing circuits (**programmable logic component**). AMD1004, [0079] (“processing circuits...can access first memory...through data transfer fabric 716 (path shown as 750-0)”).

68. 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]. The system in the Dalal-Ng Combination renders obvious **a node memory comprising data and instructions to support operation of the programmable logic component in connection with the data processing operation** at least because the **data and instructions** stored in the first memory are used by the processing circuits (**programmable logic component**) to execute operations. The ’636 patent acknowledges that storage of these **data** and **instructions** is “generally known,” and this supports my opinion that Element [1f] would have been obvious over Dalal and Ng. AMD1001, 18:6–10.

69. I will now turn my analysis to the **inter-node communications** aspect of Element [1d.i]. As I 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*, [1pre], [1b], FIG. 7. Figure 8 depicts

an hwa module having multiple computing elements 818-0 to -3 that each “take the form” of the single computing element depicted in FIG. 7, which indicates that each one of Dalal’s computing elements can be represented by the single computing element of Dalal’s FIG. 7. AMD1004, [0089], [0090], FIGS. 7–8.



AMD1004, FIGS. 7–8 (annotated).

70. FIG. 8 depicts the understanding a POSITA would have had of a computing element 718, as shown in FIG. 7, in connection with other similar computing elements, which is consistent with the connection of computing elements shown in Dalal’s FIG. 2B. *See* §VIII.D, Element [1b]. Dalal explains that the

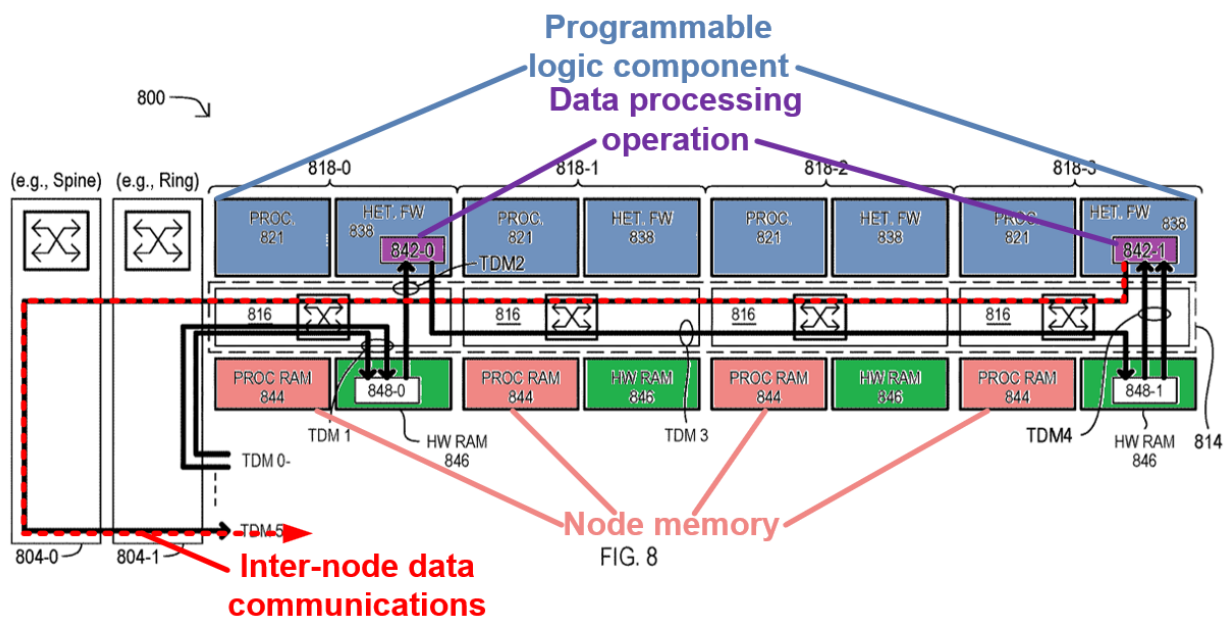
computing elements 818-0 to -3 of the hardware acceleration module are connected to other parts of a system over a first connection layer 804-0 that couples hwa modules of the same server, which is the same architecture of Dalal's FIG. 2B where an hwa module 210-00 sends packet data to the next hwa module 210-01 in the same server. AMD1004, [0042], [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," which supports my opinion that Dalal's FIG. 8 describes the architecture of FIG. 2B. AMD1004, [0092]. That is, FIG. 8 supports other parts of Dalal's disclosure (including FIG. 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]. 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].

71. In Dalal's FIG. 8, the "computing elements (818-0 to -3)" can perform a "pipelined operation" having multiple stages, which is consistent with Dalal's general description of pipelined processing. AMD1004, [0094], *see also*, [0027] ("processing pipelines...can be physical pipelines, with data flowing from one computing element to the next"); [0036] ("packet data can then travel...within the destination hwa module...to an element...packet data can be processed in a

pipelined fashion, traveling to subsequent elements”). One of these stages includes “data in memory space 848-0” being “read out by execution process (e.g., kernel) 842-0 of computing element 818-0” and “a result from execution process 842-0” being “stored at memory space 848-1 in second memory 846.” AMD1008, [0094]. Next, Dalal explains that 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 intermodule fabric) according to Dalal. Execution process 842-1 “operate[s] according to a same network application” corresponding to the “programs” of Dalal’s FIG. 7 which are run by the *programmable logic component* and stored in the *node memory*. AMD1004, [0095], *see also*, [0076]–[0088], FIGS. 7–8.

72. A POSITA would have understood that execution process 842-1, like the execution processes 742-0 to -2 of FIG. 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 FIG. 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

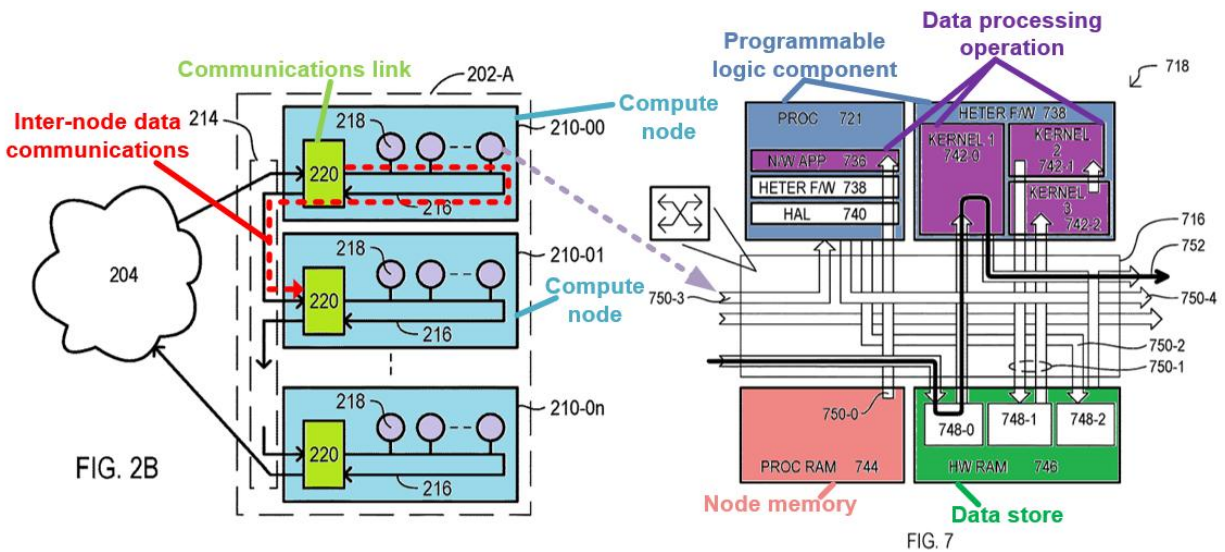
compute node in the *execution pipeline*. *Id.* A POSITA would have understood that the *data* and *instructions* stored in Dalal’s *node memory* facilitate *inter-node data communications* by causing execution process 842-1 to forward a result to another hwa module (*compute node*) in the same server because Dalal explains that the first connection layer 804-0 includes “direct inter-module connections” to hwa modules “of a same server.” *Id.*



AMD1004, FIG. 8 (annotated).

73. 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. For example, using Dalal’s *programmable logic device* to run the program stored in the node memory causes data to (1) flow through multiple computing elements within the hardware accelerator module, and (2) flow to a next hardware accelerator module in the *execution pipeline* via the *communications link*. AMD1004, [0035], [0036], [0042], [0045], [0075]–[0078], FIGS. 1, 2B–2C, 7. By running the program stored in the *node memory*, the hardware accelerator module processes received packet data and outputs the packet data to a subsequent hardware accelerator component in the execution pipeline, thereby facilitating *inter-node data communications* via the *communications link*. The *data* and *instructions* stored in the “hwa module 210-00” *node memory* facilitate *inter-node data communications* by advancing the packet data through the processing pipeline to be forwarded because Dalal explains that packet data is “processed by one or more elements 218 in hwa module 210-00” and subsequently “forwarded to a next hwa module 210-01.” AMD1004, [0042].



AMD1004, FIGS. 2B, 7 (annotated).

- (vi) *Element [1d.ii]: wherein the node memory further comprises algorithm memory including resource-specific memory locations to support the data processing operation;*

74. The Dalal-Ng Combination renders obvious 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]. 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]. The program "can be

any suitable program, but in particular embodiments can be part of a machine learning library, such as TensorFlow.” *Id.*

75. 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 (“algorithm memory 1841 may be implemented as pre-allocated addresses, blocks, or sectors of node memory 144... algorithm memory 1841 may generally comprise resource-specific data storage cells or memory locations”). Dalal expressly discloses that the “first memory[*node memory*]...can be...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*. AMD1042, p.3 (“explaining that “SDRAM devices contain a...storage array...[e]ach device consists of typically eight or 16 banks that can work in parallel...[a] bank consists of a memory array, divided into rows, each row containing a certain number of columns); AMD1045, 3:64–4:9 (describing techniques for “allocating” data among “first and second banks” of SDRAM”).

76. 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]. Buffers such as these that are allocated for specific data are *resource-specific memory locations* akin to the '636 patent. AMD1001, 40:25–41 (describing "buffers" and "tables" as examples of "resource-specific...memory locations"). In the Dalal-Ng Combination, Dalal's first memory (*node memory*) would have been implemented with one or more buffers (*resource-specific memory locations*) that are pre-allocated at given addresses to assist the processing circuits in accessing information relating to "computation actions called by a program" stored in the first memory. AMD1006, [0078]; *supra*, §VIII.C.

77. According to the '636 patent, the *algorithm memory* can be a part of the *node memory* that stores *data* and *instructions* for supporting *operation* of the *programmable logic component* in connection with the *data processing operation*, which is similar to Dalal's first memory that stores programs (*data* and *instructions*) for supporting *operation* of the processing circuits (*programmable logic component*). AMD1001, 40:25–41 ("algorithm memory...may generally be operative to store data and instructions necessary or desirable...to perform write/read/execute operations in support of programmable logic").

- (vii) *Element [1e]: a data mover component to facilitate intra-node data communications between the programmable logic component and the node memory; and*

78. The Dalal-Ng Combination renders obvious [1e]. Dalal explains that a data transfer fabric 716 provides data transfer paths that represent the claimed *data mover component*. AMD1004, [0082], *see also*, [0033]–[0036], [0040]–[0043], [0075], [0079], [0081]–[0089], FIGS. 1–2C, 7, 8; *supra*, [1d.i]. For example, Dalal’s processing circuits (*programmable logic component*) can access the first memory (*node memory*) through the data transfer fabric via the path 750-0, indicating that the part of the fabric providing the path 750-0 is a *component* that moves *data*. AMD1004, [0079], FIG. 7 *see also*, cl.1, FIG. 7. It is apparent that path 750-0 provides a *data mover component* that facilitates *data communications* between the *programmable logic component* and the *node memory* by allowing the processing circuits to access the programs in the first memory to run these programs. *Id.* Dalal also explains that the “data transfer fabric” is “configured to enable *data* transfers between the processing circuits and the first...memory circuits,” further confirming that the path 750-0 facilitates *data communications*. *Id.*

79. These *data communications* are *intra-node* because the first memory and the processing circuits are located within the same hardware acceleration module and therefore within the same *compute node*. AMD1004, [0034], [0075]–

[0088], FIG. 7. Dalal expressly describes the data transfer fabric as being an “*intra-module fabric*,” further confirming that the fabric supports *intra-node data communications* within the same *compute node*. AMD1004, [0034].

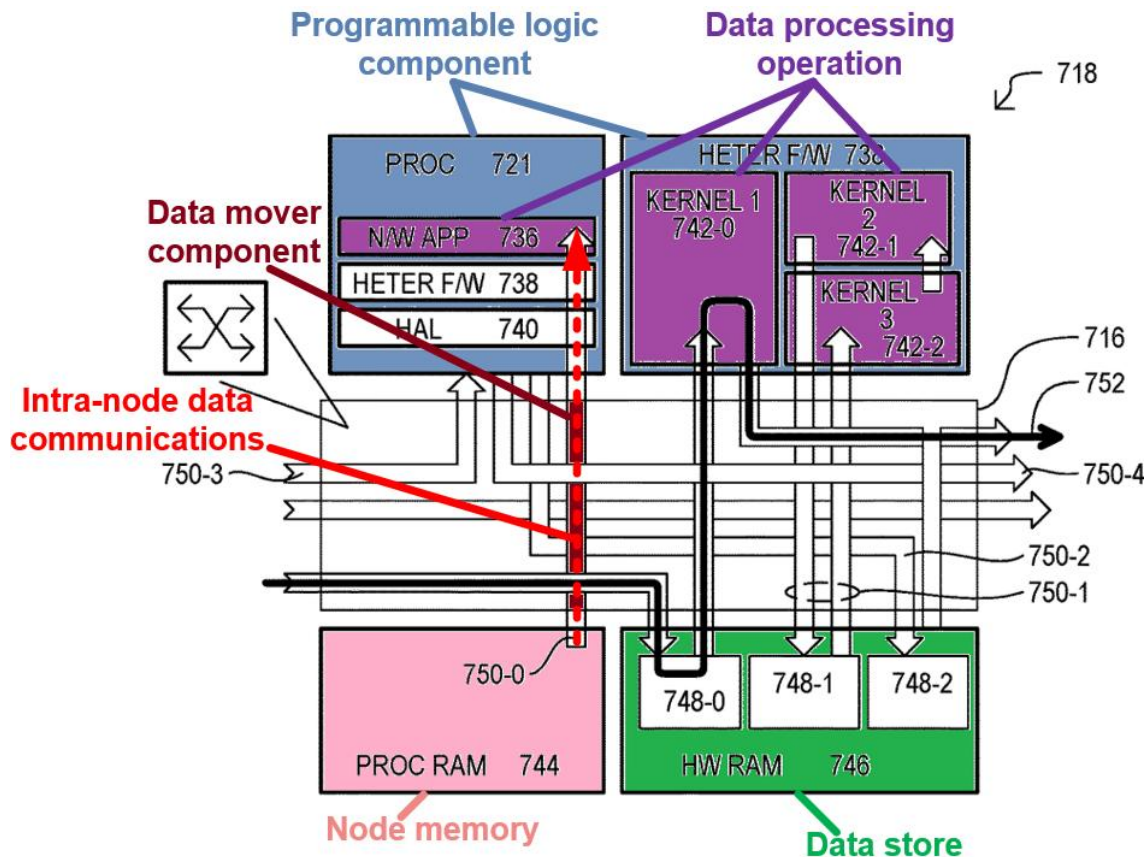


FIG. 7

AMD1004, FIG. 7 (annotated).

80. Dalal explains that the data transfer fabric can be “programmable” and “can be formed by buses on a programmable logic device, such as an FPGA.” AMD1004, [0084]–[0086], *see also*, [0087] (explaining that the entire compute element including the fabric can be “a single integrated circuit device, such as a computational FPGA”), FIG. 7. The fact that the data transfer fabric can be formed

in programmable logic provides further support that the fabric represents a *data mover component*, because the data mover apparatus (DMA) of the '636 patent is located fully within the “programmable logic block” of a compute node. AMD1001, 15:55–16:17, 16:45–61. As I described above, Dalal’s data transfer fabric is “configured and operative to facilitate data communications,” much like the data mover component of the '512 patent. AMD1001, 3:5–25. It was well known by the time of the '636 patent for a *data mover component* to be included in programmable logic. AMD1025, p32 (“[t]he AXI DataMover...can be optionally programmed...to provide a reduced resource utilization footprint in the target FPGA”).

(viii) *Element [1f]: a storage interface component to facilitate intra-node data communications between the programmable logic component and the data store;*

81. The Dalal-Ng Combination renders [1f] obvious (1) based on Dalal’s teachings alone, and (2) based on a combination of Dalal-Ng, as I have explained in more detail below.

82. First, *data communications* occur between the *programmable logic component* and the *data store* of the same *respective compute node* (i.e., the communications are *intra-node*), as indicated by the fact that Dalal’s second memory 746 (*data store*) is accessed by various computing circuits including the processing circuits, 721, 738 (*programmable logic component*). AMD1004, [0078]–[0082], FIG. 7. The computing element includes data transfer paths (*storage*

interface component) that *facilitate intra-node data communications* through Dalal's data transfer fabric 716 between the *programmable logic component* and the *data store*. AMD1004, [0082], [0081] ("memory spaces...serve to provide connections between...components"), [0078]–[0080], FIG. 7.

83. As I described above in my analysis of Element [1d], the second memory 746 (*data store*) within each computing element of the hardware accelerator module includes "predetermined memory spaces" storing *records* accessed by a *programmable logic component* when executing a program. AMD1004, [0078], [0080]–[0082]; *supra*, Element [1d]. Therefore, to retrieve the *records* stored in the *data store*, it is apparent that *intra-node data communications* occur between the *programmable logic component* and the *data store*. AMD1004, [0080], *see also*, [0078], [0081], [0082], FIG. 7.

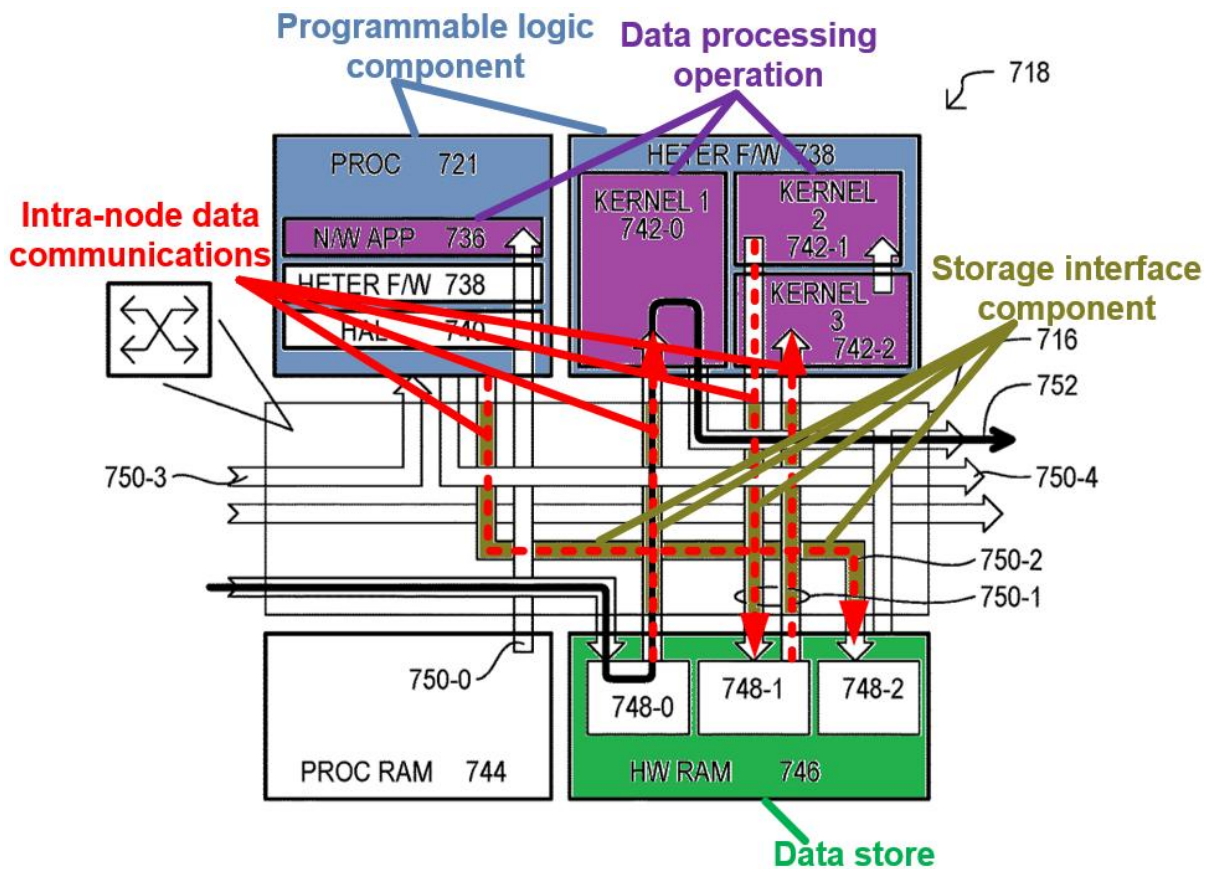


FIG. 7

AMD1004, FIG. 7 (annotated).

84. The Combination also incorporates Ng’s express teaching of “memory *interfaces*.” AMD1005, [0035]–[0038], FIG. 3. Ng explains that the memory interfaces are coupled to a RAM and allow the neural network accelerator to read from the RAM and writ” to the RAM. AMD1005, [0035], [0036], [0038], [0040], [0041], FIG. 3. The memory interfaces therefore facilitate *intra-node data communications* between the RAM and the neural network accelerator (*programmable logic component*). *Id.*

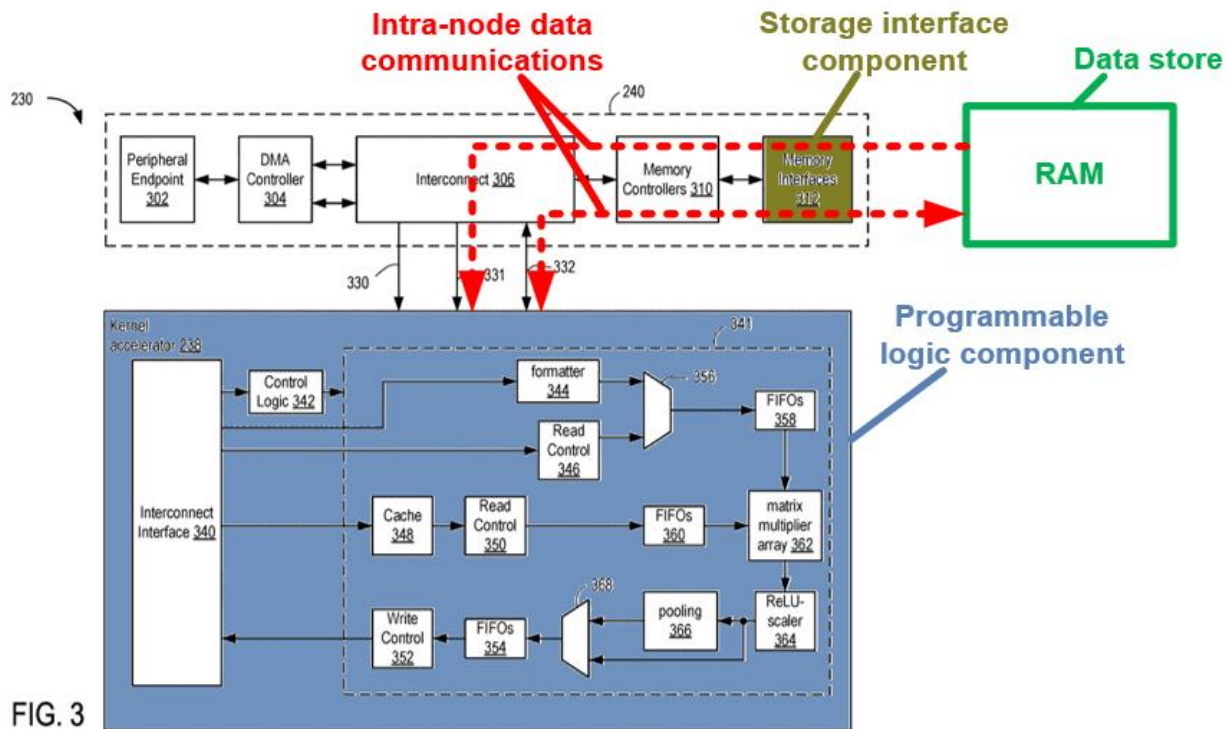


FIG. 3

AMD1005, FIG. 3 (annotated).

85. To provide dedicated circuitry configured with one or more memory protocols of the data store (e.g., DDR RAM), 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. AMD1004, [0080] ("second memory...accessible by programmable logic (FPGA), such as static RAM (SRAM)"), [0085]; AMD1005, [0032], [0035], [0043]; *supra*, §VIII.C. The computing element in the Dalal-Ng Combination 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]. This is depicted below in a modified version of Dalal's Figure

7.

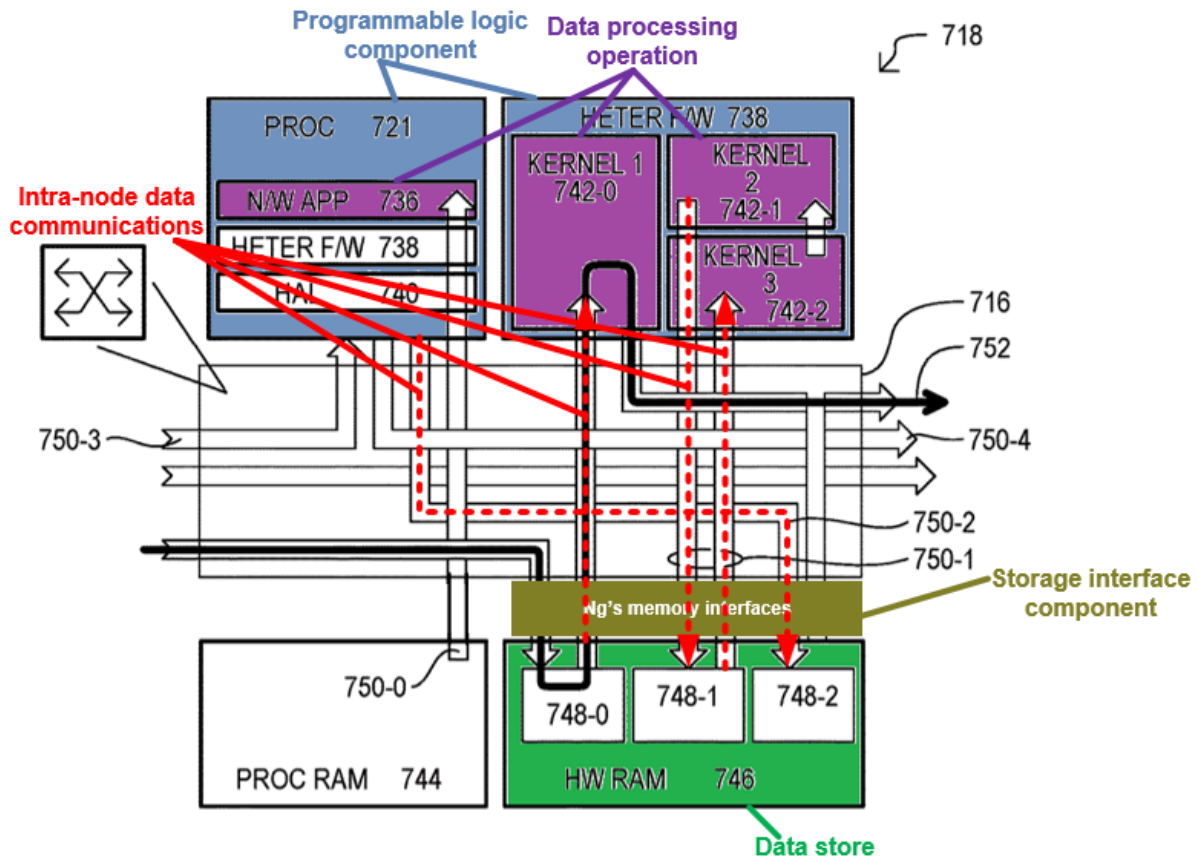


FIG. 7

AMD1004, FIG. 7 (modified & annotated).

86. The '636 patent describes a *storage interface component* as including components interchangeably referred to as a “memory interface” and a “data store interface” that facilitate communication between the *programmable logic component* and the *data store*. AMD1001, 24:59–25:17 (describing “storage interface component” including “memory interface”), 15:31–57 (including “data

store interface”), and Ng’s “memory interfaces” are similar to this description. AMD1001, 25:50–26:5 (describing “storage interface component” including “memory interface”), 16:18–44 (including “data store interface”).

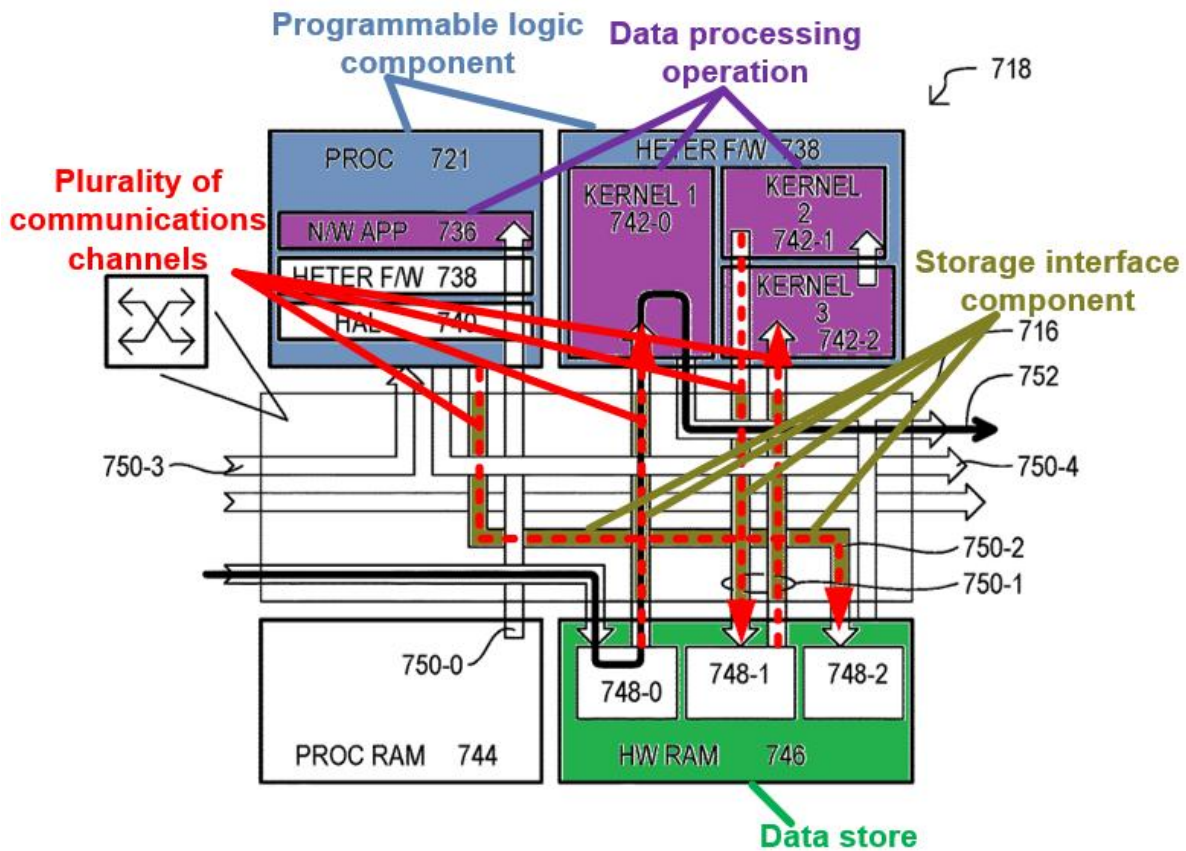
87. The memory interfaces of Ng therefore provide the claimed *storage interface component* to facilitate *intra-node data communications* between a *programmable logic component* and *data store*. A POSITA would have been motivated to implement Dalal’s hardware accelerator modules to include memory interfaces between the processing circuits and the second memory as suggested by Ng, because doing so would have enabled the modules to manage communications between programmable logic and memory efficiently according to protocols that are compatible with the respective components. The ’636 patent acknowledges that a “data store interface” that “manages access” to a “data store” is “generally known in the art,” which supports my argument that the Dalal-Ng Combination renders the *storage interface component* obvious. AMD1001, 11:19–21, *see also*, 16:37–44.

(ix) *Element [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*

88. The Dalal-Ng Combination renders [1g] obvious (1) based on Dalal’s teachings alone, and (2) based on a combination of Dalal-Ng.

89. First, as I described above in connection with my analysis of Elements [1e]–[1f], Dalal’s data transfer fabric provides different data transfer paths that

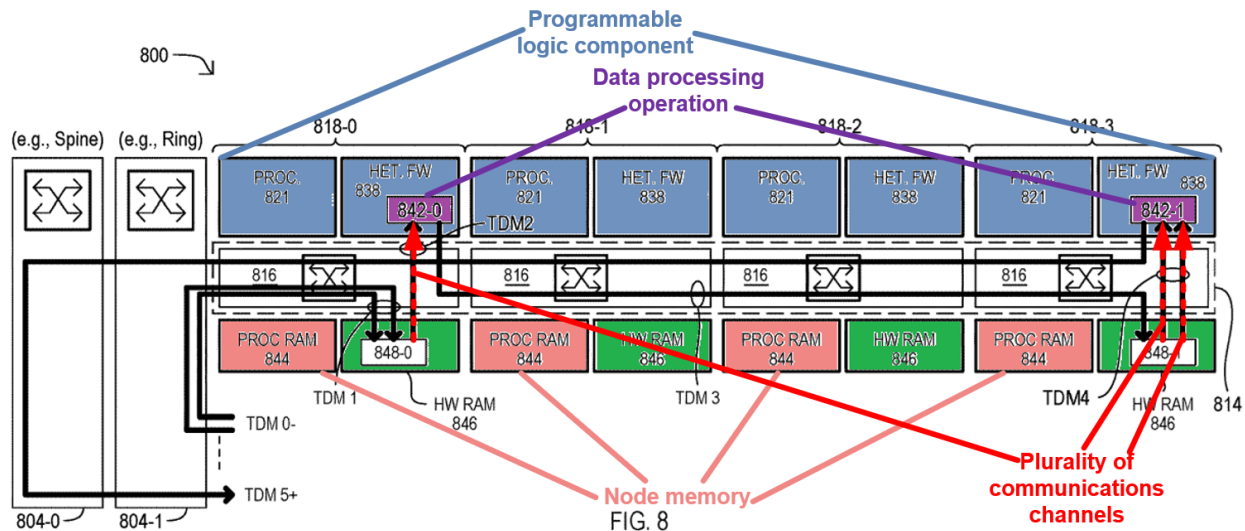
include “connections between...processing circuits 721 (*programmable logic component*)...[and] second memory 746 (*data store*).” AMD1004, [0082], *see also*, [0081] (“various possible connections”), [0084] (“providing connections as needed”), FIG. 7; *supra*, Elements [1e]–[1f]. As depicted below in 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*), including data transfer paths labeled “750-1,” and “750-2” in Figure 7. *Id.*



AMD1004, FIG. 7 (annotated).

90. Figure 8 of Dalal also depicts a *plurality of communications channels* that *transfer data* between memory spaces in the second memory (*data store*) and processing circuits (*programmable logic component*) within a hardware accelerator module. AMD1004, [0094] (“[a]t stage TDM 2, data in memory space 848-0 can be read out by execution process (e.g., kernel) 842-0 of computing element 818-0... [a]t stage TDM 4, execution process 842-1 can read data from memory space 848-

1”), [0095], FIG. 8. These amount to a *plurality of communications channels* that *transfer data* between the *programmable logic component* and the *data store*.



AMD1004, FIG. 8 (annotated).

91. Second, the Combination also incorporates Ng’s express teaching of multiple “memory interfaces.” AMD1005, [0036], [0038]. Ng’s disclosure supports my opinion that the *storage interface component* utilizes a *plurality of communications channels* to transfer data between the *programmable logic component* and a memory, because the memory interfaces include a “first one” that is used to read from the RAM and a “second one” that is used to read from and write to the RAM. AMD1005, [0038], *see also*, [0035], [0036]. It is clear that two separate and distinct interfaces between a programmable logic component and a memory represent *plurality of communications channels*. In the combination, the *storage interface component* utilizes at least one communication channel for

reading, and at least one communications channel for reading and writing. *Supra*, §VIII.C. The modified version of Dalal's Figure 7, below, demonstrates that Ng's memory interfaces would have utilized a ***plurality of communications channels*** that ***transfer data*** between the processing circuits and the second memory.

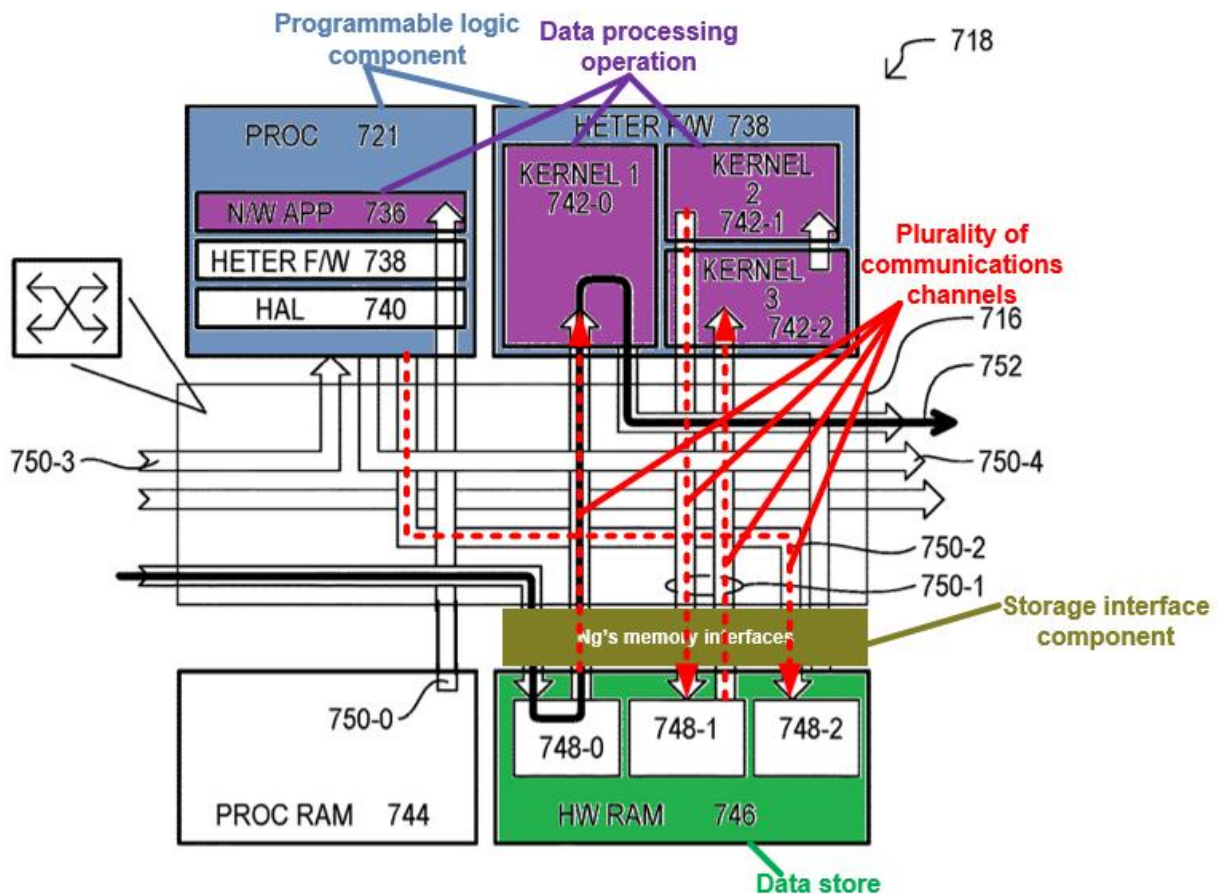


FIG. 7

AMD1004, FIG. 7 (modified & annotated).

92. Multiple references from before the time of the '636 patent reveal that a ***storage interface component*** including a ***plurality of communications channels*** were well known in the context of hardware accelerators. AMD1028, 8:4–24

(describing a plurality of “lanes” to connect to a flash backbone); AMD1043, [0039] (describing multiple “NAND interfaces” for accessing “sets” of NAND flash devices).

- (x) *Element [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.*

93. The Dalal-Ng Combination renders obvious Element [1h]. Dalal’s *data processing operation* involves “execution processes” that perform *computation* actions called by a program. AMD1004, [0078], *see also*, [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]. 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], *see also*, [0115] (“data-intensive computing”); AMD1002, 163 (Applicant asserted that an “example” of “computationally expensive algorithms” includes “Big Data or artificial intelligence applications”); *supra*, §V.B.

94. Dalal's execution pipeline represents a ***distributed processing compute environment*** that executes a 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) both perform computation actions corresponding to the operation. AMD1004, [0078], *see also*, [0042] ("pipelined processing can be scaled up within a server by use of additional hwa modules"), [0043] ("different hwa modules...can be in communication via data path 214"). Furthermore, Dalal explains that data is transferred in packet form, which is a characteristic of a ***distributed processing compute environment***. AMD1004, [0034]–[0045]. 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], *see also*, [0115], [0126], [0174]. Therefore, Dalal's compute node is deployed in a distributed processing compute environment to execute the computationally expensive algorithm.

2. Claim 2

- (i) ***Element [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.***

95. The Dalal-Ng Combination renders obvious [2]. According to Dalal, server switches and hardware accelerator modules are connected by buses that serve as a fabric. AMD1004, [0032] (“[s]erver switches 106 can be connected to host processor 108 and hwa modules (110-00 to -*mn*) by one or more buses 112-0/1”), [0033], [0034] (“a bus 112-1 that connects hwa modules (110-00 to 110-0*n*) of a same server 102-0 can serve as a fabric 114”), [0035], [0036], [0043], [0064], [0089]–[0099], [0123], [0170], FIGS. 1, 2B, 8, 12, 18; *supra*, Elements [1b]–[1c]. Because the fabric is connected to the switching circuit (*communications link*) of each hwa module 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 the hwa module (e.g., 210-00) (*compute node*) employs a *serial communications protocol* that enables *bi-directional data communication*. AMD1004, [0032], [0040]–[0048], FIGS. 1, 2B; *supra*, [1c].

96. Multiple references from before the Critical Date of the ’636 patent confirm that PCIe is a *serial communications protocol* that enables *bi-directional data communication*. AMD1018, p.1 (“Peripheral Component Interconnect Express (PCIe) is a *serial* expansion bus standard”), p.3 (“The bandwidth of the PCI Express lane is about the 250MB/s in each direction”); AMD1023, p.1 (“PCIe...is an industry standard serial bus protocol that offers high bandwidth communication

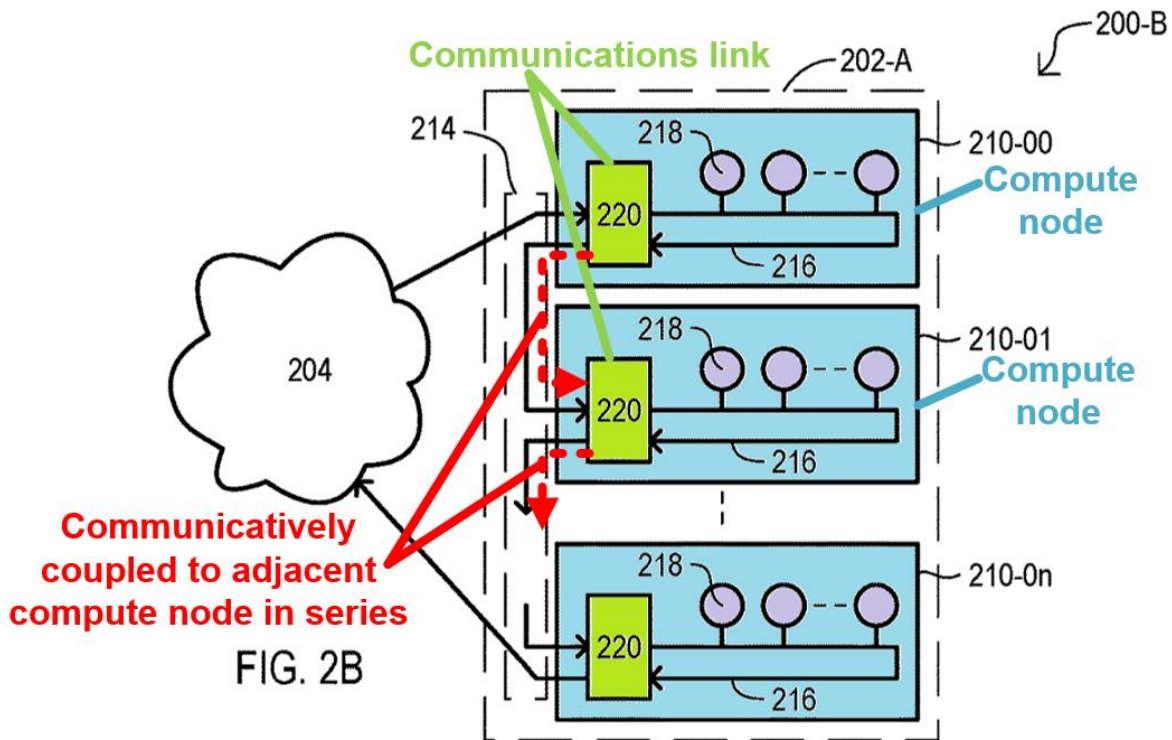
link”); AMD1026, p.27 (“PCI Express Link consists of two, low-voltage, differentially driven signal pairs: a transmit pair and a receive pair”); AMD1008, 3:30–55 (“a lane that supports...serial data communication...permit **bi-directional communication** between the PCIe devices”). This is consistent with my understanding of PCIe by the time of the ’636 patent.

97. Ng’s disclosure further reinforces the obviousness of using a **serial communications protocol** that enables **bi-directional data communication** such as PCIe. AMD1005, [0030]–[0043], FIGS. 2–4; *supra*, §VIII.C. For example, the combination also incorporates Ng’s supplemental teaching of a “peripheral bus” connected to a hardware accelerator that “can implement an expansion bus standard, such as...PCIe.” AMD1005, [0030], *see also*, [0036] (describing a “PCIe endpoint” of a hardware accelerator that “provides a physical interface to” the peripheral bus), [0043], FIGS. 2–4. Ng therefore describes a hardware accelerator comprising **communications link** that employs a **serial communications protocol** that enables **bi-directional data communication** and a POSITA would have been motivated to implement Dalal’s switching circuits with Ng’s PCIe endpoint to ensure that Dalal’s hwa modules connect seamlessly with the PCIe bus. *Supra*, §VIII.C.

3. Claim 3

- (i) ***Element [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.***

98. The Dalal-Ng Combination renders obvious [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]. As depicted below in Dalal’s FIG. 2B, this communication occurs via the switching circuit (*communications link*) of the hwa module 210-00” (*compute node*). *Id.* This is evidenced by the fact that Dalal expressly discloses communication between compute nodes. AMD1004, [0042] (“[r]eceived packet data can be processed by...hwa module 210-00...forwarded to a next hwa module 210-01...such processing can continue through multiple hwa modules”), [0043] (“hardware acceleration modules “can be in communication via data path 214”), *see also*, [0047] (“hwa modules of same server can be in communication via a data path”), [0094]. The fact that Dalal explains that these *communicatively coupled* modules are *in series* with one another supports my position here. AMD1004, [0045] (“serial flow”), [0047] (“processing...*in series*”), *see also*, [0040]–[0044], [0048], FIGS. 2A–2B. My opinion is also supported by Dalal’s Figure 2B, which plainly illustrates modules connected *in series*, the output from a first module 210-00 being passed to the next modules 210-01 through 210-0n sequentially in order. AMD1004, FIG. 2B.



AMD1004, FIG. 2B (annotated).

4. Claim 6

- (i) *Element [6]: The compute node of claim 1 wherein the programmable logic component comprises a field programmable gate array (FPGA).*

99. Dalal's hwa modules have a series of computing elements, each including processing circuits that comprise a *programmable logic component*. AMD1004, [0077] (explaining that processing circuits include logic "derived from...*programmable* circuits"), *see also*, [0033] ("*programmable* logic circuits"), [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 the 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], *see also*, [0032] (“[t]he hardware accelerator 116 includes a *programmable* IC 228...[t]he *programmable* IC 228 can be an *FPGA*”), [0058], [0081] (“an *FPGA* implementation of the *programmable* IC”), [0082]–[0086], FIGS. 2–3, 9; *supra*, §V.A.3.

100. 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], *see also*, [0033], [0065], [0080], [0096], [0104]. Furthermore, the second memory of each computing element includes “relatively fast memory *accessible by programmable logic (FPGA),*” which supports my position that it would have been obvious to implement Dalal’s processing circuits with an FPGA because the processing circuits access the second memory. AMD1004, [0080]. A POSITA would have understood that it is beneficial to implement Dalal’s processing circuits with an FPGA as suggested by Ng, in part because Ng expressly discloses that using hardware accelerators “can be advantageous because an FPGA circuit can process data faster” than other kinds of

processors “and consume less power in the process.” AMD1005, [0006]; *supra*, §VIII.C.

5. Claim 7

- (i) *Element [7]: The compute node of claim 1 wherein the compute node further comprises a processor to manage operation of the data mover component.*

101. The Dalal-Ng Combination renders obvious [7]. As I described above in my analysis of Element [1e], Dalal’s processing circuits within each hardware accelerator 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] *see also*, [0081] (“a data transfer fabric can be programmable...connections are configurable”), [0082] (“data transfer fabric can provide data transfer paths between components”), [0083], [0084] (“a fabric...can be programmable, providing connections as needed”), FIG. 7; *supra*, Element [1e]. The fact that Dalal describes a *processor* is supported by Dalal’s description of “one or more module *processors*” that are “capable of accessing heterogeneous components of the module over multiplexed connections.” AMD1004, [0029], *see also*, cl. 7 (“processing circuits comprise at least one module *processor*”).

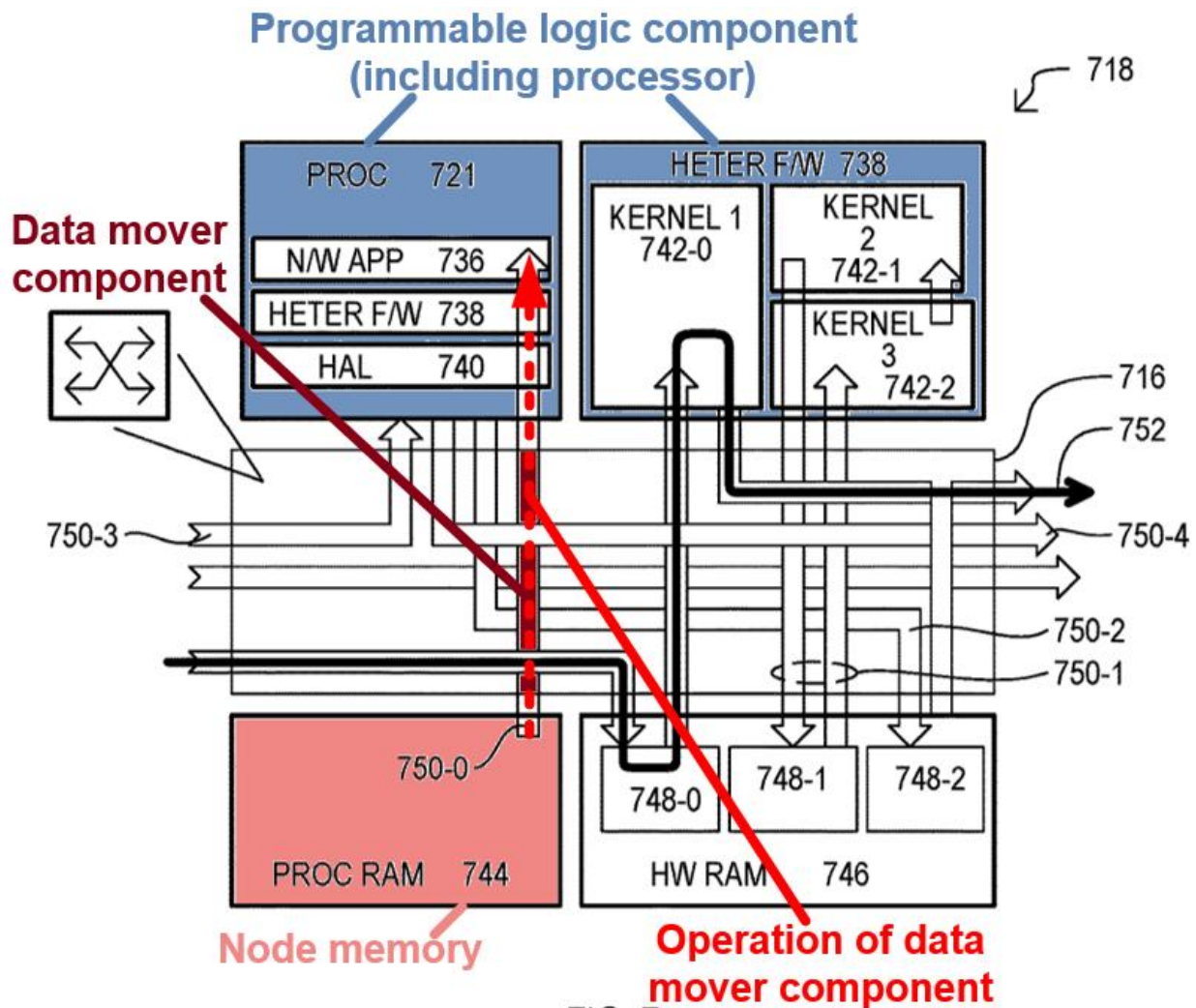


FIG. 7

AMD1004, FIG. 7 (annotated).

102. The Combination 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*, Element [1e]; *supra*, §VIII.C. 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.* The read control circuit and the write control circuit can include a **processor** that can **manage operation** of Ng's memory controllers that provide the claimed **data mover component**, because Ng expressly discloses that the neural network accelerator within Ng's **programmable** IC "can access the RAM...through the memory controllers." AMD1005, [0037], *see also*, [0036], [0038], [0040], [0041], FIGS. 2–3. Ng explains that the **programmable** IC of the hardware accelerator can include micro**processor**(s), meaning that Ng contemplates a **processor** to manage operation of the **data mover component**. AMD1005, [0080].

6. Claim 8

- (i) ***Element [8]: The compute node of claim 7 wherein the data mover component comprises a device controller.***

103. The Dalal-Ng Combination renders obvious [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]–[0082], *see also*, [0084]–[0086]; *supra*, Elements [1pre], [1e], [7]. The fact that the data transfer fabric "can provide data transfer paths between components" to **control** data movement in a **device** supports my opinion that Dalal's "data transfer fabric" (**data mover component**) in each hardware accelerator module represents a **device controller**. AMD1004, [0082], *see also*,

[0084] (“providing connections as needed”), [0085], [0086] (“enable processing pipelines to be constructed”).

104. Dalal’s data transfer fabric is controlled by an ARM *processor* of the *programmable logic component*, which is similar to the *device controller* described by the ’636 patent which is “operative to facilitate...data communications” either “independently” or “in cooperation with control signals.” AMD1004, [0085]; AMD1001, 16:45–61. Because Dalal explains that the data transfer fabric (*data mover component*) operates in cooperation with control signals from the ARM *processor* of the *programmable logic component*, this supports that the data transfer fabric includes a *device controller*. *Id.*

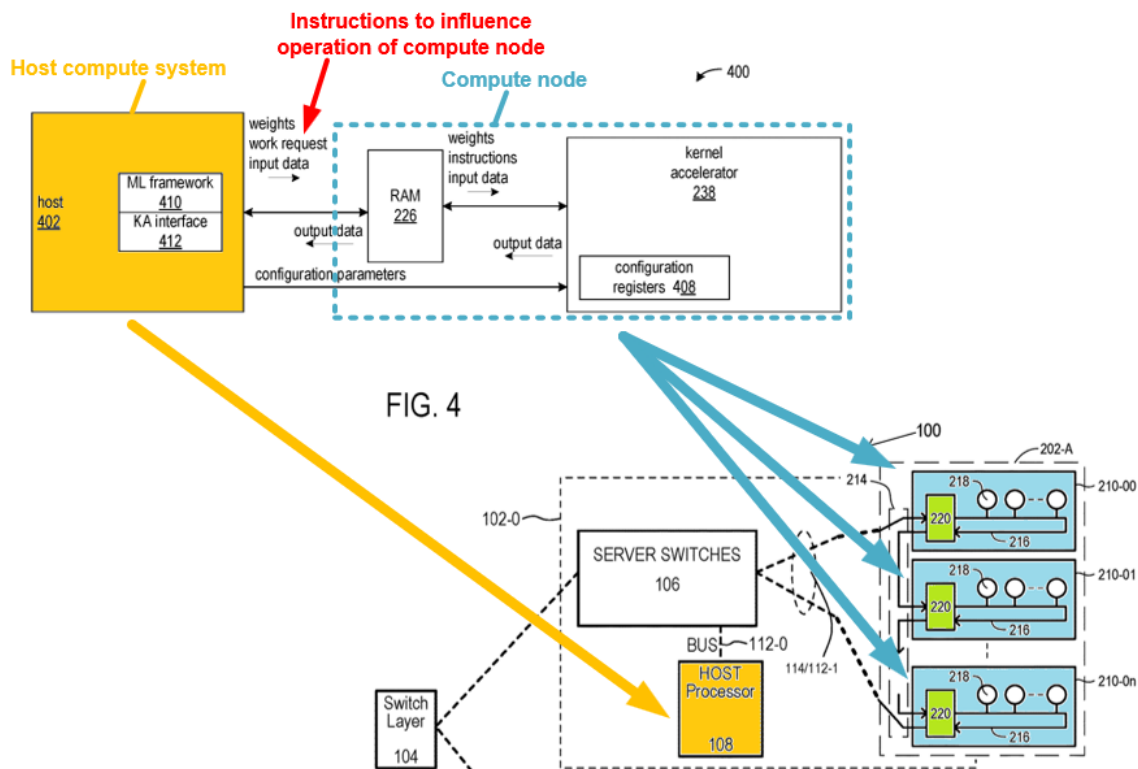
105. The Dalal-Ng Combination also incorporates Ng’s supplemental teaching of “memory *controllers*” of the hardware accelerator *compute node* which provide the claimed *device controller*. AMD1005, [0035]–[0037], [0043], FIG. 3; *supra*, §VIII.C. Ng explains that a neural network accelerator “can be implemented on a...*device*,” meaning that the memory *controllers* dictate operations on a *device*. AMD1005, [0043]. A POSITA would have been motivated to implement the servers of Dalal to include a *device controller* within each hardware accelerator module, because this would have beneficially provided a way to manage processor-memory communications in every module.

7. Claim 9

- (i) *Element [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.*

106. As I described above in my analysis of Elements [1pre]–[1a], 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]. The Dalal-Ng Combination 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, [0042] (“*host computer system*...communicatively coupled to neural network accelerator... initiates neural network operations on the neural network accelerators”), [0047] (“*host* waits until the neural network accelerator signals completion of performing the operations before instructing the neural network accelerator to commence performing the neural network operations of the next layer”), *see also*, [0007]–[0008], [0021]–[0024], [0029], [0041], [0043]–[0046], [0048]–[0050], FIGS. 4–5; AMD1004, [0032], [0070] (“[a] host processor...can be any suitable processor device”), [0063], [0069], [0073], [0077],

[0099]–[0103]; *supra*, §VIII.C. In the Dalal-Ng Combination, the *instructions* would have influenced *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.*



AMD1004, FIGS. 1, 2B Combination Figure (annotated); AMD1005, FIG. 4 (annotated).

107. In the Dalal-Ng Combination, a POSITA would have found it obvious to implement Dalal’s “host processors” with Ng’s “*host computer system*” so that the host executes *instructions* to *influence operation* of each hardware accelerator module *compute node* in the *execution pipeline*. AMD1004, [0032], [0063], [0069], [0073], [0077], [0099], [0101], [0103], [0123], [0135], [0170], [0172]; AMD1005,

[0007], [0008], [0021], [0024], [0029], [0041]–[0047], [0049], [0050], FIGS. 2, 4–5. For example, Ng explains that the output from the hardware accelerator for one later of the neural network can be used as the input for the next layer of the neural network, which is strikingly similar to Dalal’s pipelined processing where the output from one module “can be further processed by elements” in the next module. AMD1004, [0042]; AMD1005, [0047]. The hardware accelerator modules in Dalal’s *execution pipeline* would have benefited from tailored *instructions* from Ng’s *host compute system* that *influence operation* at that module. AMD1004, [0045].

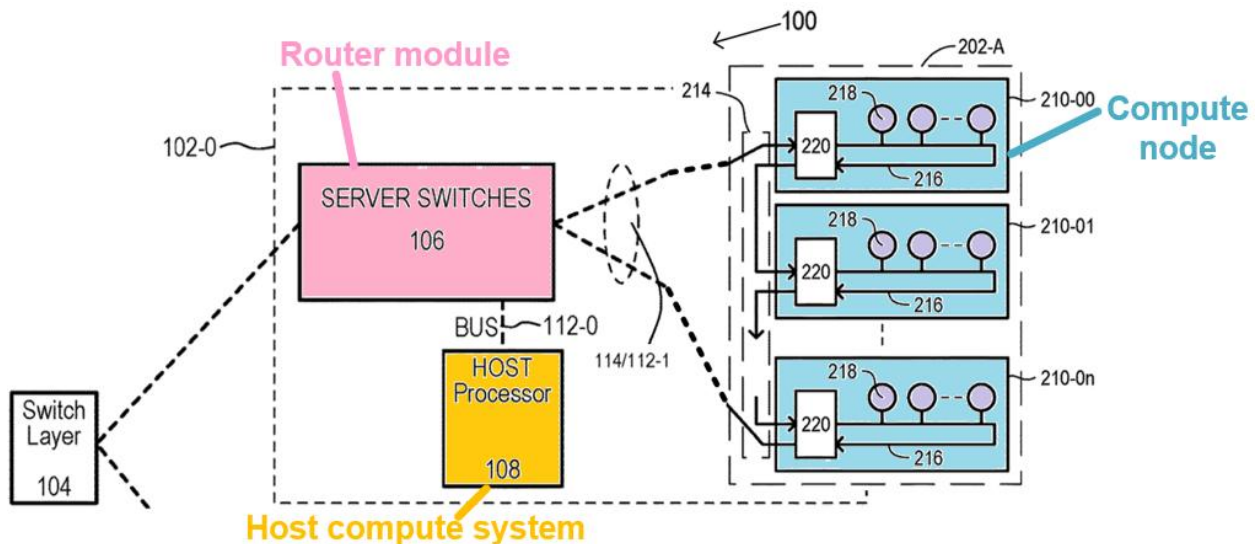
8. Claim 10

- (i) *Element [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.*

108. The Dalal-Ng Combination renders obvious [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. 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], *see also*, [0033]–[0036], [0123], [0170].



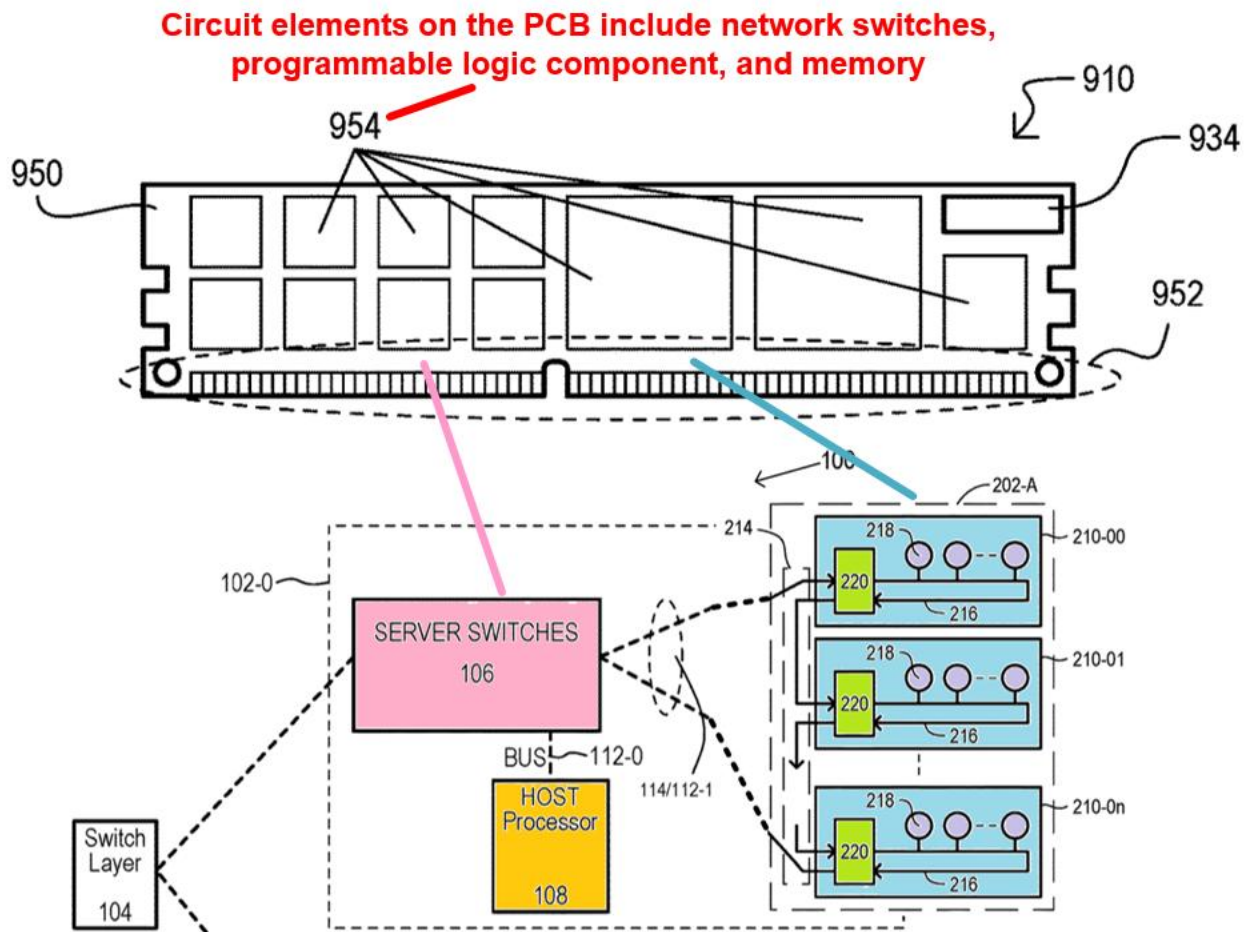
AMD1004, FIGS. 1, 2B Combination Figure (annotated).

109. Dalal discloses that each server includes server switches that provide the claimed *router module* and explains that the server switches “can be connected to...hwa modules” each representing a *compute node*. AMD1004, [0032], *see also*, [0035], [0036], [0123], [0170], FIG. 1; *supra*, Elements [1pre]–[1a]. Dalal’s server switches 106 (*router module*) are “connected to host processor 108” and “include one or more network switches that enable packet data to be switched” between the host processor (*host computer system*) and the hwa modules (*compute nodes*). AMD1004, [0032], *see also*, [0096], [0123], [0170], FIGS. 1, 12, 18.

110. Dalal explains that the network switch is *combined with* and *integrated into* the same printed circuit board that also includes memory and processor cores of

a hardware accelerator module (*compute node*). AMD1004, [0096] (“circuit components” include “memory; a programmable switch (e.g., network switch); and/or one or more processor cores”), FIG. 9. Thus, a POSITA would have understood that this printed circuit board is an *interface card* communicatively coupled to the *host compute system* into which a network switch of a *router module* and a *compute node* are *integrated* and *combined*. AMD1004, [0032], [0096], [0123], [0170], FIGS. 1, 9, 12, 18. It was well known before the Critical Date of the ’636 patent for a card to include a router and hardware accelerators. AMD1006, [0038] (describing a “card” comprising a “printed circuit board” that includes a “router” and “hardware acceleration units”).

111. Dalal explains that the printed circuit board’s *first compute node* can be connected to additional compute nodes because the interface card “can connect with other” hardware accelerator modules “via a bus...connected to physical interface 952.” AMD1004, [0096]–[0097], FIG. 9.



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

112. Dalal's printed circuit board is similar to the interface card of the '636 patent, which is also described as a "printed circuit board (PCB)" that integrates multiple different components. AMD1001, 11:46–61, 13:47–14:8. Integrating a switch and one or more hardware accelerator compute nodes into the same interface card was well known by the time of the '636 patent. AMD1024, 8:41–60 (describing an interconnect switch and multiple hardware accelerator modules integrated on the same printed circuit board. The fact that the '636 patent acknowledges that it is

“generally known” for a PCB to integrate a router module and compute node to “cooperate with structural components of [a] host compute system” supports my opinion that Claim [10] is obvious over the Dalal-Ng Combination. AMD1001, 11:46–61.

9. Claim 11

- (i) *Element [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.*

113. The Dalal-Ng Combination renders obvious [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]; *supra*, Elements [1b]–[1d.i]. Dalal explains that these programs are “written” on a “framework,” indicating that the programs include *computer-executable code*. AMD1004, [0076], *see also*, [0078], FIG. 7. 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] (“part of a machine learning library, such as TensorFlow”), FIG. 7; AMD1001, 48:36–40

(“algorithm for a neural network executing under TensorFlow™”). 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], *see also*, [0094], FIGS. 2B, 8.

114. Because the program “can be part of a machine learning library,” it is clear that running the program involves executing *operations* associated with a *computationally expensive algorithm* because computations (such as machine learning) are done according to algorithms. AMD1004, [0078], FIG. 7. Computation actions are well known to be done according to algorithms in computer code, and this is especially true for machine learning operations. Furthermore, Dalal describes a “programming model for processing large data sets with a parallel, distributed *algorithm*.” AMD1004, [0115].

115. The Combination also incorporates Ng’s supplemental teaching of a “hardware accelerator” (*compute node*) that is part of a “class of machine learning *algorithms* that uses multiple layers of nonlinear processing units for feature extraction and transformation.” AMD1005, [0002], *see also*, [0007] (“performing neural network *operations*”), [0023] (“neural network accelerator controls the

execution of the *operations* of the layers of the NN”), [0024], [0042] (“neural network *operations* on the neural network accelerators”). Such 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.

116. 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” and this represents using information in a computer-executable code to executes a respective operation, akin to Dalal’s processing circuits which read programs to perform computation actions. AMD1005, [0063], *see also*, [0045]–[0050]. These teachings reinforce that it would have been obvious to implement a *respective compute node* to accept results of a *preceding operation* and pass results of the respective operation to a *next compute node*. *Id.* According to Ng, machine learning *algorithms* can involve *operations* such as “matrix multiplication or convolution.” AMD1005, [0021], *see also*, [0022], [0041] (“performs matrix multiplication operations for implementing a neural network”), [0042] (“convolutions or matrix multiplications in the layers of the neural network”).

117. For every layer in a neural network, the **host compute system** sends a “work request” to the **compute node** (e.g., hardware accelerator 116) that guides an **operation** associated with the respective layer of the neural network. AMD1005, [0045], [0047], [0049], [0050], FIGS. 4–5. In addition, the **host compute system** can establish “configuration parameters” in “configuration registers” in the hardware accelerator’s **programmable logic component**. AMD1005, [0046], *see also*, [0027], [0035], [0047], [0050]. The **compute node** (e.g., hardware accelerator 116) uses this **information** to execute a respective **operation** associated with the neural network **algorithm**. AMD1005, [0027], [0042], [0045], [0046], [0047].

118. To execute a given layer of the neural network (**respective operation**), the **host compute system** can instruct the hardware accelerator to perform matrix multiplication using an “input” matrix representing the output of a previous neural network layer (**results of a respective preceding operation**) to generate an “output” matrix that is used as the input for a subsequent neural network layer (**respective additional results**). AMD1005, [0007], [0008], [0022]–[0025], [0044], [0045], [0047], [0066], cl. 1, cl. 11. In the Combination, one of Dalal’s hardware accelerator modules can execute a neural network layer using results of a **preceding operation** generated by a previous module and pass this output to a subsequent module (**next compute node**) in the **execution pipeline**. *Supra*, §VIII.C.

119. A POSITA would have been motivated to implement Dalal's *execution pipeline* comprising a plurality of *serially coupled* hardware accelerator modules with a *host compute system* that issues work requests to each of the hardware accelerator modules including *information* associated with a machine learning algorithm for executing a respective *operation* associated with the *computationally expensive algorithm*, because doing so would have enabled the *execution pipeline* to collectively perform a larger machine learning task. In this combination, the *information* associated with the *algorithm* for executing a respective *operation* would have been part of a *computer-executable code* (e.g., the program) that the processing circuits execute to perform the machine learning *algorithm*. AMD1004, [0076]–[0081], FIG. 7.

10. Claim 12

- (i) *Element [12]: The compute node of claim 11 wherein the computer-executable code associated with the computationally expensive algorithm is incorporated in a bit stream.*

120. The Dalal-Ng Combination renders obvious [12]. Dalal discloses that “programs can be 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. 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. 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]. It was well known by the time of the '636 patent that a time division multiplex (TDM) fabric uses a continuous *bit stream* to transport data. AMD1007, 1:23–30 (“TDM, short for Time Division Multiplexing, is a type of multiplexing that combines data Streams”), 4:61-67 (“[a] method and system for processing one or more TDMs...includes encapsulating...packets...to provide synchronous bit streams into fixed size cells”); AMD1040, 3:12–22 (describing “time division multiplexing” resulting in a bit stream having a “bit rate of f_2 ”) Dalal’s data processing fabric is no different. Furthermore, in the Dalal-Ng combination, the *information* includes “configuration parameters” which are uploaded to programmable logic in “configuration *bitstreams*” indicating that *the computationally expensive algorithm* in the Dalal-Ng Combination would have been *incorporated in a bit stream*. AMD1005, [0027], [0042]–[0043].

IX. GROUND 1A: DALAL, NG, AND RAGHAVA

A. Raghava (AMD1006)

121. Raghava describes a system including a “host processor coupled to a programmable acceleration coprocessor” that “may be used to accelerate a variety

of functions, which may include...convolutional neural networks.” AMD1006, [0017]–[0022], [0030] FIG. 1. The coprocessor “may be a...programmable logic device” including “FPGAs” and “may be configured to implement one or more hardware (HW) accelerators.” AMD1006, [0022], [0030]–[0033], [0038], [0039], FIGS. 1, 3–4. Raghava explains that the coprocessor includes one or more hardware accelerators that can include “external input-output (IO)” blocks. AMD1006, [0033], FIGS. 3–4. One example of an interface that can be used on the coprocessor is an “Open NAND Flash Interface (ONFI)” and Raghava explains that ONFI is “suitable...for communicating with memory” connected to the coprocessor. AMD1006, [0033], FIGS. 3–4.

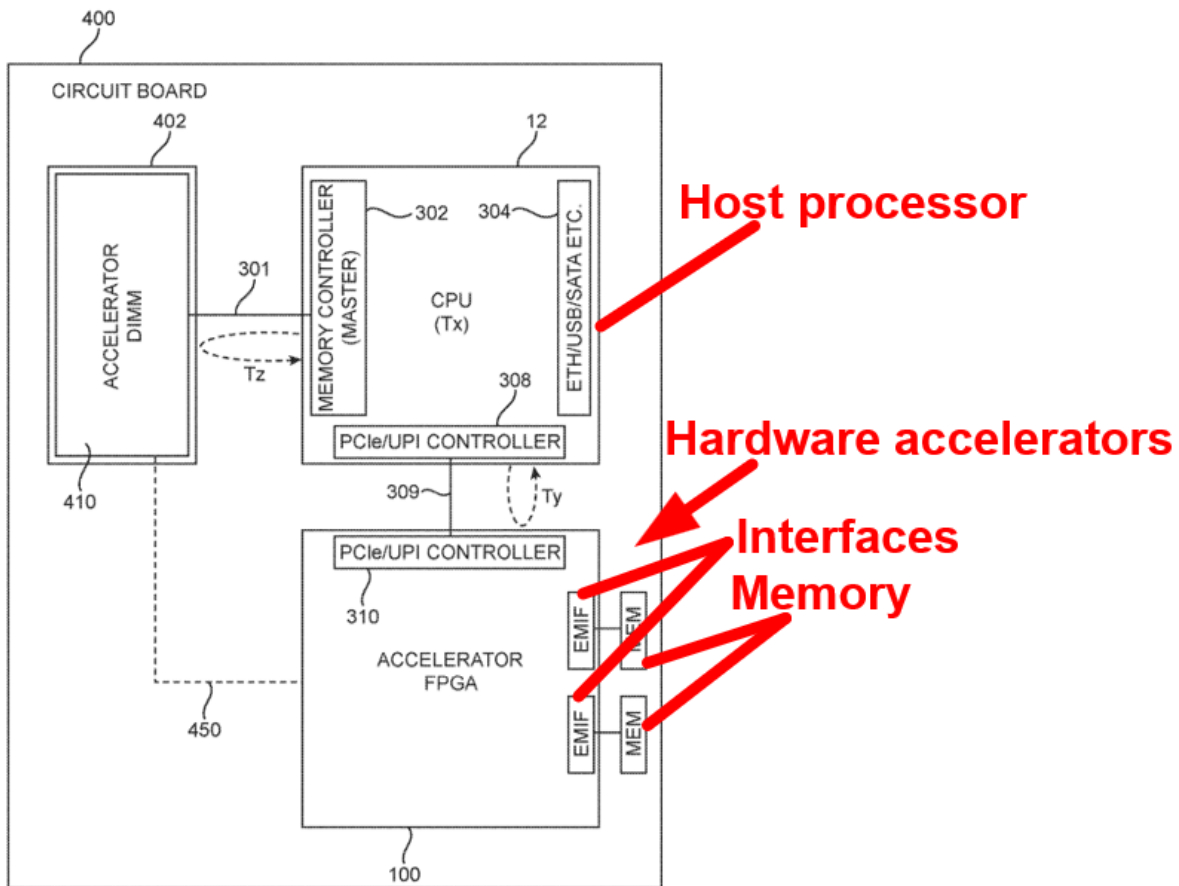


FIG. 4

AMD1006, FIG. 4 (annotated).

B. The Dalal-Ng-Raghava Combination

122. Dalal, Ng, and Raghava are all 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*, §§VIII.A–C. Dalal explains that “module memories” on the hardware accelerator modules “can be of any suitable type, including...flash memory.” AMD1004, [0067]. Raghava discloses an

embodiment where a hardware accelerator includes NAND flash memory and interfaces (i.e., “ONFI”) for connecting to the flash memory. AMD1006, [0033]. A POSITA would have been motivated to implement Dalal’s hardware accelerators with Raghava’s suggestion to use flash memory because doing so would have provided a convenient way to increase storage capacity and maintain storage in the case of power loss.

123. It is apparent that a POSITA would have found it obvious to combine the teachings of Dalal-Ng with the flash memory and ONFI storage interfaces of Raghava because the combination merely involves the application of a known technique to a known system to achieve predictable results. Dalal and Ng both disclose that hardware accelerators can use non-volatile memory, and a POSITA would have recognized that it would have been beneficial to implement these accelerators with Raghava’s flash memory. AMD1004, [0033], [0067], [0096], [0105]; AMD1005, [0032], [0034], [0080]. Because Dalal and Ng involve systems involving (i) a host that communicates with hardware accelerators to perform a data processing operation and (ii) flash, which is a solid state memory, combining the teachings of Dalal-Ng with Raghava’s express teaching of flash memory and ONFI would have been obvious and a POSITA would have reasonably expected success. AMD1004, [0032], [0078]; AMD1005, [0024].

124. The following reasons, taken separately or collectively, support the Dalal-Ng-Raghava Combination.

125. 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. This is confirmed by multiple references from before the time of the '636 patent. AMD1022, 1 (explaining that “non-volatile memory” such as “flash” has “notable features such as non-volatility” and “fast write speed,” and explaining that “flash memory....has benefits of a large capacity and relatively high performance”); AMD1038, 1 (“large capacity flash memory”). Dalal contemplates that hardware accelerator memories “can be of any suitable type” such as “RAM” or “flash memory.” AMD1004, [0067]. It would have beneficially increased memory capacity and removed power dependency on continued storage to implement Dalal’s hardware accelerators with NAND flash memory as described by Raghava.

126. 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 the Dalal-Ng Combination 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]. Dalal and Ng describe

hwa modules having multiple different kinds of memory, and a POSITA would have understood that Raghava's teachings are helpful for managing this dynamic. AMD1004, [0067]; AMD1005, [0032].

127. Third, it would have been obvious for a POSITA to implement the hardware accelerators of Dalal-Ng with NAND flash memory and ONFI as described by Raghava because doing so represents one of a finite number of predictable solutions for configuring memory of a hardware accelerator module. A POSITA would have appreciated that non-volatile memory is a viable option for high capacity data storage, and it would have been beneficial to implement Dalal's data store with flash memory. Indeed, Dalal and Ng describe hardware accelerator modules that include flash memory. AMD1004, [0067] (“[m]odule memories...can be of any suitable type, including...flash memory”); AMD1005, [0032] (“[t]he hardware accelerator...includes a non-volatile memory” that “can include any type of non-volatile memory, such as flash memory”).

C. Claim Element Analysis

1. Claim 4

(i) *Element [4]: The compute node of claim 1 wherein the data comprise an Electronically Erasable Programmable Read Only Memory (EEPROM) device.*

128. The Dalal-Ng-Raghava Combination renders obvious [4]. The '636 patent explains that “all Flash is EEPROM (as a technical matter),” meaning that

any flash memory device includes an **EEPROM device**. AMD1001, 10:65–11:21. Dalal’s “second memory” (HW RAM) corresponds to the **data store**. AMD1004, [0075], [0080]–[0082], [0094], cls.1, 11; *supra*, Element [1b]; *supra*, §IX.B. For the reasons noted above, in the Dalal-Ng-Raghava combination, the **data store** of the Dalal-Ng Combination would have been implemented with Raghava’s accelerator FPGA, which includes an EEPROM 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.*

129. Raghava discloses that an accelerator FPGA (**programmable logic component**) can include external input-output (IO) blocks that provide the claimed **storage interface component** and memory that provides the claimed **data store**. AMD1006, [0033], [0034], [0014], [0046], [0074], cl. 10, FIGS. 3–5. Raghava discloses that the IO blocks support the “Open NAND Flash Interface (ONFI)” protocol and explains that these IO blocks are “suitable interfaces for communicating with memory.” AMD1006, [0033]. This means that Raghava’s memory includes NAND flash memory in embodiments where the IO blocks are compatible with ONFI. Because Raghava discloses flash memory, Raghava provides the claimed **EEPROM device**. AMD1004, [0033], [0034], FIGS. 3–4.

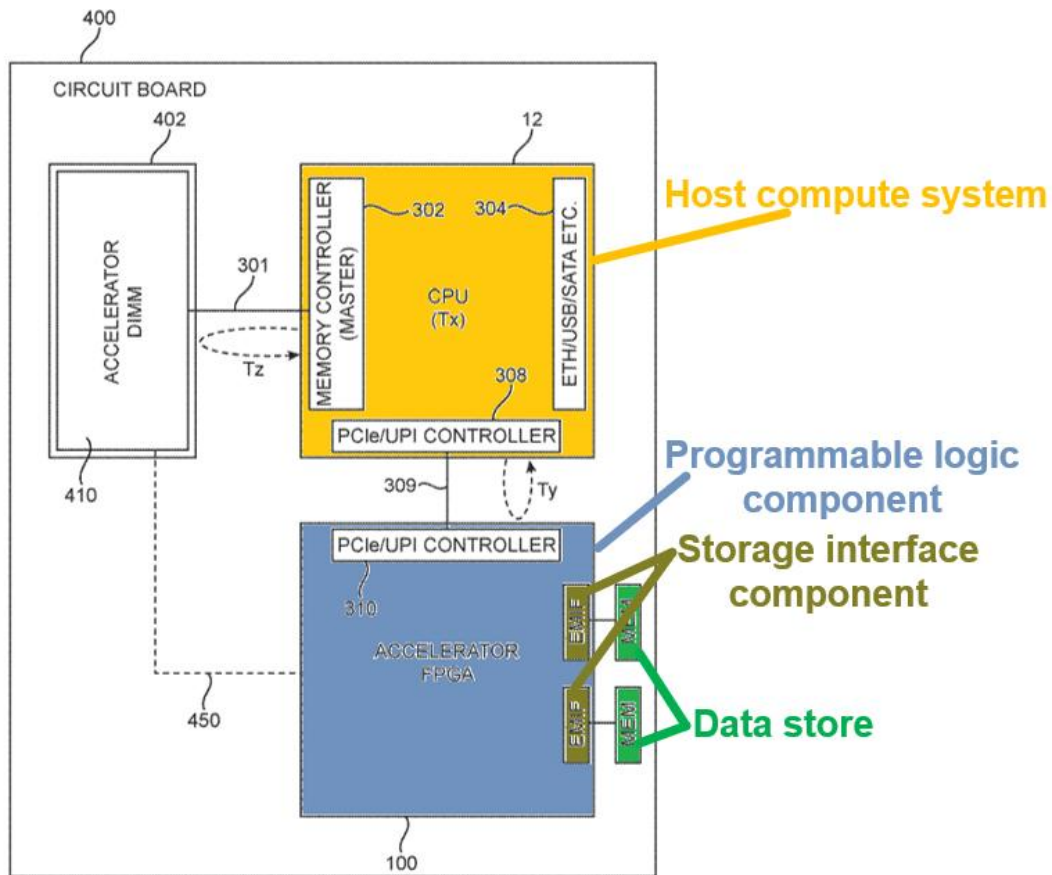


FIG. 4

AMD1006, FIG. 4 (annotated).

2. Claim 5

- (i) *Element [5]: The compute node of claim 4 wherein the storage interface employs an Open NAND Flash Interface (ONFI) protocol.*

130. The Dalal-Ng-Raghava Combination renders obvious [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], cls.1, 18–20; AMD1005, [0035], [0036],

[0038]; *supra*, [1f]–[1g]. For the reasons noted above, in the Dalal-Ng-Raghava combination, the ***storage interface component*** of the Dalal-Ng Combination would have been implemented with Raghava’s accelerator FPGA, which include input-output (IO) that employ ***ONFI***. AMD1006, [0033], *see also* [0014], [0046], [0074], cl.10, FIGS. 3, 5; *supra*, [4]; *supra*, §IX.B. A POSITA would have understood that ONFI is widely adopted in the computer hardware industry as a standard for interfacing with Flash memory.

131. Raghava discloses that the external input-output (IO) blocks” that provide the claimed ***storage interface component*** in each hardware accelerator module (***compute node***) can employ the “***Open NAND Flash Interface (ONFI)***” protocol, and a POSITA therefore would have found it obvious to implement each ***compute node*** in the Dalal-Ng combination with a ***storage interface component*** that employs ***ONFI*** as suggested by Raghava. AMD1006, [0033], *see also*, [0014], [0046], [0074], cl. 10, FIGS. 3, 5; *supra*, [4]. A POSITA would have found it obvious to make this combination, because doing so would have provided nonvolatile storage at increased capacity. *Supra*, §IX.B.

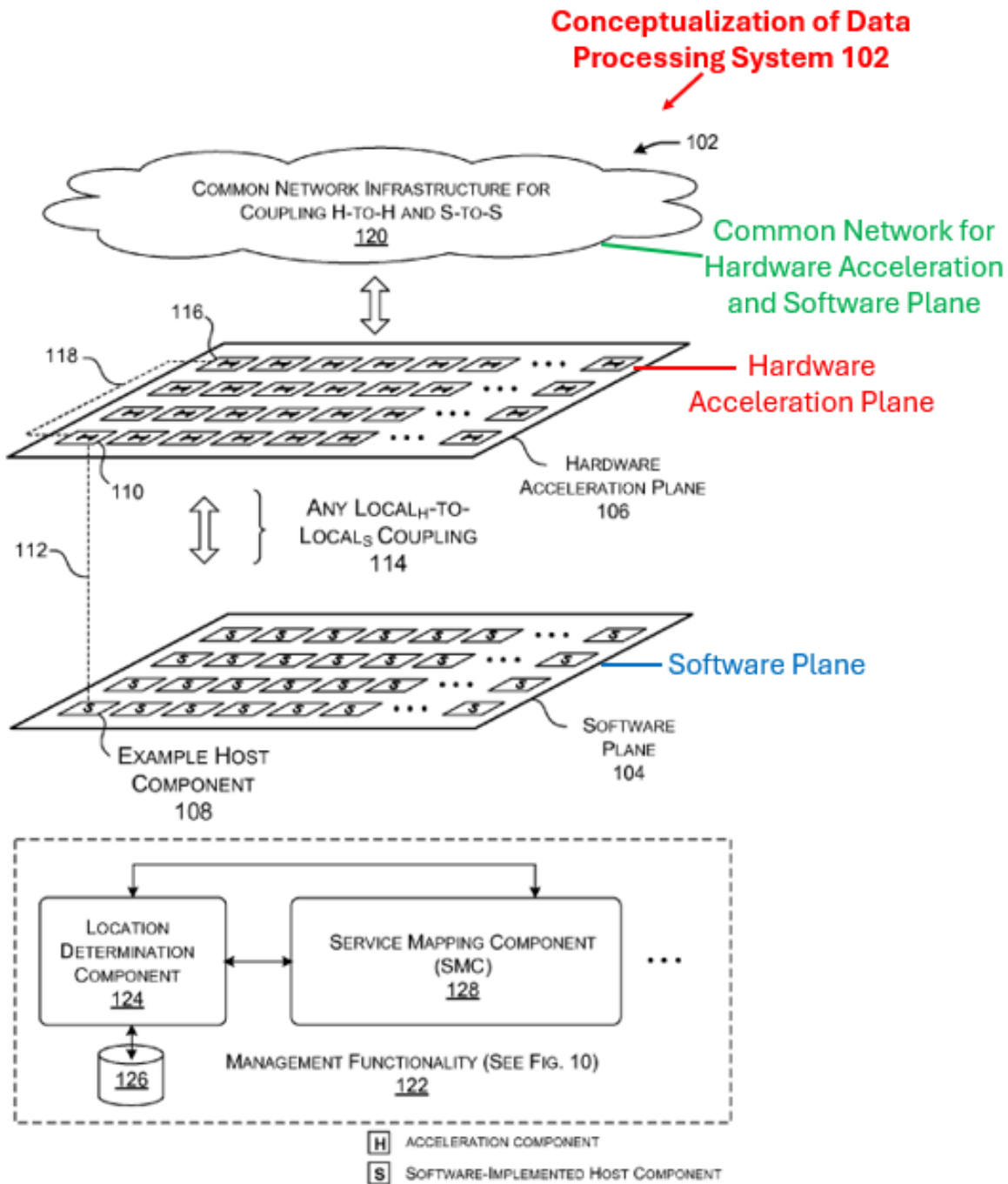
X. GROUND 2A: HEIL AND JUN

A. Heil (AMD1010)

132. Heil is directed to data processing systems for implementing multi-component services. AMD1010, [0040], *see also* AMD1010, [0003], [0040],

[0053], [0218]–[0248], Abstract. Heil’s data processing systems include multiple server unit components, each having a local host component paired with a local acceleration component, like the ’636 patent’s multiple compute nodes each having a “processing system 430”/“memory 450” paired with a “programmable logic 410.” AMD1010, [0053]; AMD1001, FIG. 4; *supra* §V.A. Heil explains how these “server unit components” are configured by the host component and communicatively connected to process a request as part of a multi-component service “without involvement from any host component.” AMD1010, [0003], [0232], [0248]; *see also* [0040], [0053], [0218]–[0248]; *infra* Elements [1d.i]–[1e].

133. Heil’s data processing systems (*e.g.*, 102 in Figure 1 and 802 in Figure 8 both 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 (shown below), the two logical networks share a common network. AMD1010, [0048]–[0058], [0089], FIGS. 1, 8.

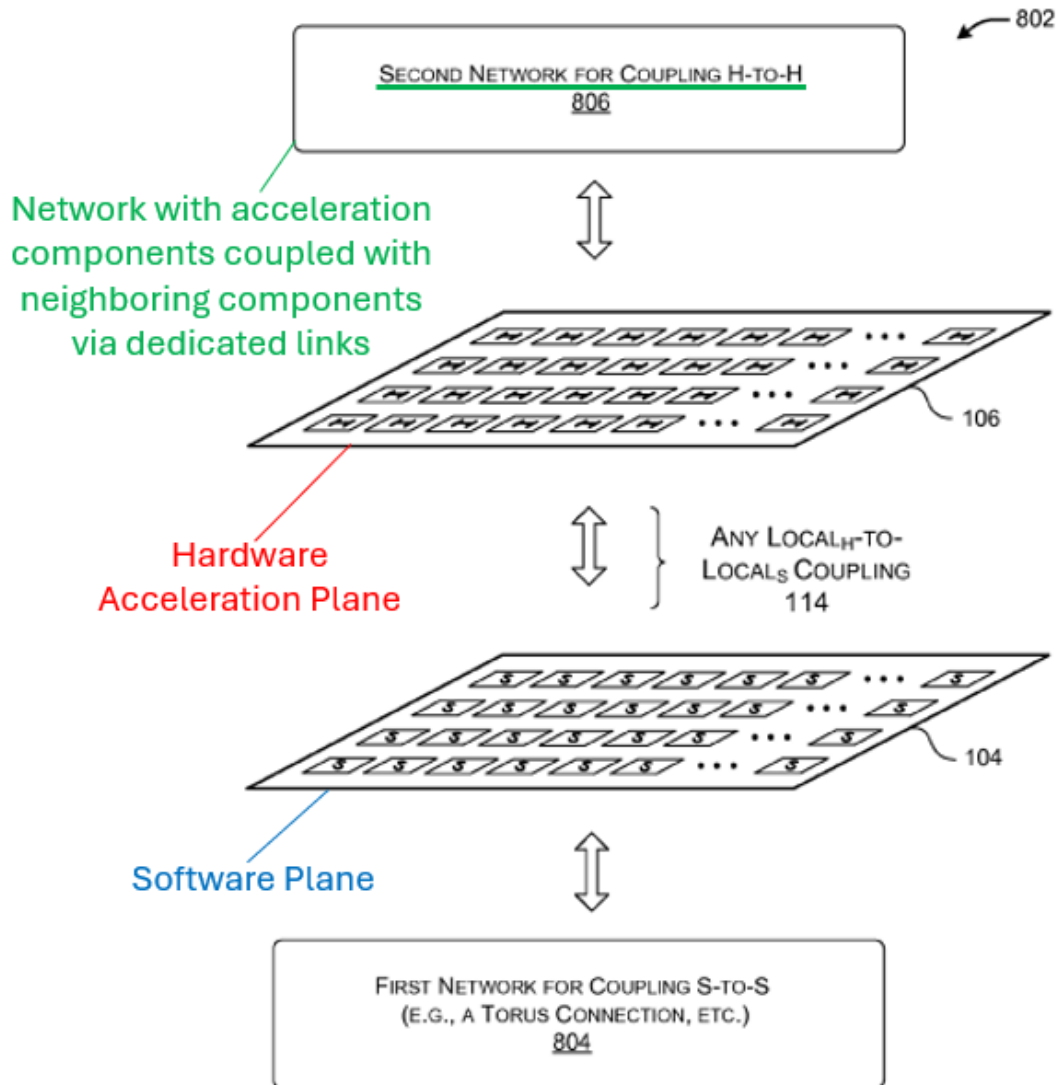


AMD1010, FIG. 1 (annotated).

134. Heil's Figure 8 (shown 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]; *see also* [0058], [0096]. For example, Heil explains that dedicated links between “neighboring acceleration components” are established “for connecting [them] together [in a] network topology.” AMD1010, [0089]. I note that in Figure 8, an example of the second network that is, a torus network. AMD1010, [0089].

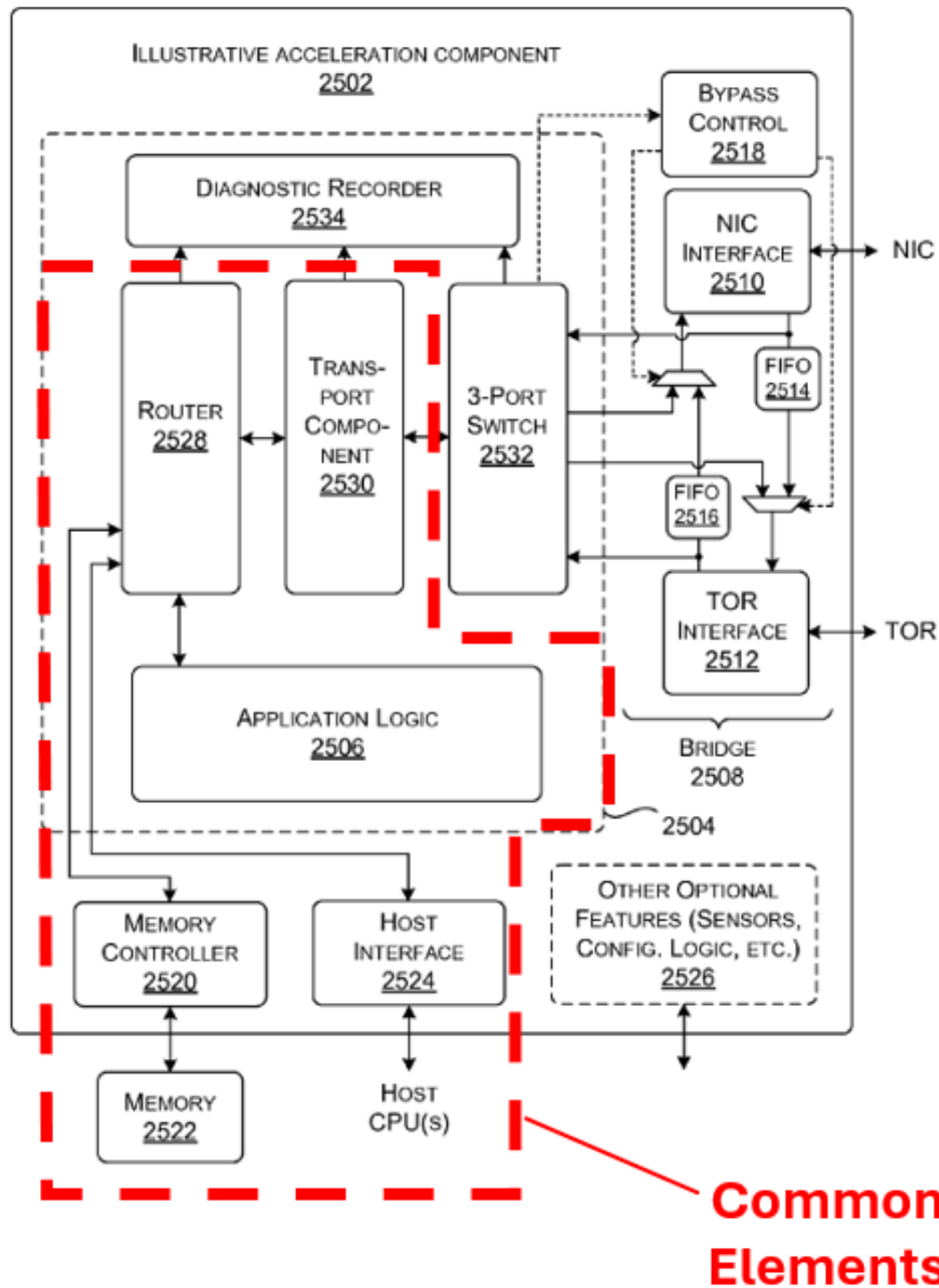
**Conceptualization of Data
Processing System 802**



AMD1010, FIG. 8 (annotated).

135. In both Figures 1 and 8, Heil's acceleration components are represented by the boxes labeled with "H" in the hardware acceleration plane 106. *See* AMD1010, FIGS. 1, 8 (both annotated above). Figure 25 (shown below) presents an illustrative example of these acceleration components as illustrative acceleration

component 2502. The acceleration component includes, among additional elements, 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.



AMD1010, FIG. 25 (annotated).

136. 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 with dedicated links (e.g., at TOR interface 2512) and would have still utilized the above identified common elements to implement core capabilities. For example, a POSITA implementing the embodiment shown in Heil’s Figure 8 would have looked to the disclosure of Figure 25 to implement the application logic 2506 as the reconfigurable component for performing an assigned operation, the memory controller 2520 as the component used for “interaction between the acceleration component 2502 and local memory 2522,” the host interface as the component used to enable “the acceleration component to interact with a local host component” (including host CPU(s)), the router 2528 “for routing messages between various internal components,” and the “transport component 2530” to interface with remote acceleration components. AMD1010, [0160]–[0168]; *see also* AMD1010, [0014], [0089], [0139], [0157]–[0158], [0171]–[0173]–[0186], [0226], FIGS. 26–29.

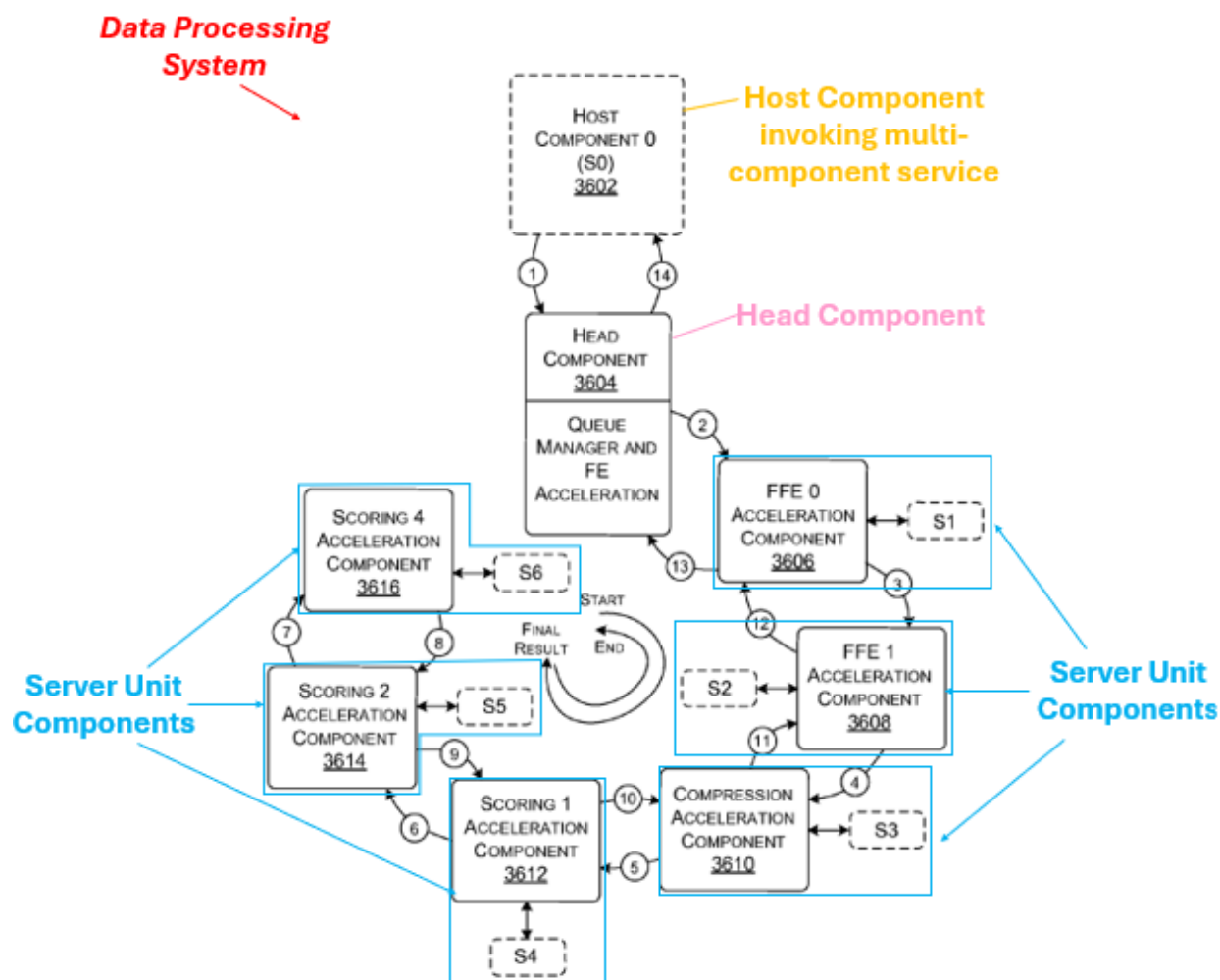
137. Moreover and as an additional basis, a POSITA would have found obvious to implement these common elements because utilizing the same elements to fulfill the same core capabilities would have allowed system 802 to operate. A POSITA would have understood that because both embodiments in Figures 1 and 8

utilize the same FPGA-based acceleration component, it would have been a matter of routine engineering for a POSITA to implement these internal FPGA components with common functionality within the embodiment in FIG. 8.

138. A POSITA would have also been motivated to utilize common components across systems 102 (From FIG. 1) and 802 (from FIG. 2) to reduce the risk of altering the functionality of these common components in a manner that causes problems and to avoid needing to recreate these common components.” AMD1010, [0159]–[0162]. Specifically, Heil explains that the common elements are part of an “inner shell” architecture that “that remain the same” across different application scenarios. AMD1010, [0161]–[0162]. For example, this architecture “reduces the risk that a developer may alter core inner or outer shell functions in a manner that causes problems within the data processing system 102 as a whole.” AMD1010, [0161]–[0162]. Heil also explains that the inner and outer shell architecture negates the need to “recreate these common components.” AMD1010, [0159]–[0162]. To achieve the same benefits, a POSITA would have utilized these components across different embodiments. Reusing such component designs across different system designs was a well-known design technique to minimize development costs and ensure quality.

139. Turning now to Heil’s disclosure of services, Heil explains that the data processing systems above execute multi-component services. AMD1010, [0218]–

[0223], [0229], FIGS. 36, 40. In one example, referenced in Figure 36 (also shown below) Heil describes a multi-component service for assigning “a ranking score to a pairing of a query and a document.” AMD1010, [0218]–[0223]. In this example, a “host component 3602” invokes a service by sending a request to a “head component 3604.” AMD1010, [0219]–[0220]. Heil explains that a network of server unit components executes the invoked service using a processing pipeline with communicatively coupled acceleration components (e.g., 3606, 3608, 3610, 3612, 3614, and 3616 in FIG. 36) to perform “complex feature computations” and “generate a final ranking score.” AMD1010, [0221]–[0223], *see also* AMD1010, [0068] (“processing pipeline”). “The labels S1, S2, . . . , S6 denote local host components that are associated with the respective acceleration components shown in FIG. 36.” AMD1010, [0223]

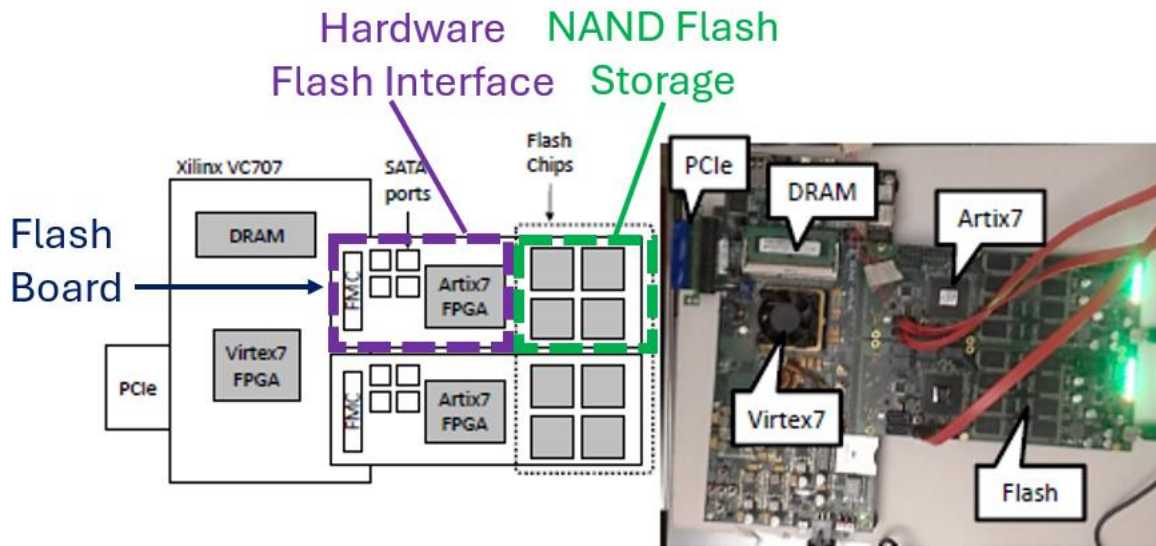


AMD1010, FIG. 36 (annotated).

B. Jun (AMD1011)

140. Jun describes a data processing system that includes a collection of FPGA-based compute nodes. AMD1011, pp.7–9, 13, *see also id.* 1–5. Jun describes an example 20-node system that improves upon a previous 4-node prototype. AMD1011, pp. 2, 4. Jun explains that the nodes can be implemented in “[a]ny network topology.” AMD1011, 5, FIG. 5.

141. Jun's nodes integrate flash storage with an FPGA processor device in order to execute analytics operations on large data sets (*e.g.*, Big Data analytics). AMD1011, pp.1–5, 12–13; *see* AMD1001, 9:11–16; 10:65–11:21, 27:39–50; 48:26–31, FIG. 1. Jun explains that it is advantageous to use NAND Flash Storage in place of DRAM memory to effectively handle tasks on large datasets. AMD1011, pp.2, 4, 12. Jun's NAND Flash Storage is deployed in conjunction with a multi-channel hardware flash interface and connectors to the FPGA processor on a “flash board.” AMD1011, pp.4, 8. The hardware interface and connectors on the flash board facilitate communication between the FPGA processor and the NAND Flash Storage. AMD1011, pp.4, 8. I note that in annotated Figure 10 below, the programmable acceleration component corresponds to the “Virtex FPGA” (*e.g.*, processor/accelerator) that is part of the “Xilinx VC707” device. AMD1010, pp.7–8, FIG. 10.



AMD1011, FIG. 10 (annotated).

C. The Heil-Jun Combination

142. Heil and Jun like the claims of the '636 patent belong to the field of endeavor of data processing systems employing reconfigurable hardware. AMD1010, [0003], [0232], [0248]; AMD1011, p.5; AMD1001, 2:31–48, 3:8–28, 38:32–43; FIG. 17; FIG. 17. Additionally, each reference are also reasonably pertinent to a technical problem shared with the '636 patent of accelerating processing by offloading tasks to reconfigurable hardware. AMD1010, [0003], [0051], [0224], [0232], [0248]; AMD1011, pp.1–6 (“storage devices form a separate network among themselves via high-performance serial links...data traffic between the storage devices, the integrated network ports removes the overhead of going to

the host software to access a separate network interface.”); AMD1001, 2:31–48 (“independent of...a host compute system”), 3:5–25, 37:44–55; FIG. 17.

143. I note that Jun references a paper AMD1014 describing a Microsoft project “called Catapult.” AMD1011, pp.4, 14. This paper refers to the same technology disclosed in Heil. AMD1014, FIGS. 5 *see* AMD1010, FIGS. 36-37; *see also generally* AMD1014. Both AMD1014 and Heil illustrate a similar figure for implementing an algorithm (“multi-component service”) including the same or similar steps on in a ring flow structure. AMD1010, [0218]–[0223], FIG. 36; AMD1014, 6-9, FIG. 5. There is also substantial overlap between the authors of AMD1014 and the inventors of Heil. AMD1010, Cover; AMD1014, 1. In fact all of the inventors of Heil are listed as authors on AMD1014. AMD1010, Cover; AMD1014, 1. Microsoft is both the Applicant for Heil and the listed entity for AMD1014 on page 1. AMD1010, Cover; AMD1014, 1. Jun states that it “has similar goals in terms of reconfigurable hardware acceleration” to this Microsoft project and this also applies to Heil’s FPGA/accelerator formed network. AMD1011, p.4.

144. As I discussed above, Heil’s data processing system utilizes a plurality of “server unit components” each comprising a host component and an acceleration component (FPGA device). *Supra*, §X.A (overview of Heil). In the context of Heil’s acceleration component and as I will explain in more detail below, a POSITA

would have been motivated to implement Jun's "flash board"—including its NAND Flash Storage, and hardware flash interface, and connectors—in place of the DRAM memory elements that Heil describes.

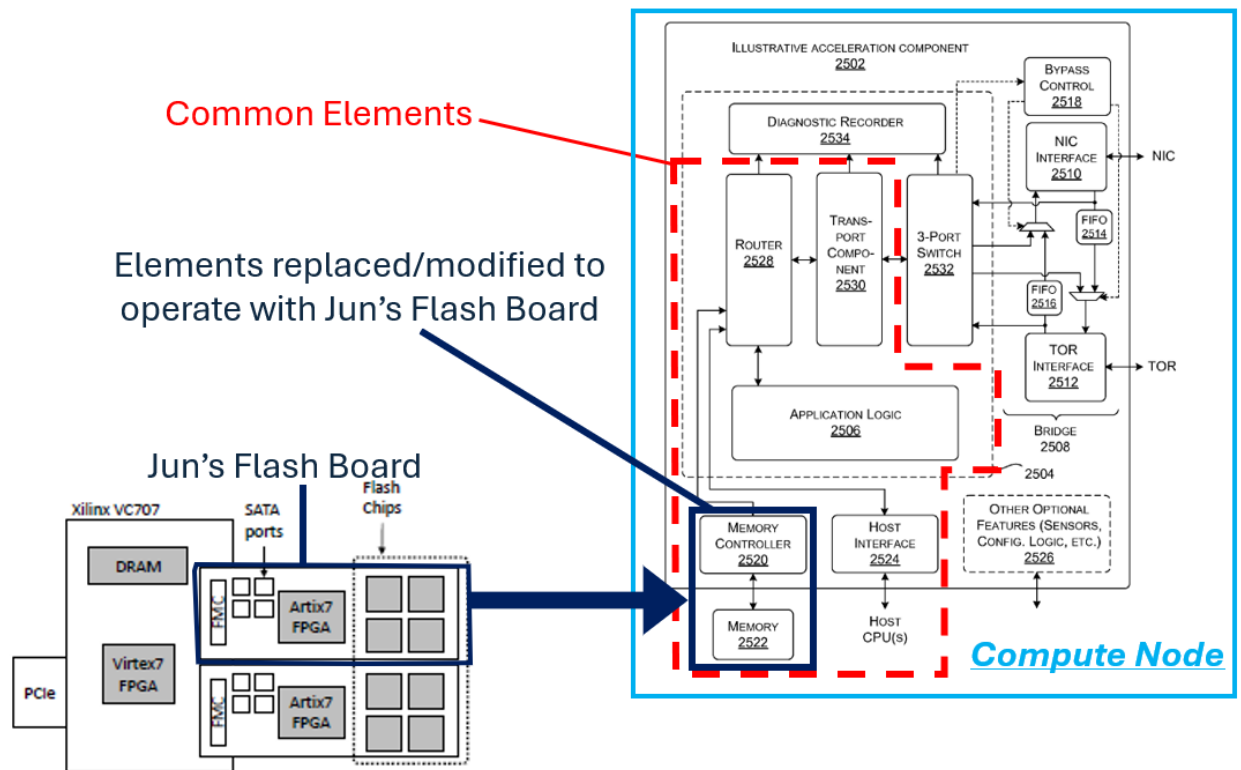


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and AMD1010, FIG. 25).

145. Based on my review, the following reasons, taken separately or collectively, support the Heil-Jun Combination.

146. First, a POSITA would have recognized that implementing Jun's "flash board" elements allows Heil's data processing system to accelerate multi-component services on larger datasets utilizing the same number of server unit components. Jun

demonstrates that hosting large datasets on DRAM-based acceleration components requires a large cluster of compute nodes. *See e.g.*, AMD1011, p.4 (using “flash devices to accelerate lower cost systems that do not have enough collective DRAM to host the entire dataset”), *see also* AMD1011., pp.2–3, 12. Jun explains that replacing DRAM with NAND Flash Storage enables high-capacity data hosting on fewer compute nodes, significantly reducing the required number of compute nodes or expanding the capacity of a given collection of compute nodes. AMD1011, pp.3–4. A POSITA would have been motivated to implement the Heil-Jun combination in order to accelerate services on datasets that exceed the DRAM capacity for the collection of acceleration components in Heil’s data processing system.

147. Second, a POSITA would have understood that implementing Jun’s “flash board” elements within Heil’s acceleration component allows for a reduction in architectural complexity by allowing implementations of services to require fewer server unit components. For example, it would have been clear to the POSITA that replacing Heil’s DRAM with Jun’s NAND 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. *See* AMD1011, Abstract (explaining that a dataset requiring 100 DRAM-based servers “could be stored easily in the flash memory of a rack-sized cluster.”). 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 (e.g., because there would be less elements and components to maintain and/or replace), and makes the system easier to implement and use. *See* AMD1011, 2-3.

148. Third, a POSITA would have understood and obvious that implementing Jun's "flash board" elements in Heil's acceleration component would have facilitated a reduction in capital cost because NAND Flash is a more cost-effective memory medium than DRAM. As Jun explains, NAND Flash Storage is "an order of magnitude cheaper price compared to DRAM." AMD1011, p.3, Abstract, 4, 12; *see also* AMD1037, 5 ("A RAMCloud could be constructed with flash memory as the primary storage technology instead of DRAM ('FlashCloud'), and this would be cheaper and consume less energy than a DRAM-based approach.").

149. Fourth, a POSITA would have understood and found obvious that replacing Heil's DRAM-based storage with Jun's "flash board" elements would have facilitated improved power efficiency and further resulting in lower operating costs, among other benefits. As Jun explains, NAND Flash Storage "consumes an order of magnitude less power" than the DRAM-based alternative. AMD1011, pp.3 (explaining that DRAM's "high energy consumption...limits its widespread use except for extremely performance and latency-sensitive workloads"), 11 ("less power hungry than a cloud based system with enough DRAM to accommodate 10TB

to 20TB of data”); *see also* AMD1037, 5 (“A RAMCloud could be constructed with flash memory as the primary storage technology instead of DRAM (‘FlashCloud’), and this would be cheaper and consume less energy than a DRAM-based approach.”).

150. 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. For example, Jun recognized a problem with existing DRAM-based acceleration components and described a straightforward technique to address it using NAND Flash Storage. In this context, combining the teachings of Heil and Jun would have been obvious, a simple substitution of known elements that a POSITA would have reasonably expected to succeed.

151. I also note that a POSITA would have reasonably expected a successful outcome from the above-discussed combination of teachings from Heil and Jun. . For example, a POSITA would have reasonably expected to succeed because both systems utilize similarly networked compute nodes that, in both references, utilize FPGA-based acceleration to perform similar processing tasks. Additionally, this conclusion is supported by my prior discussion at §§X.A (Heil), X.B (Jun) and my prior discussion on motivation-to-combine at §X.C. As I have explained, substituting DRAM with Flash storage—was well known by the critical date. *See*

AMD1011, AMD1022, AMD1037. A POSITA bringing this well-known substitution with Heil, in the manner proposed above at §X.C, would have been reasonably certain that the result of the combination would have arrived at the purported invention of '636 patent (as outlined below). The POSITA would have also understood that these integrated features are fully compatible and would have worked in combination in the same (or substantially the same way) that they work individually.

D. Claim Element Analysis

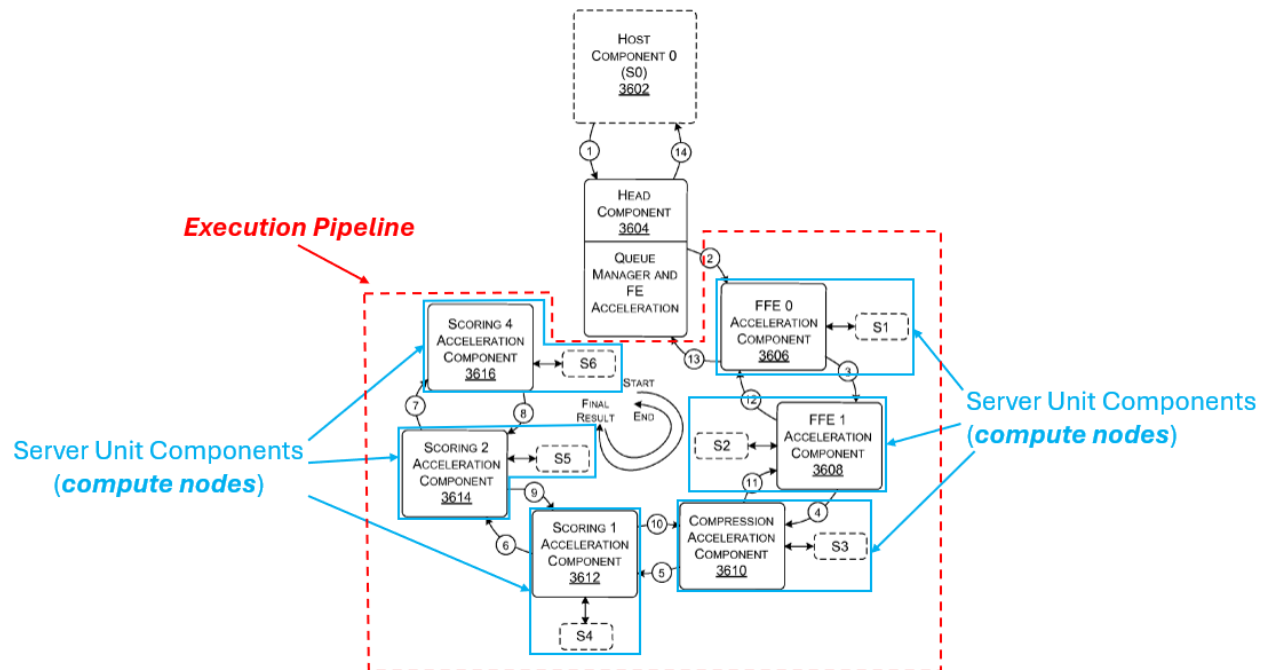
1. Claim 1

(i) *Element [1pre]: A compute node for use in a data processing system, the compute node comprising:*

152. The Heil-Jun Combination renders obvious Element [1pre] and Claim 1 as a whole. Heil's describes multiple server unit components, (*compute nodes for use in a data processing system* for performing computations/data processing, and that each having a local host component paired with a local acceleration component. *Supra*, §X.A; AMD1010, [0040], [0053], [0089], [0218]–[0223], FIGS. 8, 36, 40; *see also* [0003], [0232], [0248].

153. As just one example in Figure 36, the server unit components (acceleration component 3606 with S1, acceleration component 3608 with S2, acceleration component 3610 with s3, acceleration component 3612 with s4,

acceleration component 3614 with s5, and acceleration component 3616 with S6) each represent *a compute node*. AMD1010, [0218]–[0223] (also indicating that the S unites are local host components), 0053] (“The host component in the server unit component is referred to as the ‘local’ host component”); *see also* AMD1010, [0074], [0084], [0089], [0229], FIGS. 8, 40.



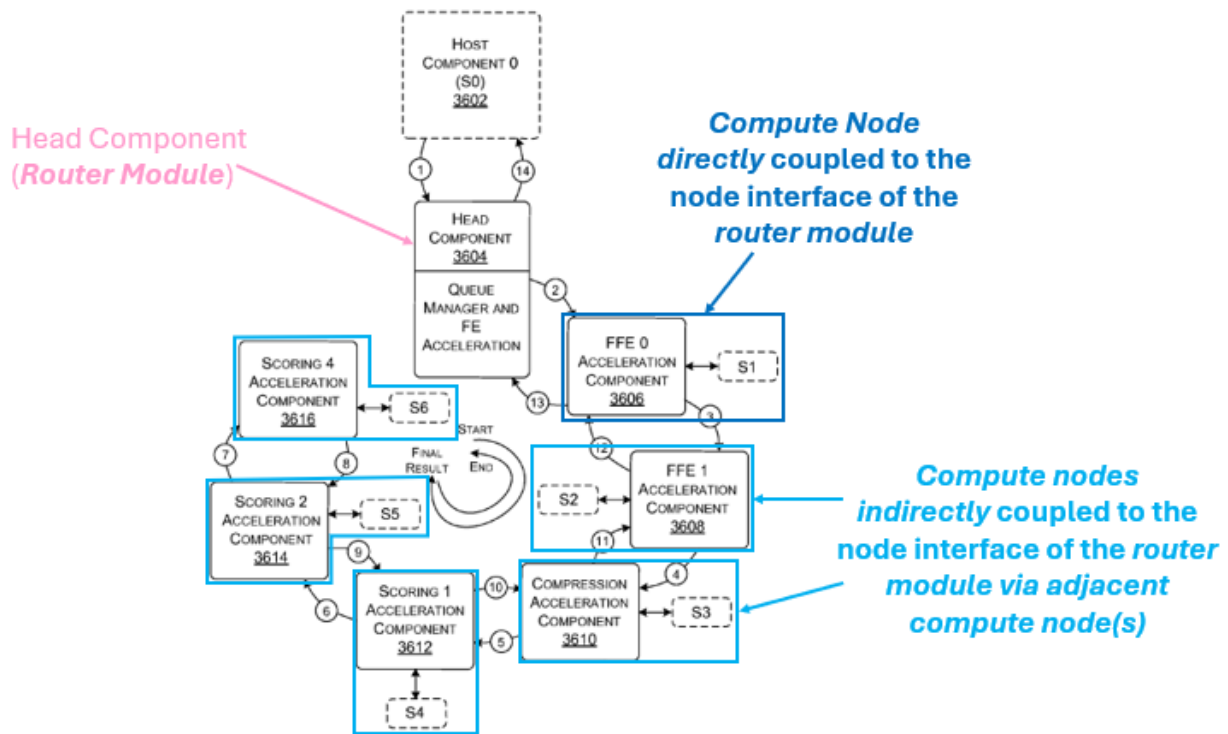
AMD1010, FIG. 36 (annotated).

- (ii) *Element [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;*

154. The Heil-Jun Combination renders obvious Element [1a] and Claim 1 as a whole. AMD1010, [0040], [0065]–[0069], [0073]–[0089], [0099]–[0100];

[0114]; [0160]–[0170]; [0192]; [0198]–[0205]; [0218]–[0225]; [0229]–[0236], [0248]–[0251], FIGS. 1–5, 8, 10, 16, 30, 32–33, 36, 38–40.

155. As I discussed for Element [1pre], Heil’s server unit components (*compute nodes*) each represent *a compute node*. In the example shown in Figure 36, Heil explains how a host component 3602 (*host compute system*) is *communicatively coupled* with the rest of the *data processing system* to “invoke the service by sending a request to a first acceleration component of the multi-component service.” AMD1010, [0219]–[0220], FIG. 36; *see* AMD1001, 9:17–46 (“Host compute system 199 may generally be embodied in or comprise a computer server” and “may be employed to initiate, instantiate, or otherwise to request data processing operations to be executed by system 100.”). Heil’s first server unit component (*compute node*), for example, acceleration component 3606 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], [0218]–[0223], [0229], FIGS. 8, 36, 40; *see also* [0055] (“the local acceleration component 110 communicates with the remote acceleration component 116 via a link 118.”), [0068] (“processing *pipeline*”), [0074].



AMD1010, FIG. 36 (annotated).

156. Heil discloses an example implementation of an acceleration component in Figure 25 (reproduced below). AMD1010, [0022], [0160]. As discussed at §X.A, a POSITA would have understood or found obvious that the acceleration component included Heil's, 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 below in Figure 25. AMD1010, [0167]–[0168] (“The transport component 2530 formulates packets for transmission to remote entities (such as remote acceleration components), and receives packets from the remote acceleration components (such as remote

acceleration components).”), [0181], FIG. 25; *see supra*, §XA. 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). AMD1010, [0167]–[0168] (“The transport component 2530 formulates packets for transmission to remote entities (such as remote acceleration components), and receives packets from the remote acceleration components (such as remote acceleration components).”), [0181], FIG. 25; *see supra*, §X.A.

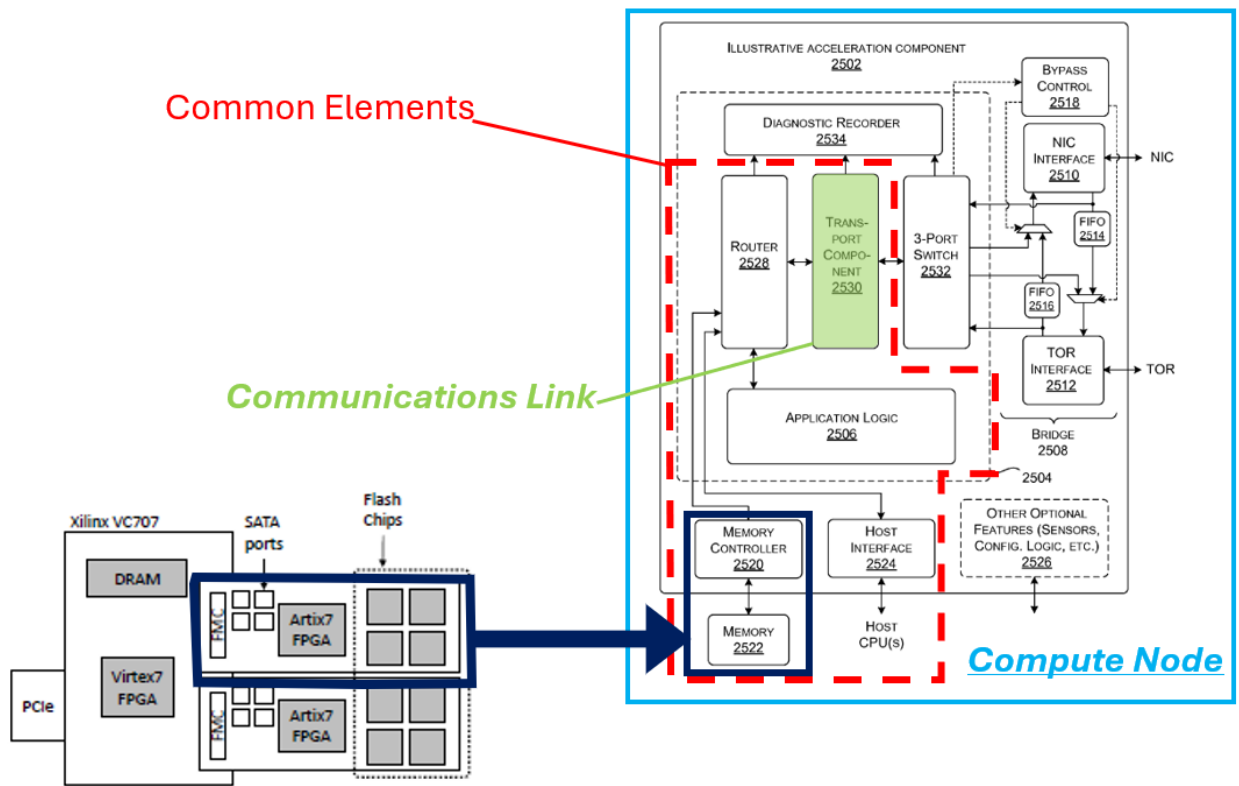


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and
AMD1010, FIG. 25).

157. Heil explains that “[e]ach acceleration component” within a respective server unit component (*compute node*) can be “*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. 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.” AMD1010, [0089], FIG. 8; *see also* AMD1011, 5 (“[a]ny network topology”); FIG. 5.

158. 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.

159. 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].

160. 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 exceedingly apparent and desirable for implementing the service shown in Figure 36 (and other embodiments). AMD1010, [0089] *see also* AMD1010, [0084] (“the principles set forth herein can be applied to smaller systems”). In fact, the same inventors demonstrated this exact example being implemented on “one dimension of the torus” in the related paper for Microsoft’s project “Catapult.” AMD1014, 6-9, FIG. 5; AMD1010, [0218]–[0223], FIG. 36; *see also supra* §X.C. I note that a 1D torus network topology is a linear arrangement of nodes where each node is connected to its two immediate neighbors, and the two end nodes are linked together to form a continuous ring.

161. 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]–[0213], FIG. 36. In fact, a 1-dimensional serial network would have been an efficient physical layout to support the example service shown in Figure 36.

162. Fourth, the POSITA would have recognized that a 1-dimensional serial network topology would have reduced the cabling complexity of the data processing system. *See* AMD1014, 3 (demonstrating that there is a tradeoff between “routability and cabling complexity” between different number of dimensions in torus networks)).

163. 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. While theoretically, the maximum dimensions for a torus network topology is unlimited, a POSITA would have understood that is given the practical computing and networking constraints the number of dimensions would have been limited to 5D or 6D. AMD1032 (describing a five-dimensional example for a super computer). In this context, a POSITA would have recognized that the options were limited to 1D (*e.g.*, a single ring *see* AMD1010, FIG. 36, AMD1014), 2D (a grid/mesh *see* AMD1010, FIG. 8), 3D (cube *see* AMD1010, FIG.

34), or a complex super computer implementations with 4 to 6 dimensions (AMD1032). From this finite number of options, it would have been obvious to a POSITA to select the 1D topology and that such an arrangement would have a reasonable expectation of success because it minimizes the coordination of different systems and routing logic.

(iii) *Element [1b]: a data store comprising records associated with a data processing operation;*

164. The Heil-Jun Combination renders obvious Element [1b] and Claim 1 as a whole. As I discussed at Section X.C, the Heil-Jun Combination integrates Jun’s teachings to utilize NAND Flash Storage with Heil’s acceleration components. Accordingly, in the Combination, the accelerator component in each of Heil’s pipelined server unit components (*compute nodes*) includes the “flash storage” (*data store*) taught by Jun. AMD1011, p.4, FIGS. 2–3, 9, 11; *see also* AMD1011 pp.2–4. In more detail, Jun’s incorporated teachings include using NAND flash storage in place of DRAM with a reconfigurable hardware acceleration node. AMD1011, pp.4, 8, FIGS. 2–3, 9, 11.

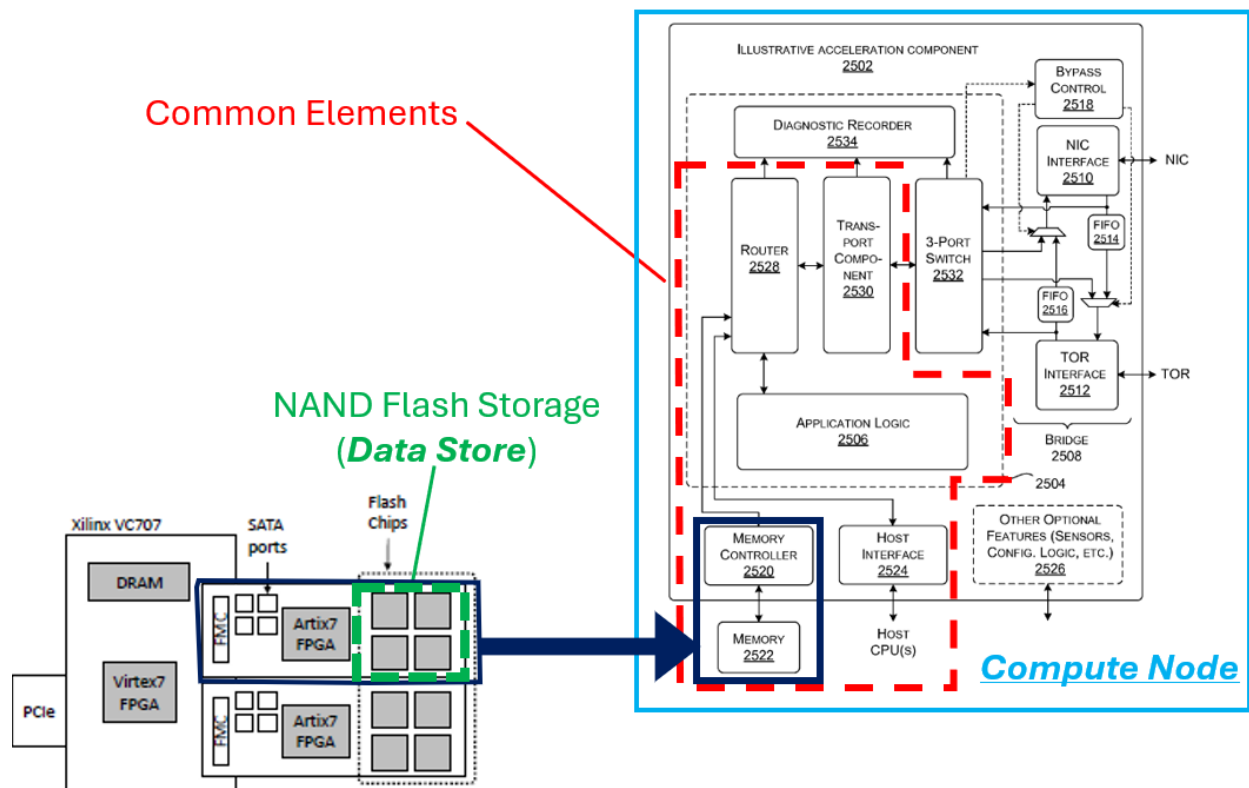
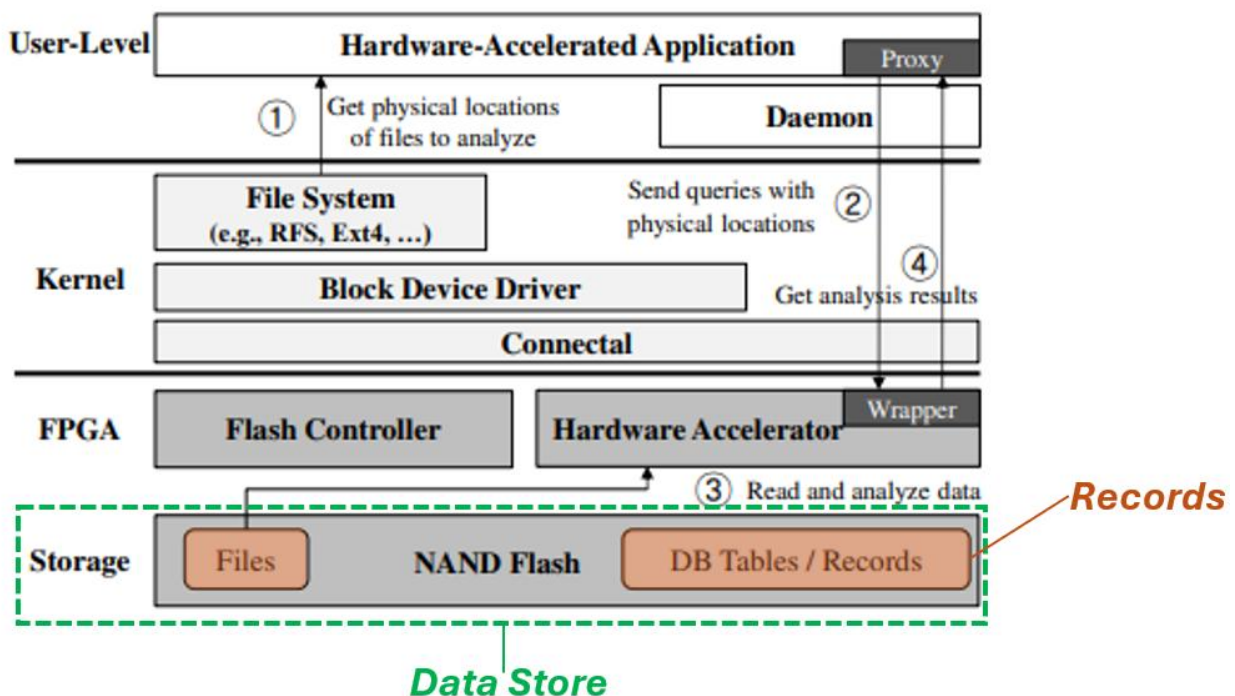


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and AMD1010, FIG. 25).

165. Jun also provides teachings regarding the claimed *records associated with a data processing operation*. For example, Jun’s NAND flash storage (*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 the acceleration components taught by Heil. AMD1011, 4, 7–8, 10, 12, FIG. 8. Also like Heil, the data stored in the NAND Flash data (e.g., including the “Files” and “DB Tables/*Records*”) is associated with a *data*

processing operation. See e.g., AMD1011, pp.4 (“with the in-store processor to perform computation on the data”), 7 (“Applications can then provide in-storage processors with a stream of physical addresses (see (2)), so that the in-storage processors can directly read data from flash.”), 10 (“a LSH query accelerator, where all of the data is stored in flash and the distance calculation is done by the in-store processor on the storage device.”), 12 (“consults the file system for a list of physical addresses of the files to search (haystack). This list is streamed to the accelerator, which uses these addresses to request for pages from the flash controller.”); *see also* AMD1001, 8:25–33 (“as is generally known in the art, these data are typically stored or maintained in files or datasets (or ‘records’ or ‘pages’)”).



AMD1011, FIG. 8 (annotated).

166. As I discussed, the Heil-Jun Combination incorporates Jun's NAND Flash (*data store*) with the acceleration component in each of Heil's pipelined server unit components (*compute nodes*). *Supra*, §X.C. Moreover, a POSITA would have understood and found obvious to utilize Jun's NAND Flash for storing *records associated with a data processing operation*, including, for example, one or more of the operations shown in the example of Heil's Figure 36. AMD1010, [0218]–[0222] (“The next two acceleration components (3606, 3608) perform more complex feature computations.”), [0223] (“The last three acceleration components (3612, 3614, 3616) generate a final ranking score using all of the feature values computed thus far as input information to the scoring calculation, and using any environment-specific score-calculation equation or algorithm”) [0229] (“the particular acceleration component performs its assigned *operation*”), FIGS. 36, 40; *see also* AMD1010, [0133]–[0140], [0157], [0161]–[0162], [0175], [0178], [0248]; *see also supra*, Elements [1pre], [1b].

167. In one example, Jun describes a “[s]tring search operation” that is “often used in database table scans” AMD, 1011, 12. In this example, a POSITA would have understood and found obvious that a database record includes a collection of fields and that a database table stores database records rows, where each column stores the values for one field of the database records. Accordingly, Jun teaches that NAND flash storage (the *data store*) comprises database records

associated with a database table scan (e.g., as part of the string search operation). Moreover, a POSITA implementing the example in Heil's Figure 36 (e.g., a multiple component service that "assigns a ranking score to a pairing of a query and a document") would have included an analogous operations that utilizes database records, for example, to extract features from the database records. AMD1010, [0218]–[0223], FIG. 36; *see also* [0127] ("perform a search algorithm").

168. I note that Heil also teaches that data is communicated between various components of the system via "packets" carrying data payloads for processing by the respective acceleration components of the pipelined server unit components (*compute nodes*). AMD1010, [0057], [0078] (explaining that acceleration components "perform processing on packets"), [0082], [0163], [0168]–[0170], [0182]–[0193], [0218]–[0223], [0229], FIGS. 36, 40. The acceleration component 2502 depicted in Heil's Figure 25 includes a communication path between the router 2528 that receives incoming data packets and the local memory 2522 (Jun's flash storage (*data store*) in the Combination), indicating that packets are stored there (as *records*) for data processing (e.g., the data processing operations discussed above). AMD1010, [0164], [0167], FIG. 25. The Heil-Jun Combination also renders obvious [1b] on this basis.

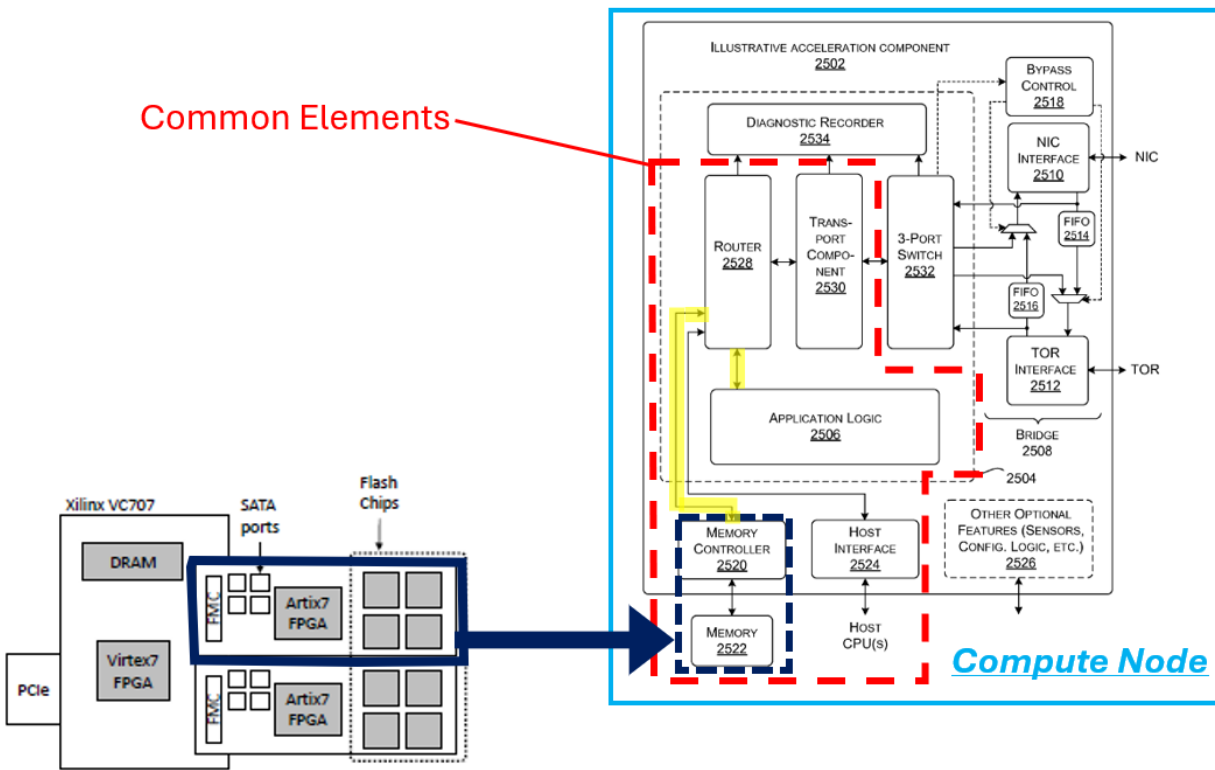


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and AMD1010, FIG. 25).

- (iv) *Element [1c]: a programmable logic component to execute the data processing operation in cooperation with the data store;*

169. The Heil-Jun Combination renders obvious Element [1c] and Claim 1 as a whole. For example, each of Heil’s respective acceleration components within the pipelined server unit components (*compute nodes*) can be implemented using “field-programmable gate array (FPGA) devices” and include reconfigurable “application logic” (*programmable logic component*). AMD1010, [0050], [0139] (“reconfiguring...the application logic”), [0157]–[0158] (“reconfiguration of the

application logic”), [0160]–[0162] (explaining that the application logic represents the resource most subject to change in the acceleration component), [0229], [0252], FIGS. 23–25; *see also* AMD1001, FIG. 4, 15:3–30, 37:61–67. Because Heil’s application logic is implemented on an FPGA device, it is a logic component programmable by reconfiguring its internal hardware connections. *Id.*

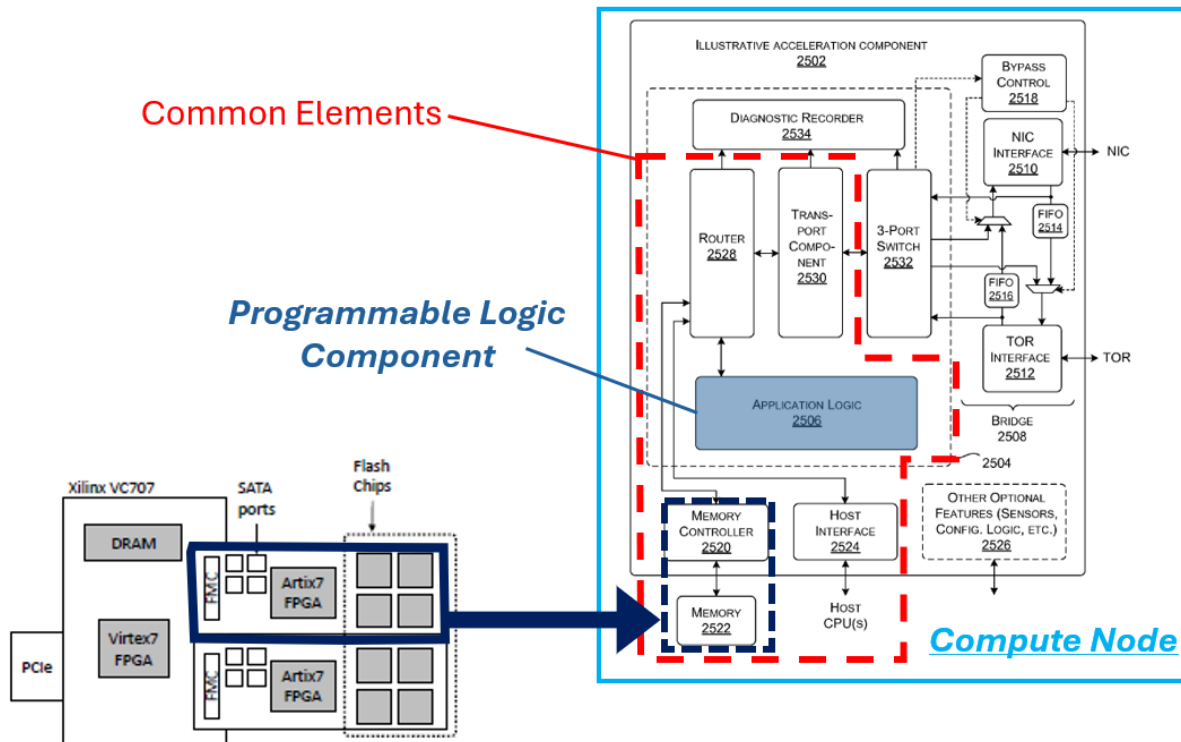
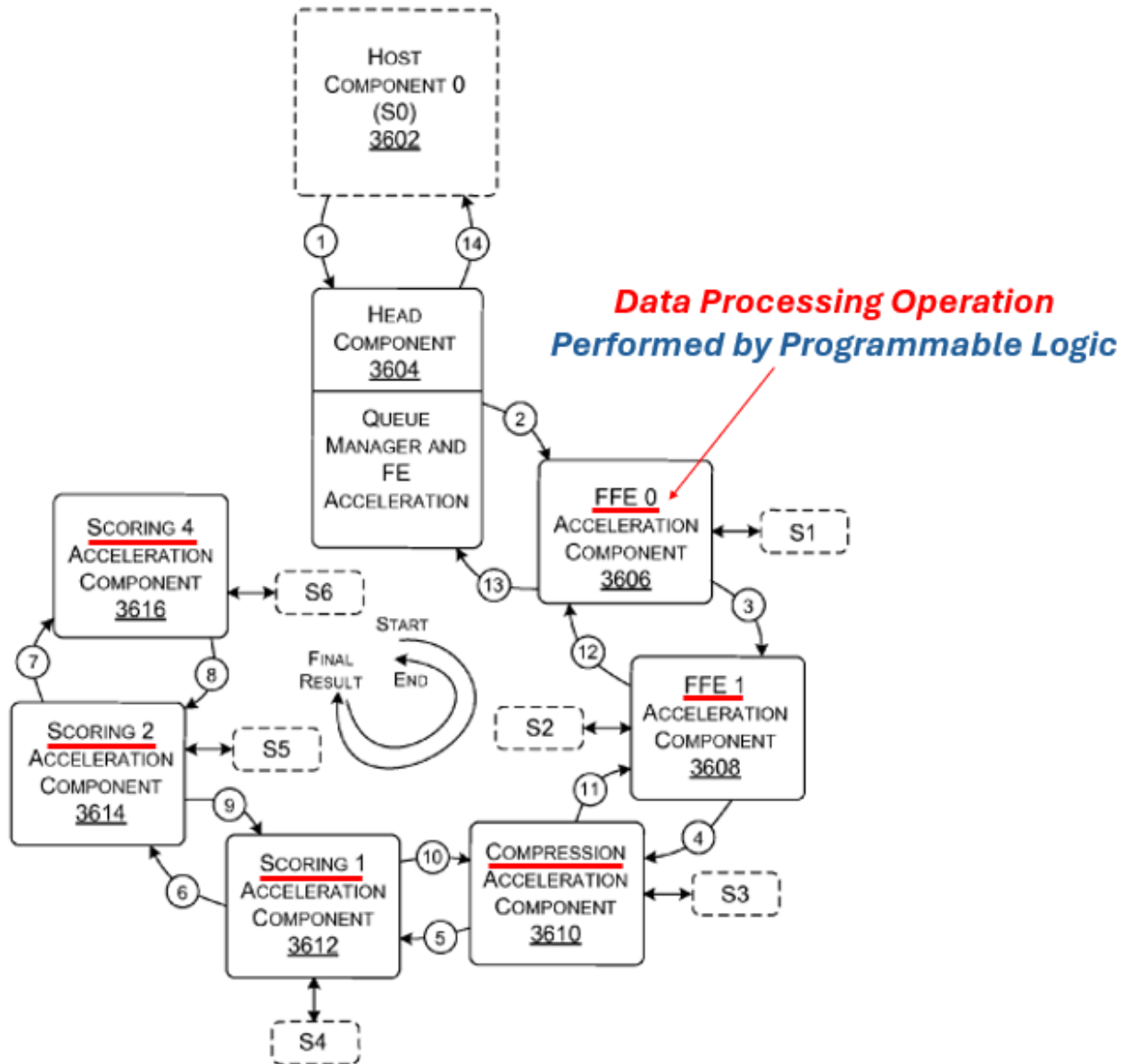


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and AMD1010, FIG. 25).

170. Heil also teaches that each acceleration component, via its configured application logic, performs a respective data processing service or operation

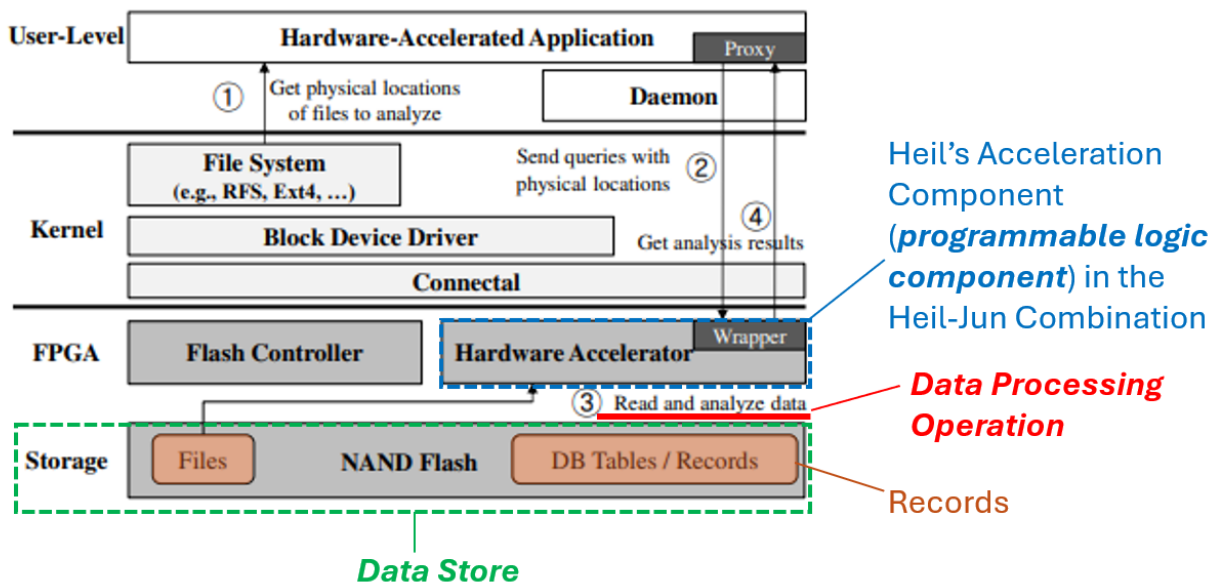
(executes a data processing operation). *Supra*, Elements [1pre]–[1b]; AMD1010, [0222] (“The next two acceleration components (3606, 3608) perform more complex feature computations.”), [0223] (“The last three acceleration components (3612, 3614, 3616) generate a final ranking score using all of the feature values computed thus far as input information to the scoring calculation, and using any environment-specific score-calculation equation or algorithm”); [0229] (“the particular acceleration component performs its assigned *operation*”), FIGS. 36, 40; *see also* AMD1010, [0133]–[0140] (discussing “logic” on the acceleration component “for performing [a] service”), [0157] (explaining how the “application logic” is governed by a “model” that is implemented via a “configuration image[]” and that “corresponds to logic that performs a function in a particular manner”), [0161]–[0162], [0175], [0178] (noting “services [that] are implemented by the application logic”), [0248] (“The hardware acceleration component further includes logic configured to perform a part of the multi-component service to provide output information, based on the input information that is received.”).



AMD1010, FIG. 36 (annotated).

171. As I discussed at Element [1b], the Heil-Jun Combination includes the feature of storing various *records* in Jun's NAND Flash Storage (*data store*) for performing the *data processing operation* assigned to the respective acceleration component of Heil's server unit components (*compute nodes*). Accordingly, and in

view of my discussion here at Element [1c], the Combination further provides that the application logic (*programmable logic component*) of each acceleration component operates *in cooperation with* Jun's NAND Flash Storage (*data store*) to execute its assigned *data processing operation*.



AMD1011, FIG. 8 (annotated).

- (v) *Element [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,*

172. The Heil-Jun Combination renders obvious Element [1d.i] and Claim 1 as a whole. As I discussed, Heil's server unit component (*compute node*) includes a local host component and local acceleration component. *Supra*, §X.A, Element [1pre]. 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*.

173. In Heil's disclosure each 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."), FIG. 21; *see also*, [0156]–[0157] (explaining how the application logic is implemented via a "configuration image[]" and that "corresponds to logic that performs a function in a particular manner"), [0124] ("A configuration stream specifies the logic to be 'programmed' into a recipient acceleration component."), FIG. 22.

174. Heil further teaches a method in which the local host component coordinates this configuration by "forwarding a configuration stream (corresponding to a configuration image) to the acceleration component." AMD1010, [0155] ("the local host component 2112 can perform the above-described [configuration-coordination] operation"). To transfer data to its counterpart local acceleration component, the local host component places *data and instructions*—here, the configuration image/stream—in the main (*node*) *memory*, so the *data and*

instructions can be retrieved by the acceleration component using PCIe with DMA (direct memory access). AMD1010, [0165], FIG. 25; *see also*, [0155] (“Each configuration image contains logic that can be used to implement a corresponding service.”), [0173]–[0175], FIG. 27, *infra* Element [1e]. In additional, Heil’s disclosure makes clear that 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. *See* AMD1010, [0174], [0195] (identifying “RAM of any type(s)” as “storage resources” for the host’s CPU to access “code, settings, data, etc.”), FIGS. 27, 31.

175. I note that a POSITA would have understood or found obvious that a configuration stream includes data defining the logic and instructions for configuring the logic. *See* AMD1046, 3:38–42 (“*instructions and data* (e.g., a configuration image)”). For example, the configuration image includes configuration logic data (e.g., logic bits) and configuration instructions (e.g., protocol level commands) to configure the configuration logic at the FPGA. *See* AMD1015 pp.2, 23, AMD1027 pp.8–10, 12, 38, 47–79, 95, 118; AMD1046, 3:38–42 (“*instructions and data* (e.g., a configuration image)”)); AMD1010, [0124], [0155]–[0158], [0171]–[0172].

176. 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*. See AMD1010, [0195], [0148]–[0152]; *supra*, Elements [1b]–[1c] (where I describe example *data processing operations* performed by configured application logic including the example data processing operations shown in FIG. 36). For example, Heil explains that “[t]he host component 3102 can also include any storage resources 3106 for storing any kind of information, such as code, settings, *data*, etc.” and “carry out associated *instructions* stored in any storage resource or combination of storage resources.” AMD1010, [0195], *see also* [0148]–[0152] (explaining examples where each host component implements “a local management components” including features for storing “logic and information.”).

177. The Heil-Jun combination renders *data and instructions to facilitate inter-node data communications via the communications link* obvious via Heil’s teachings that the acceleration component of each server unit component (*compute node*) receives “setup information” for establishing *inter node* connections. AMD1010, [0229], *see also* [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]. Heil teaches that 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”), *see also* AMD1010, [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). AMD1010, [0165], FIG. 25, *see also* AMD1010 [0173]–[0175], *infra* Element [1e]. Accordingly, as would have been apparent or 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.*, *see* AMD1010, [0181], [0229], [0248]. In addition and as an alternative basis, a POSITA would have 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***. *See* AMD1010, [0174], [0191] (“application logic 2506 can regulate the flow of packets”), [0195], FIGS. 27, 31.

178. The ***data and instructions*** associated with the configuration image in Heil’s local host main memory (***node memory***) also ***facilitate inter-node data communications via the communications link***, e.g., by ensuring that data is appropriately processed before being transferred to the next node in the pipeline and by regulating the flow of packets. For example, as I explain below in my analysis

for claim 11. *Below* Element [11]. For example, executing the assigned operation based on the configuration image causes the processed data to (1) flow through the acceleration component, and (2) to a next acceleration component of the next server unit component (*compute node*) in the multi-component service via the “transport component” (*communications link*). AMD1010, [0167]–[0168], [0191] (“the application logic 2506 can regulate 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 constituent acceleration unit is appropriately configured to perform the correct data processing.

- (vi) *Element [1d.ii] wherein the node memory further comprises algorithm memory including resource-specific memory locations to support the data processing operation;*

179. The Heil-Jun Combination renders obvious Element [1.d.ii] and Claim 1 as a whole. 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 explains that the main memory includes “a kernel-pinned input buffer 2708 in main [*node*] *memory*.” AMD1010, [0173]–[0175]. 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. compare with* 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; *compare with* 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. 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] (“a pinned memory space mapped to the coprocessor memory...[e]ach entry may point a logical block address of a space with a predetermined size in the pinned memory space, and the logical block address may be mapped to a physical block address of a space with a predetermined size in the coprocessor memory.”)).

180. Additionally, the “kernel-pinned input buffer” is an *algorithm memory* at least because operates to support the data processing operation, for example, by communicating instructions and data for a configuration image and setup information as discussed in Element [1d.i]. *See also* AMD1010, [0127], [0216] (“the

multi-component service corresponds to an...*algorithm*"); [0218] (explaining that "the multi-component service assigns a ranking score to a pairing of a query and a document" and that such an algorithm was traditional "performed entirely in software by applying a model produced by machine-learning"), [0219]–[0223] ("score-calculation equation or *algorithm*"); *infra* Element [1h] ("computationally expensive algorithm"). 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; *see also* AMD1001, 18:50-65; 40:25-41 ("as is generally known"), 53:8:21.

(vii) *Element [1e]: a data mover component to facilitate intra-node data communications between the programmable logic component and the node memory; and*

181. The Heil-Jun Combination renders obvious Element [1e] and Claim 1 as a whole. As I 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], [0089] ("Like the data processing system 102 of FIG. 1, 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; *see also*, [0165]. 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*). *See* AMD1010, [0173]–[0175] (explaining PCIe/DMA communication from buffers from the local host main memory to the local acceleration component’s application logic), FIGS. 25, 27; *see also* [0085], [0089], [0124], [0155]–[0156], FIGS. 6, 8, 22; *supra*, Elements [1d.i]–[1d.ii]; *infra* Elements [7]–[8]; AMD1001, 16:3-10 (explaining that the data mover is configured to facilitate data communications).

182. Accordingly, a POSITA would have found obvious that Heil’s compute node includes a PCIe and DMA based data mover component *to facilitate intra-server unit component data communications between the programmable logic component and the node memory* including, for example, to configure the *application logic* (programmable logic) with a configuration image from the main memory (*node memory*). AMD1010, [0085], [0089], [0124], [0155]–[0156], [0174]–[0175].

183. A POSITA would have understood and found obvious that PCIe and DMA based memory access would have included elements such as s “PCIe Core[s]” and controller (e.g., “DMA Engine,” “PCIe controller”). *See* AMD1014, FIG. 3, p.4

(describing a PCIe interface with DMA support) *see also* AMD1014, pp.1–3, 7 (“A PCIe DMA transfer moves the score, query ID, and performance counters back to the host”); AMD1011, pp.4 (“a host interface implemented over PCIe”), 5, (“PCIe DMA”), 6 (“DMA Engine”), 8–11, FIGS. 7 (“Host-FPGA interface over PCIe” and “DMA Engine”), 9, 10.

(viii) *Element [1f]: a storage interface component to facilitate intra-node data communications between the programmable logic component and the data store;*

184. The Heil-Jun Combination renders obvious Element [1f] and Claim 1 as a whole. As discussed, the Heil-Jun Combination incorporates Jun’s Flash (*data store*) to serve as the local memory for Heil’s acceleration component. *See supra*, §§X.B–C, Element [1b]; AMD1011, pp.4–5, 8, FIGS. 2–4, 10. Jun explains that 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 (explaining the benefits of the hardware interface, including “cutting down on latency,” “exposing all degrees of parallelism,” and “allowing higher level system stacks”), 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*.

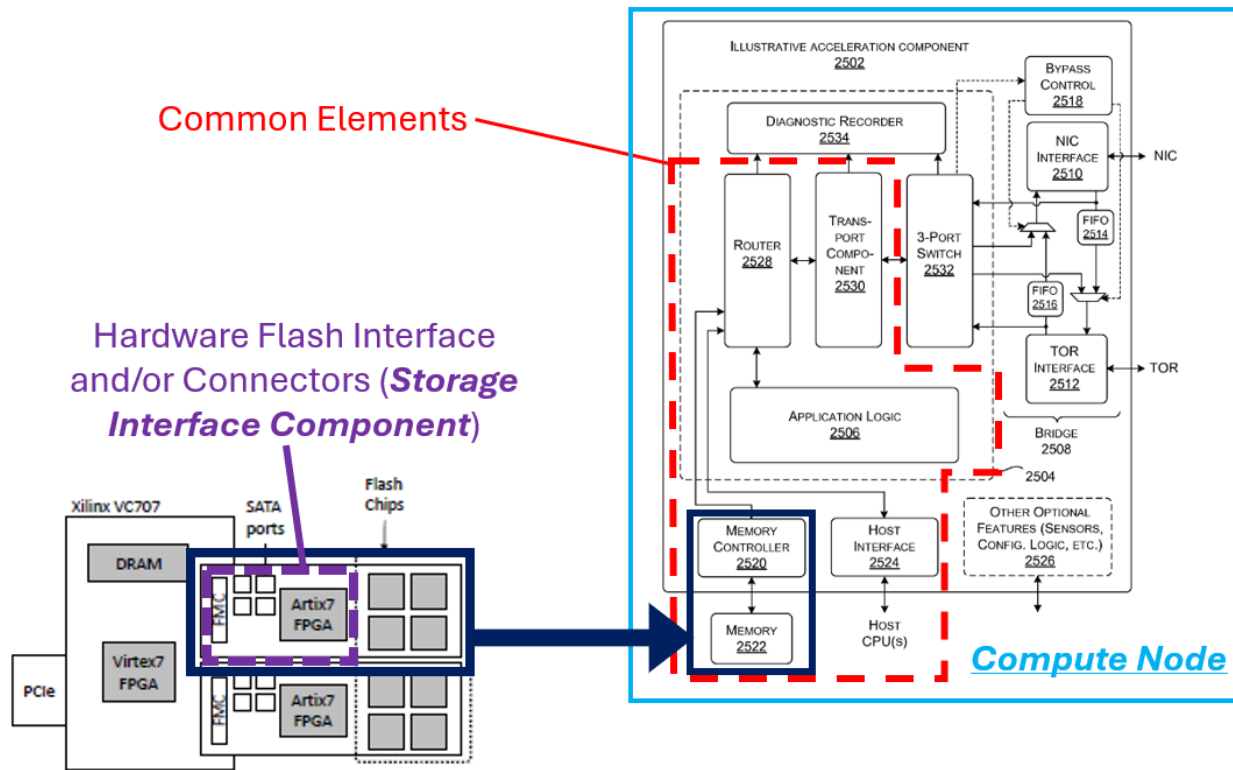
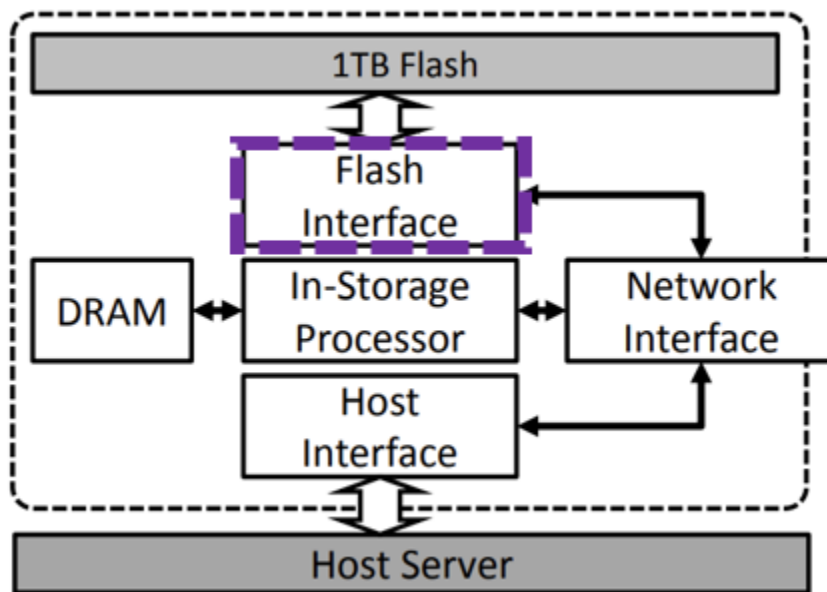


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and
AMD1010, FIG. 25)



AMD1011, FIG. 2 (annotated)

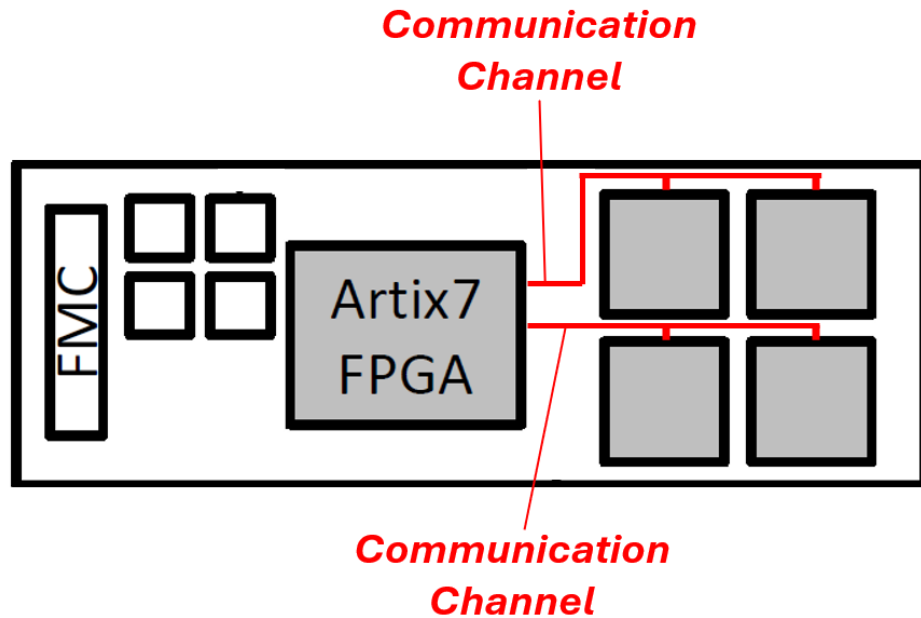
185. In the Heil-Jun Combination Heil's acceleration components incorporate Jun's flash board, which a POSITA would have understood included a *storage interface component to facilitate intra-node data communication between Jun's NAND Flash Storage (data store) and the application logic (programmable logic component)*. See *supra*, §X.C, Elements [1b]–[1c]; see also AMD1001, 11:19–21 (the '636 patent states that the “data store interface 145 manages access...to data store 143, as is generally known in the art.”), 16:18–17:48 (“as is generally known in the art,” and “in accordance with known principles”); 39:65–40:8.

- (ix) ***Element [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***

186. The Heil-Jun Combination renders obvious Element [1g] and Claim 1 as a whole. As discussed, the Heil-Jun Combination incorporates Jun’s teachings of a flash board including Flash (***data store***) in conjunction with suitable connectors and hardware interface (***storage interface component***). *Supra*, §§§X.B–X.C, Elements [1b]–[1c], [1f].

187. Jun explains that an Artix-7 FPGA manages data transfer from the Flash Storage utilizing ***multiple parallel communication channels*** to access NAND Flash Chips included in the Flash Storage. AMD1011, FIG. 10 (*see below*), pp.4 (“Flash devices or SSDs achieve high bandwidth by grouping multiple flash chips into several ***channels***, all of which can operate in parallel”), 6 (“data reads from flash chips on multiple buses in parallel”) 9; *see also* AMD1012, FIG. 2-3 . In this manner for the Heil-June Combination, Jun’s storage interface provides high bandwidth flash storage access (e.g., to Heil’s application logic) by executing multiple commands that operate in parallel across the multiple channels. *Id.*

Jun's Flash
Board →



AMD1011, FIG. 10 (modified & annotated).

188. This understanding of Jun's flash board is confirmed by AMD1012 (Liu). Liu is directed to the same type of flash-based compute nodes (BlueDBM node) disclosed in Jun including the flash board. See AMD1012, pp.3, ("BlueDBM...Big Data flash storage platform), 16–18, 23-28 ("The BlueFlash board is the underlying flash storage device in BlueDBM. We expect to use the platform for the next 3-4 years in the exploration of Big Data applications at scale.")³; compare with AMD1011; *Supra*, §§X.B; X.D (Elements [1b], [1f]–[1g]).

³ I note that Liu published in September 2014, several months before Jun published in June 2015. AMD1012, p.1; AMD1011, p.2

Liu demonstrates multiple channels between NAND Flash Chips and the Atrix-7 FPGA as part of a BlueDBM node's flash board. AMD1012, FIG 2-3 (below), pp. 23-28. Liu also explains the advantages of utilizing parallelism (e.g., "Bus parallelism") to achieve "a significant amount of bandwidth." AMD1012, p.24, *see also* AMD1012, p.26 (providing specifications for a BlueDBM flash board indicating that eight NAND Buses were selected). Accordingly, a POSITA would have also found obvious that Jun's flashboard employs multiple communications channels to the NAND Flash Chips.

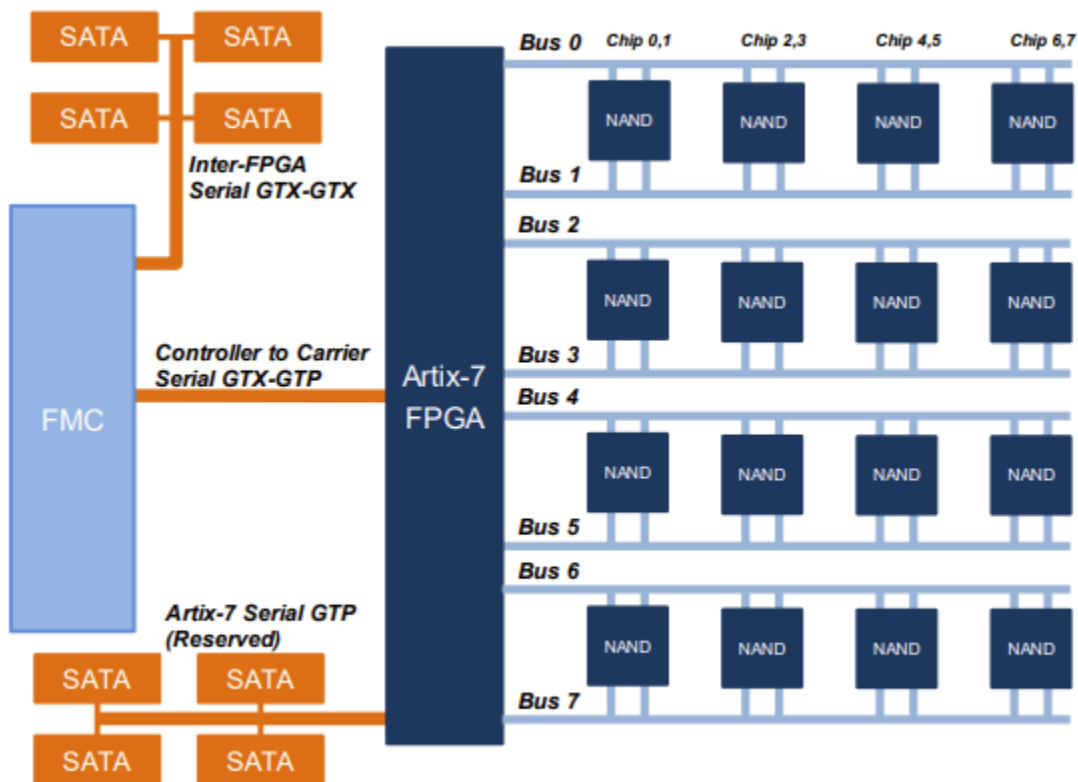


Figure 2-3: Flash Board Architecture

AMD1012, FIG. 2-3.

189. The Heil-Jun Combination also renders Element [1g] obvious on the basis of Jun's teachings that "[t]he communication between the flash board and the Virtex 7 FPGA processor is done by a 4-lane aurora **channel**." AMD1011, p.8, FIGS. 2 ("Flash **interface**), 10. According to Jun, the "flash board also hosts 8 SATA connectors, 4 of which pin out the high-speed serial ports on the host Virtex 7 FPGA, and 4 of which pin out the high-speed serial ports on the Artix 7 chip." AMD1011, p.8. Accordingly, Jun's flash board includes a **storage interface component** that **utilizes a plurality of communication channels** (e.g., 4-lane aurora **channel** and/or SATA connectors) **to transfer data** to and from Jun's NAND Flash Storage (**data store**). See *supra*, §§X.B–C.

- (x) ***Element [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.***

190. The Heil-Jun Combination renders obvious Element [1h] and Claim 1 as a whole. As I discussed for Elements [1b]–[1c], the Heil Jun Combination teaches a **programmable logic component** that executes the **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], *see also* [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*. See AMD1010, [0229], FIG. 40. I understand that 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* V.B.

191. Heil teaches executing a multi-component service including 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; *see also* [0040], [0065], [0068]–[0069], [0071], [0073]–[0075]; [0083], [0144], [0151], [0198]–[0210], [0225].

2. Claim 2

- (i) *Element [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.*

192. The Heil-Jun Combination renders obvious Claim 2. As I 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 “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*. . For example, AMD1014 (directed to the same project as Heil) demonstrates that “[f]our high-speed *serial* links running *SerialLite III* (SL3), a lightweight *protocol* for communicating with neighboring FPGAs.” AMD1014, 4; *see also* AMD1011, p.4 (describing the system in AMD1014 and explaining that the FPGAs “form a network among themselves via high-speed *serial* links...”), 9. Accordingly, a POSITA would have understood and found obvious to *employ a serial communications protocol* to communicate over Heil’s dedicated links.

193. Heil also discloses the transport component (*communications link*) utilizing Ethernet to communicate with remote acceleration components. AMD1010, [0184]. I note that Ethernet is a *serial protocol*. AMD1047, p.310 (“Ethernet is a *serial protocol*”). For at least the reasons I outlined above, a POSITA would have been motivated to implement the same or a similar serial protocol at the transport component (*communications link*) when the acceleration components are communicatively computed via “dedicated links.” See AMD1010, [0089], [0180]–[0184], [0186], [0218]–[0223], FIG. 36. AMD1014, 3–4; AMD1011, 4, 9.

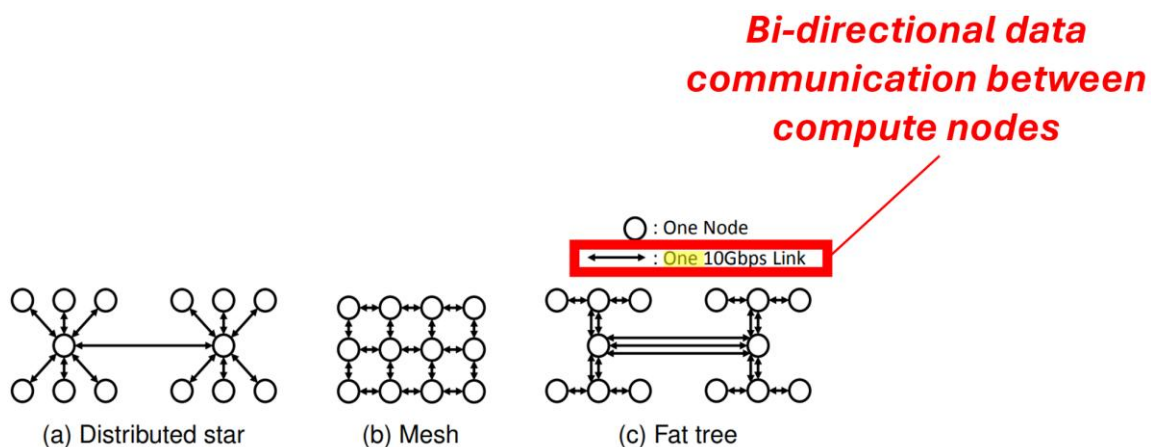
194. A POSITA would have also immediately recognized that this communication is *bi-directional* at least because Heil’s torus network architecture dictates bi-direction communication in order to connect neighboring nodes via dedicated cable links. See AMD1010, [0218]–[0223], FIG. 36 (demonstrating the 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), [0089]; *supra* 1A. For example, Heil describes the transport component (*communications link*) sending and receiving packets. AMD1010, [0180]–[0183]. Heil also explains that the transport component(*communications link*) includes a failure handling component that “buffers all sent packets until it receives an acknowledgement (ACK) from the receiving node.” AMD1010, [0186]. This requires *bi-directional communication* with the receiving node (e.g., in order to

send the packet and receive the acknowledgement). Heil also shows as an example that the communication links (transport component) can use Ethernet which is a *bi-directional serial communication protocol*. AMD1010, [0180]–[0184]. Additionally, AMD1014 (directed to the same project as Heil), explains that “each inter-FPGA network link supports 20 Gb/s of peak *bidirectional* bandwidth.” AMD1014, p.3.

195. The Heil-Jun Combination also renders claim 2 obvious one the basis of 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 (showing *Serial* Advanced Technology Attachment (“SATA”) cables). For example, the ’636 patent explains that communication between the host and router is *bi-directional* and listing SATA as an example for providing such communication, confirming that SATA was a known serial protocol for enabling bi-directional communication. AMD1001, 9:47–10:40. 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*.

196. I note that a POSITA would have understood that a serial protocol would be employed in a system using high-performance *serial* links with *serial* transceivers. I note that a POSITA would have recognized that "SATA" stands for "*Serial* Advanced Technology Attachment" and relates to a technology and protocol for transmitting data one bit at a time allowing for faster speeds and thinner cables. See AMD1033, 12, 24 ("Serial ATA is a high-speed serial link replacement for the parallel ATA attachment"), 130 ("sent serially bit by bit"). A POSITA would have also recognized that SATA uses a serial protocol to operate. See AMD1033, 1, 12, 14, 28, 87, 201, 278. Notably, a POSITA would have recognized that the S in SATA specifically relates to the type of *protocol*. See AMD1033, 1, 12, 87 ("*serial* ATA protocol").

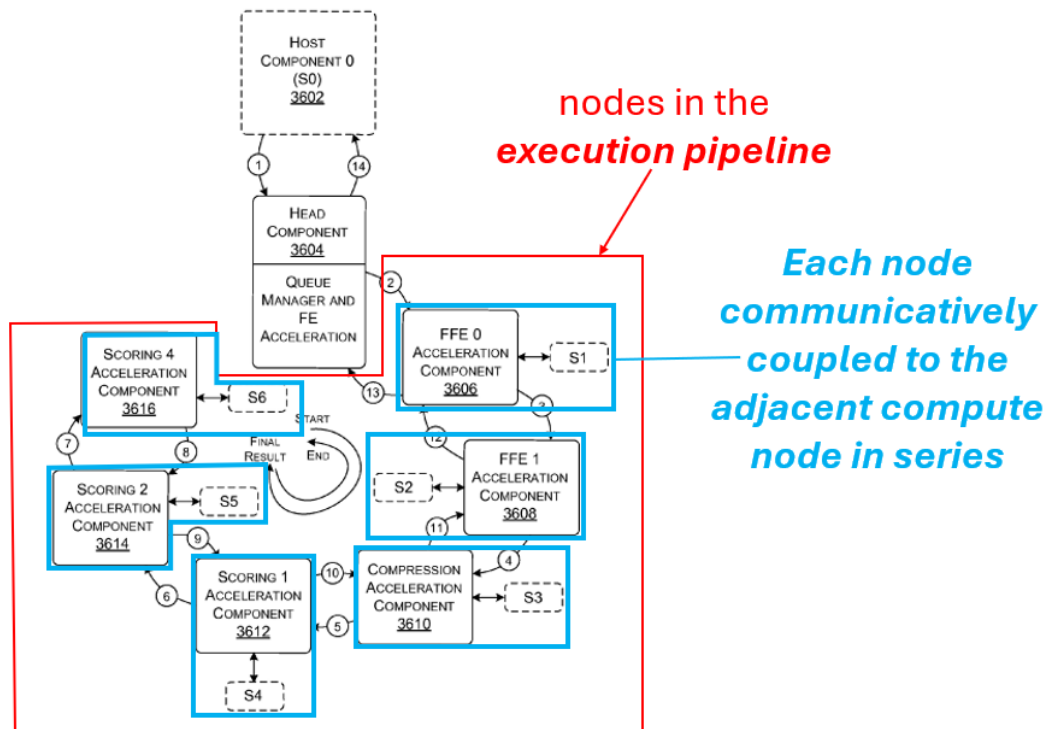


AMD1011, FIG. 5. (annotated).

3. Claim 3

- (i) *Element [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.*

197. The Heil-Jun Combination renders obvious Claim 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*.



AMD1010, FIG. 36 (annotated).

198. The server unit component is *communicatively coupled* to the *adjacent* server unit component *in series* via a transport component (*communications link*).

AMD1010, [0068] (“processing *pipeline*”), [0074], [0084], [0089], [0218]–[0223], FIGS. 25 (annotated below); 36 (annotated above).

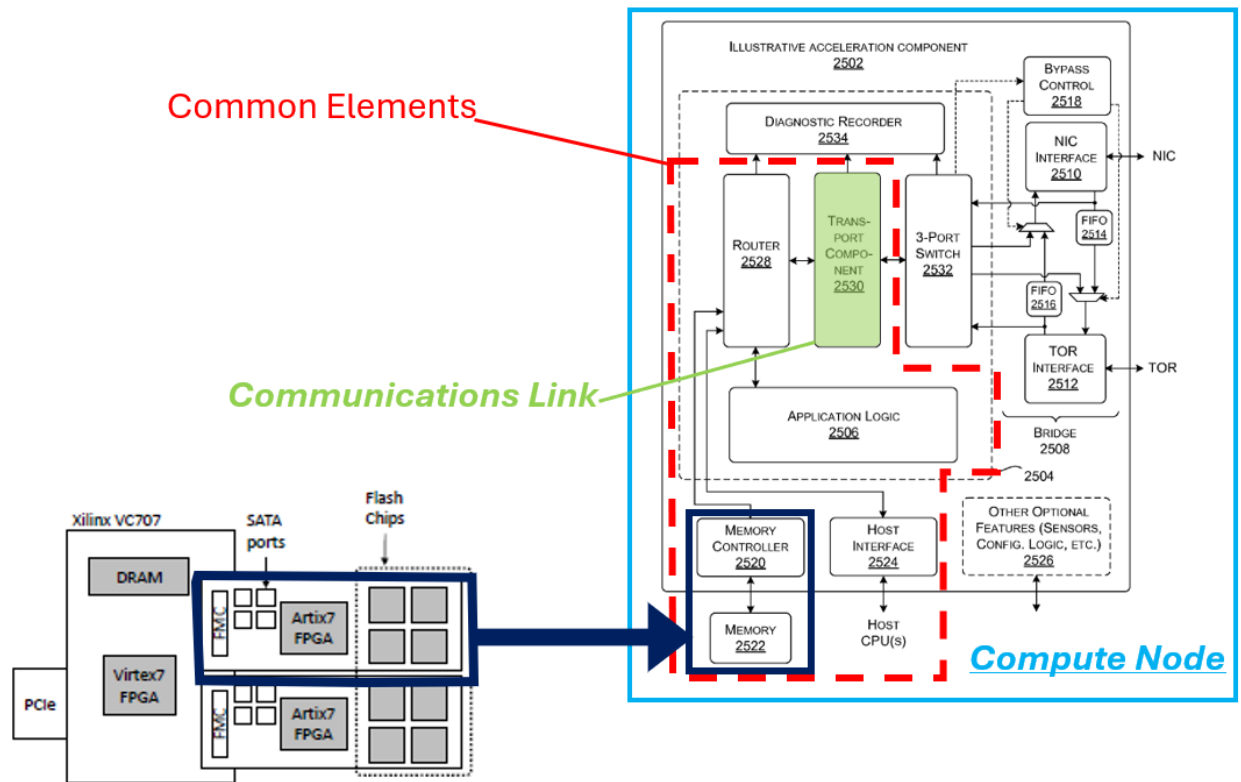


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and AMD1010, FIG. 25).

4. Claim 4

- (i) *Element [4]: The compute node of claim 1 wherein the data comprise an Electronically Erasable Programmable Read Only Memory (EEPROM) device.*

199. As I discussed, the Heil-Jun Combination incorporates Jun’s Flash (*data store*) for the server unit component (*compute node*). *Supra*, §§X.B–C , Elements [1b], [1f]–[1g]. As a POSITA would have known, NAND Flash Storage

is an *EEPROM device*.. See AMD1001, 10:65–11:21 (“all Flash is EEPROM”). A POSITA would have understood and found obvious that Flash is a specific high-density EEPROM technology. See AMD1034, 1:35-38 (“A Flash memory is a type of EEPROM”).

5. Claim 6

- (i) *Element [6]: The compute node of claim 1 wherein the programmable logic component comprises a field programmable gate array (FPGA).*

200. The Heil-Jun Combination renders obvious Claim 6. As discussed at Element [1c], each of Heil’s server components includes “application logic” (*programmable logic component*) as an element of the acceleration component, which is implemented as an FPGA device. See also AMD1010, [0049]–[0050] (“a primary example in which the acceleration components correspond to FPGA devices”), [0139], [0157]–[0158], [0160]–[0162], [0167], [0175]–[0178], [0252], FIG. 25.

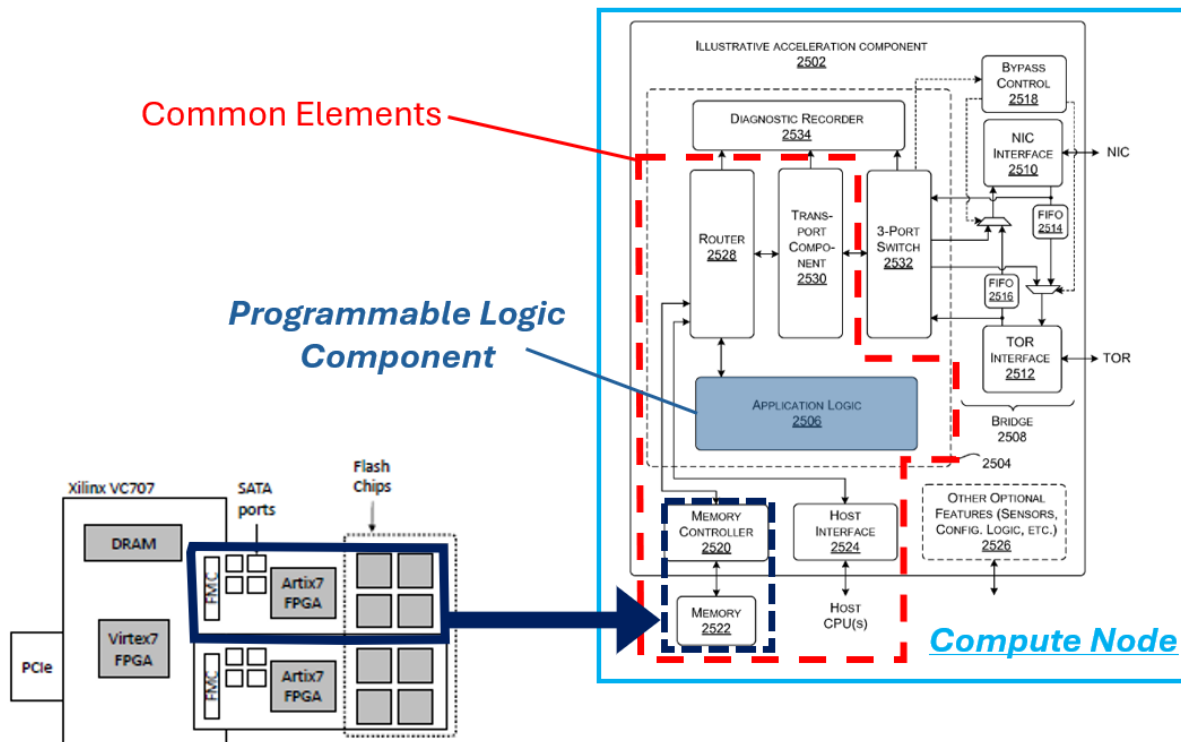


Illustration of the Heil-Jun Combination (based on AMD1011, FIG. 10 and AMD1010, FIG. 25).

201. A POSITA would have understood and found obvious that the application logic, which includes programmable/reconfigurable gate arrays based on Heil's acceleration component being an FPGA. A POSITA would have also understood that in examples where an FPGA device is portioned to include a "shell" (e.g., "a reusable portion of programmable logic common across applications") and a role (e.g., "the application logic itself, restricted to a large fixed region of the chip"), the role region on a chip *comprises a field programmable gate array (FPGA)*. See AMD1014, 4, FIG. 3.

6. Claim 7

- (i) *Element [7]: The compute node of claim 1 wherein the compute node further comprises a processor to manage operation of the data mover component.*

202. The Heil-Jun Combination renders obvious Claim 7. As I discussed for Element [1e], each of 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. *See also* AMD1010, [0165]. Heil further explains that each "[local] host component 2702 may forward 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.*, *see also* [0195] ("host component 3102 can include one or more processing devices 3104...which may implement one or more hardware threads."), [0048], [0051], [0054], [0077] (local host component includes "one or more central processing units (CPUs)"), [0133], [0248]; *see also* AMD1014, pp.2–3 ("to interface the board to the host CPU over PCIe"), 4 ("communication interface between the CPU and FPGA must satisfy two key design goals: (1) the interface must incur low latency, taking

fewer than 10 μ s for transfers of 16 KB or less, and (2) the interface must be safe for multithreading. To achieve these goals, we developed a custom PCIe interface with DMA support” and “To send data to the FPGA, a thread fills its slot with data, then sets the appropriate full bit for that slot. The FPGA monitors the full bits and fairly selects a candidate slot for DMA’ing into one of two staging buffers on the FPGA, clearing the full bit once the data has been transferred.”), FIG. 3.

203. Heil explains that the “host logic's thread is then either put to sleep until it receives a notification event from the acceleration component 2704, or it continues processing other data asynchronously.” AMD1010, [0174]–[0175]. This further clarifies that host logic’s thread is implemented as part of the host component’s central processing units. AMD1010, [0195] (“The host component 3102 can include one or more processing devices 3104, such as...one or more central processing units (CPUs), each of which may implement one or more hardware threads”), [0051]; *see also* AMD1010, [0044], [0048], [0051], [0165], [0195], FIG. 31; *see also* AMD1014, pp.4 (“a thread fills its slot with data”), 9 (“CPU Threads”); AMD1011, pp.2 (“to interface the board to the host CPU over PCIe”), 4 (“The communication interface between the CPU and FPGA must satisfy two key design goals ... [t]o achieve these goals, we developed a custom PCIe interface with DMA support.”), FIG. 3.

7. Claim 8

- (i) *Element [8]: The compute node of claim 7 wherein the data mover component comprises a device controller.*

204. The Heil-Jun Combination renders obvious Claim 8. As I discussed at 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 understood or obvious to a POSITA, PCIe/DMA functionality routinely involves a *device controller* to help manage the transfer process. *Supra* Elements [1e], [7]; AMD1010, [0165], [0173]–[0175], FIG. 27; *see also* AMD1014, p.4 (describing a PCIe interface with DMA support); FIG. 3 (“DMA Engine”). For example, the POSITA would have appreciated that a *device controller* would have been used in Heil's system to receive instructions and notify the “host logic” for receiving data from and providing data to the host component's main memory. AMD1010, [0173]–[0175], FIG. 27. A POSITA would have also understood that PCIe with DMA would have included a “device controller” (e.g., PCIe *Device Controller*, PCIe Engine, DMA Engine) to facilitate data communication with the memory. *See* AMD1039, pp.3–5, FIGS. 3–5 (demonstrating a “*device controller*” with DMA logic for PCIe with DMA). For example, 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 receiving data from and providing data to the host component's main memory. *Id.*

205. As an additional, independent basis to establish obviousness, Jun discloses 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 in order facilitate data transfer from the local host's main memory using PCIe. AMD1011, pp.4–6 (“local host software over PCIe DMA”), 6 (“the DMA engine on the hardware will read data from each buffer in order in a contiguous stream”), 9 (“Reading or writing data from the host buffers were done by DMA read/write engines.”), FIGS. 7, 9–10; *see also*. AMD1001, 15:59–16:10 (“device controller...that is configured and operative to facilitate bi-directional data communications between electronic devices, either independently or in cooperation with control signals or other instructions received from processing system 430 or specific components thereof.”).

8. Claim 9

- (i) *Element [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.*

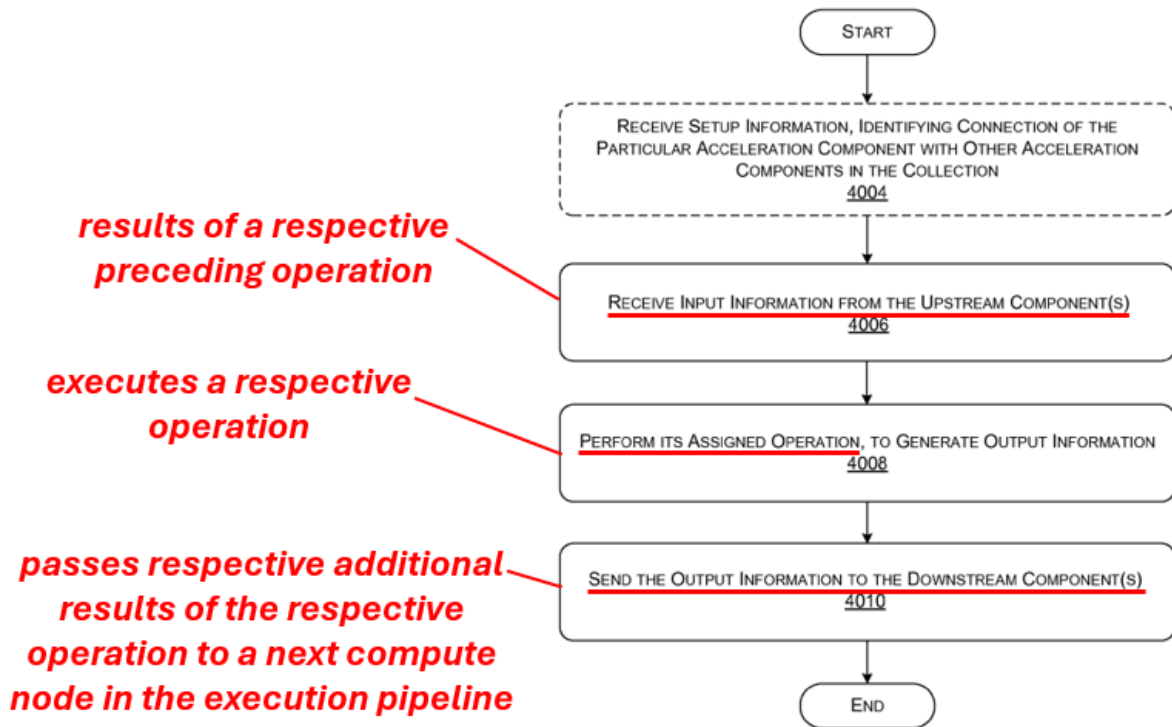
206. The Heil-Jun Combination renders obvious Claim 9. As I discussed for Elements [1pre]–[1a], the server unit component's (*compute node*) transport component (*communications link*) *enables* the server unit component to receive a

request form the host component (*host compute system*) (e.g., via a head component) to invoke the service . AMD1010, [0167]–[0168], [0181], [0218]–[0223], FIGS., FIG. 36; *see also* [0239], FIGS. 36, 40; *supra*, §X.A. In the Figure 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]; *see also* [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.* Thus, the **instructions** included in the request *influences the operation the* server unit component (*compute node*). AMD1010, [0219]–[220], [0229], FIGS. 36, 40.

9. Claim 11

- (i) *Element [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.*

207. The Heil-Jun Combination renders obvious Claim 11. As I discussed for Element [1a], Heil teaches pipelined server unit components (*compute nodes*) and each server unit component (*compute node*) includes an acceleration unit with application logic that executes *a respective operation*. *Supra* Elements [1a]–[1c]; *see also* AMD1010, [0229], FIG. 40. As discussed at Element [1c], each server unit component (*compute node*) includes an acceleration unit with application logic that executes *a respective operation*. In more detail, Heil explains that each “particular acceleration component receives input information from the upstream component(s) [*e.g., the results of a respective preceding operation*]... performs its assigned [*respective*] *operation* to generate output information ... [and] sends the output information to the downstream component(s) [*e.g., passes respective additional results of the respective operation to a next compute node in the execution pipeline*].” AMD1010, [0229], FIG. 40.



AMD1010, FIG. 40 (annotated).

208. As discussed at Element [1h], the operation assigned to the server unit component (*compute node*) is *associated with the computationally expensive algorithm*. See also 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] (“configuration image contains logic...used to implement a

corresponding service.”); AMD1001, 5:10–45, 6:26–34, 19:40–20:15, 35:28–43, 36:45–55, 44:59–45:25, 56:15–17, FIG. 22 (confirming that the *computer-executable code* in the ’636 patent is incorporated in a bit stream for FPGA configuration); *infra* Element [12]; *see also* AMD1010, [0049] (“at least some tasks run as software on FPGA software processor overlays or the like”), [0157], [0161]–[0162]. Heil therefore teaches or renders obvious that each acceleration component, in the *executes a respective operation associated with the computationally expensive algorithm using information in a computer-executable code associated with the computationally expensive algorithm*. AMD1010, [0124], [0155]; *below* Element [12] analysis; *see also* AMD1010, [0049], [0157], [0161]–[0162]; *supra* [1h].

209. I note that a POSITA would have understood or found obvious that an FPGA is configured by executing instructions stored in memory, for example by instructions inserted through the standard JTAG TAP port. The IEEE Standard 1532-2002 defines these instructions to be ISC (in-system configuration) instructions. *See* AMD1027, 47–79.

10. Claim 12

- (ii) *Element [12]: The compute node of claim 11 wherein the computer-executable code associated with the computationally expensive algorithm is incorporated in a bit stream.*

210. The Heil-Jun Combination renders obvious Claim 12. As I discussed for Element [11], Heil discloses *computer-executable code associated with the computationally expensive algorithm* including a configuration stream/image. Heil's configuration stream for an FPGA Device is *incorporated in a bit stream*. AMD1010, [0124], [0155]; *see* AMD1014, pp.2 (“bitstream reconfiguration”), 4 (“reconfiguration function, passing in the desired bitstream”), 5 (“configuring FPGAs with the correct application images”). A POSITA would have understood or found obvious that a configuration *stream*/image in the context of FPGA devices refer to a *bit stream*. *See e.g.*, AMD1015, p.2 (“a binary file called a configuration file or bitstream”); AMD1016, Abstract (“configuration bitstream”); AMD1017 p.3 (“bitstream (configuration image)”).

211. Again, I note that a POSITA would have understood or found obvious that an FPGA is configured by executing instructions stored in memory, for example by instructions inserted through the standard JTAG TAP port. The IEEE Standard 1532-2002 defines these instructions to be ISC (in-system configuration) instructions. *See* AMD1027, 47–79.

XI. GROUND 1A: HEIL, JUN AND RAGHAVA

A. The Heil-Jun-Raghava Combination

212. As an initial matter, Heil, Jun, Raghava and the claims of the '636 patent each belong to the field of endeavor in data processing systems employing reconfigurable hardware. are directed to a technical problem of coupling reconfigurable nodes to perform multi-device processing. AMD1006, Abstract, [0004], [0015]–[0017], [0040]–[0049]; *See supra*, §§IX.A, X.A–C. 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. AMD1006, Abstract, [0004], [0017], [0040]–[0049]; *See supra*, §§IX.A, X.A–C.

1. Incorporation of Raghava's ONFI Protocol with the Heil-Jun Combination

213. As I discussed, the Heil-Jun Combination incorporates Jun's teachings regarding a flash board including NAND-based Flash in conjunction with suitable connectors and hardware interface (*storage interface*) for the pipelined server unit components (*compute nodes*). *Supra*, §§ X.C; X.D, Elements [1b], [1f]–[1g].

214. As an initial matter, I note that a POSITA would have understood or found obvious that Jun would have employed the well-known ONFI protocol to interface with NAND Flash Storage. Indeed, AMD1012 (Liu), as discussed above

in X.D.3, Element [1i], is directed to the same type of flash-based compute nodes disclosed in Jun. *See* AMD1012, pp.3, (“BlueDBM...Big Data flash storage platform), 16–18; *Supra*, §§X.B; X.D, Elements [1b], [1f]–[1g] (where I describe Raghava’s teachings for the ONFI protocol); AMD1006, [0033], [0046]. Liu provides details for an “**ONFI**-compliant FPGA-based flash controller.” that corresponds the controller for Jun’s flash board. AMD1012, pp.3, 31 (“the communication protocol is defined by the Open NAND Flash **Interface** (**ONFI** 2.2) standard), 32–33 (“the **ONFI protocol**”), 37. Liu’s “ONFI-compliant FPGA-based flash controller” corresponds to the flash controller used on Jun’s **storage interface** including utilizing the same storage (NAND Flash) and flash board controller (Atrix-7 FPGA) for the same type of node (BlueDBM). AMD1012, 23-28, FIGS. 2-1, 2-3, 2-4 (“Artix-7 FPGA”) *compare with* AMD1011, pp.4–5, 7–8, FIGS. 8, 10 (“Atrix-7 FPGA”). Accordingly, Liu confirms that a POSITA would have understood that Jun’s flash board would have employed the well-known ONFI protocol **for each** server unit component (**compute node**) in Heil’s **execution pipeline**. *See supra*, §§X.B–C, X.D Elements [1d], [1h]–[1i]; *See also*, AMD1035, [0008] (describing the evolution of the ONFI protocol including that the “1.0 specification” defined “a 50 MT/s transfer rate” which is “a twenty percent improvement over legacy NAND 40 MT/s transfer rate” and that the “ONFI 2.2” (e.g., the Protocol employed by Liu) has “an asynchronous single data rate version was introduced, with a 50 MT/s

maximum transfer speed, while the maximum transfer speed for the synchronous DDR version increased to 200 MT/s.”).

215. Further, a POSITA would have been motivated to implement, at the server unit component (*compute node*), the well-known standard **ONFI** protocol based on the teachings of Raghava. *Supra* §§IX.A–B, IX.C, Element [5].

216. For example and as I discussed in this section above regarding Jun, a POSITA would have found obvious to employ the well-known **ONFI** standard *protocol* as taught by Raghava 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. *See* AMD1012, pp.3, 16–18, 23–28, 31 (“the communication protocol is defined by the Open NAND Flash *Interface* (**ONFI** 2.2) standard), 32–33 (“the **ONFI protocol**”), 37, FIGS. 2-1, 2-3, 2-4; , AMD1035, [0008] (describing the evolution of the ONFI protocol and improvements in transfer speed).

217. Additionally, the combination of the Heil-Jun Combination’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. *See* AMD1012, pp.3, 31–33, 37 (demonstrating an ONFI compliant BlueDBM flash board) . A POSITA

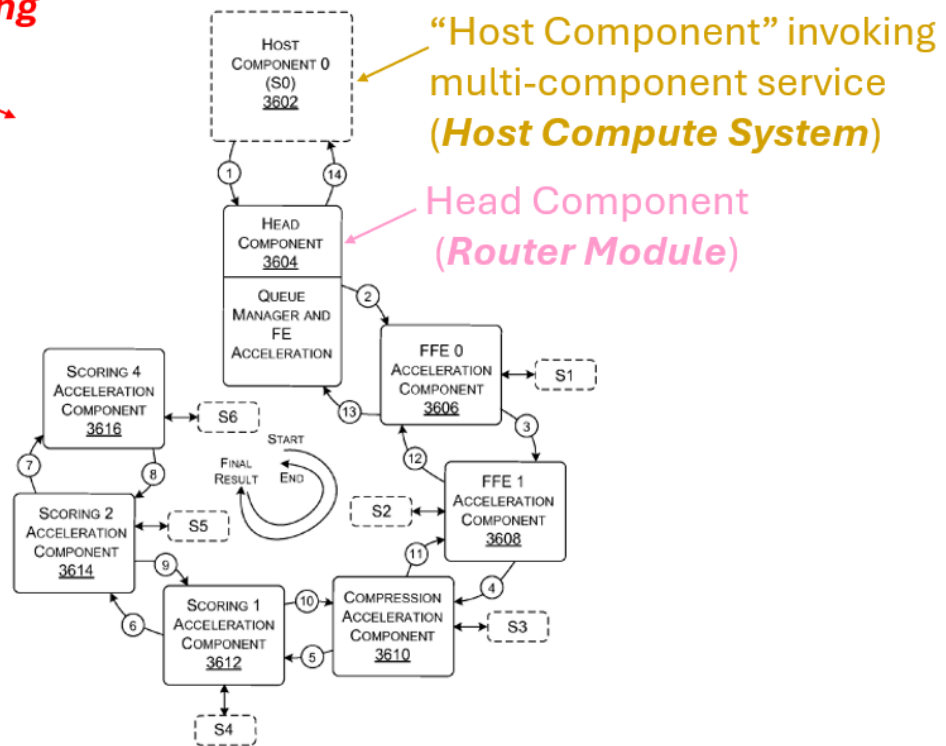
would have an expectation of success for implementing a standard *protocol* like Raghava's *ONFI* on the known components (Jun's *storage interface*) to achieve an expected result of enabling access to NAND Flash. *Id.* For example, a POSITA would have an expectation of success because ONFI is a standard 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.

2. Incorporation of Raghava's Interface Card with the Heil-Jun Combination

218. First, a POSITA would have understood or found obvious 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* §X.D, Elements [1a]–[1pre]; AMD1010, [0068] (“The head component corresponds to a point of contact by which entities in the data processing system 102 may interact with the multi-component service...”); [0219]–[0223], [0236], [0251] (“the requesting host component accesses the collection of hardware acceleration components via a head component.”), [0167]–[0168], [0176]–[0186], FIGS. 25, 27–28, 36, *see also* [0074] (“the head component may then forward plural messages to plural respective acceleration components.”), [0089], [0095] (“the local acceleration

component routes the request to a remote acceleration component. As noted above, the local acceleration component is configured to perform routing to the remote acceleration component without involvement of the local host component.”), [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] (discussing the head component’s “queue manager,” including policies based on a “round-robin basis”), [0252]; AMD1001 9:47–10:24 (enabling “data transmission”), (distributing “data packets and write commands...in accordance with a data-aware round robin strategy”), (“router 122 may execute all routing functionality on...[an] FPGA”); *Supra* §X.D, Elements [1a]–[1pre].

**Data Processing
System**



AMD1010, FIG. 36 (annotated).

219. As I show in the Figure 36 example (above), the head component 3604 (*router module*) is *communicatively coupled* to the “host component 3602” (*host compute system*) and communicates with the host component to receive the request from the “host component 3602...in operation (1)” and *route* the final score “to the host component 3602” in operation (14). AMD1010, [0218]–[0223].

220. Next, Heil discloses a host component and remote acceleration components being “implemented on a single physical component.” AMD1010, [0088], *see* [0084]–[0090] (“implemented on a single die”). Consistent with this disclosure, a POSITA would have been motivated to implement 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*, §§X.A, X.D , Elements [1a]–[1b]. In fact, Raghava demonstrates that integrating this arrangement into a single accelerator DIMM card (*interface card*) was well known. *See* AMD1006, [0040]–[0045], FIG. 5; *supra*, §§IX.1. In more detail, 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] ("receiving a read request...[and] route the read request...[to] one of HW acceleration unit 508."), FIG. 5. The '636 patent also acknowledges that this "particular structural arrangement...[is] generally known in the art." AMD1001, 14:56–15:2; 11:46–61 ("router module 120...[and] compute nodes 140...may be incorporated into [an]...interface card...as is generally known.").

221. 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 (1) 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 and (2) Raghava's teachings for a router and acceleration unit

arrangement on an interface card coupled to the host. The following reasons, taken separately or collectively, support such a Heil-Jun-Raghava Combination.

222. 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. AMD1006, Abstract, [0004] (“offloading computationally intensive tasks...to highly-parallel acceleration resources on a programmable device”), [0015]; [0017]; [0040]–[0044], [0048]–[0049]; AMD1010, [0051], [0224] (“leverage parallel computational resources”). For example, Raghava explains that “offloading computationally intensive functions onto the accelerator DIMM card is oftentimes more efficient than performing them natively on the host CPU due to the highly parallel nature of the programmable device.” AMD1006, [0015], *see also* AMD1006, Abstract, [0004], [0017]; [0040]–[0044], [0048]–[0049], [0056]. Similarly, Heil explains that “an acceleration component may perform operations...by using a large number of parallel logic elements to perform computational tasks...[and thus]...can perform some operations in less time compared to a software-driven host component.” AMD1010, [0051], *see also* AMD1010, [0224]. Heil explains that in this context “the ‘acceleration’ qualifier associated with the term “acceleration component” reflects its potential for

accelerating the functions that are performed by the host components.” AMD1010, [0051], *see also* AMD1010, [0224].

223. 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. 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/transport delay. *Id.*; *see* AMD1006, [0036]–[0037], [0053]–[0056] (“the DIMM card transport delay T_z is substantially less the PCIe transport delay”). Raghava explains that “the acceleration circuitry is formed as part of the DIMM card to take advantage of the low latency path connecting the host CPU to the DIMM card.” AMD1006, [0036]–[0037].

224. Third, a POSITA would have been motivated to implement Heil’s head component (***router module***) and server unit component (***compute node***) based on Raghava’s teachings to do so in order to improve the energy efficiency for executing services. For example, Raghava’s teaches that “the inclusion of acceleration circuitry in the DIMM card/slot improves the energy efficiency of applications.” AMD1006, [0053], *see also* AMD1006, [0035]–[0037] (“results in significant improvements in terms of energy spent per unit of computation”).

225. Fourth, combining Heil-Jun Combination'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. 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. *See* AMD1014, 2(explaining that "one option is to incorporate multiple FPGAs onto a daughtercard and house such a card along with a subset of the servers" and that this option is "straightforward to implement" while providing advantages (e.g., ability to "achieve the required capacity" and disadvantages of such an implementation).

B. Claim Element Analysis

1. Claim 5

- (i) *Element [5]: The compute node of claim 4 wherein the storage interface employs an Open NAND Flash Interface (ONFI) protocol.*

226. The Heil-Jun-Raghava combination from §XI.A.1 renders obvious Claim 5 at least on the basis of incorporating Raghava's teachings for employing the **ONFI Protocol** in the *storage interface component at* the server unit component (*compute node*), as I described above. *See Supra* §XI.A.1; *see also* §§IX.A–B; IX.C, Elements [4]–[5].

2. Claim 10

- (i) *Element [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.*

227. The Heil-Jun-Raghava Combination from §XI.A.2 renders obvious Claim 10 at least on the basis of incorporating Raghava's teachings for the *interface card* as I described above. *See supra* §XI.A.2.

XII. CONCLUSION

228. I declare that all statements made in this Declaration of my own knowledge are true and that all statements made on information and belief are believed to be true. I further declare that these statements are made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both (under Section 1001 of Title 18 of the United States Code).

Executed this 11th day of May, 2026.



Donald Alpert, PhD

APPENDIX A



Donald Alpert, Ph.D.
Camelback Computer Architecture, LLC
2310 Homestead Road
Suite C1 #411
Los Altos CA 94024
(650) 386-5301

Donald Alpert, Ph.D., is an independent consultant, specializing in microprocessors and computer systems. For more than a decade in his role as president of Camelback Computer Architecture LLC, Dr. Alpert has provided clients with expert services for patent litigation, technology assessment, and product development. He has over 40 years of wide-ranging academic and industry experience that enable him to provide expert consulting services in the following areas:

- Computers, Computer Systems
- Computer architecture
- Microprocessors
- Semiconductors
- PC Systems
- Embedded Systems
- Design Validation and Debug
- Performance Evaluation
- Cache Memory
- Instruction Sets
- Media Processing
- Power Management

Patent Analysis: Dr. Alpert has served as an expert on dozens of patent litigation cases, handling both infringement and invalidity issues. Dr. Alpert has worked as both a consulting expert and testifying expert assisting attorneys representing major high-tech corporations. He has substantial experience testifying in depositions and at trials before U.S. Federal Courts and the International Trade Commission. Dr. Alpert also consults to assess patents for acquisition and for pre-litigation review. See [here](#) for a summary of testifying experience.

Product Development: Dr. Alpert is a widely recognized innovator with over 30 patents. Dr. Alpert has worked at some of the most ground-breaking technology firms, including Zilog, National Semiconductor, and Intel, where he was the principal architect of the Pentium® Processor. See [here](#) for more information and example projects.

Employment Experience

Camelback Computer Architecture, LLC, Sacramento, CA

Consultant

2/02-Present

Computer architecture consulting services to assess technology, develop products, and protect IP.

Intel Corporation, Chandler, Arizona

Principal Engineer

12/99-1/02

Led architecture team of 10 responsible for definition and performance evaluation of Itanium™ Processor Family microprocessors.

Principal Engineer

5/98-12/99

Led architecture team of 4 responsible for technical strategy and product definition of PC graphics processors and chipsets.

Stanford University, Stanford, California

Consulting Assistant Professor

9/97-4/98

Responsible for teaching undergraduate and graduate classes in computer architecture and participating in processor-design research projects.

Intel Corporation, Santa Clara, California

Engineering Manager

10/93-6/97

Responsible for microarchitecture definition and architecture validation of the Itanium™ microprocessor, which was the first implementation of Intel's 64-bit extension architecture. Led a team of 60 engineers.

Architecture Manager

1/89-10/93

Responsible for the Pentium® Processor technical definition. Worked with customers and Intel marketing to identify product requirements and with the design engineering team to evaluate design options. Led a team of seven engineers responsible for creating technical specifications and for modeling and simulating performance using traces from DOS, Windows, and UNIX execution. After completing the Pentium Processor, led an interdisciplinary team of 30 engineers to define a new 64-bit instruction set architecture.

National Semiconductor Corporation, Herzlia, Israel

Architecture Manager

5/85-1/89

Responsible for defining new microprocessors in the Series 32000 family, including the NS32532 and "Swordfish" high-performance microprocessors. Led a team of five engineers working in the following areas: modeling and simulating high-performance microprocessors, evaluating novel computer architectures, and writing technical specifications for new products.

Zilog, Incorporated, Campbell, California

Component Architect

6/80-4/85

Worked on the Z80,000 CPU, a 32-bit microprocessor that integrated an instruction pipeline, cache, and memory management unit. Responsible for specifying the non-privileged instruction set, evaluating performance, and writing the reference manual for the microprocessor.

Burroughs Corporation, Pasadena, California

Associate Engineer

7/76-7/77

Designed computer peripheral interface controllers using MSI-TTL and 8080 microprocessors.

Alpex Computer Corporation, Danbury, Connecticut

8/75-6/76

Consulting engineer working on design of video games.

Action, Washington, DC

6/73-6/75

Peace Corps volunteer teaching mathematics and physics in a secondary school in Ghana, West Africa.

Donald Alpert Camelback Computer Architecture, LLC (650) 386-5301

Education

Stanford University 9/77-6/84
Graduate study in computer design. National Science Foundation Graduate Fellowship.
Awarded MSEE in 1978 and PhDEE in 1984.

M.I.T. 9/69-6/73
Member of Tau Beta Pi and Eta Kappa Nu. Awarded BSEE in 1973.

Professional Activities

Member of Microprocessor Report editorial board (2002-present)

Anthem College Commencement Address (18-Dec-2009)

Panel session about history of microprocessor development at National Semiconductor Corporation, Computer History Museum (2008).

[\[http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf\]](http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf)

IEEE Technical Committee on Microprocessors and Microcomputers, Chair (1999-2000) and Vice-Chair (2001-present).

Lectures about the Itanium™ Processor Family at U.C. Irvine (1999, 2000).

Cool Chips I, Keynote speaker (Tokyo, 1998)

IEEE Hot Chips Symposium Program Co-chair (1994), member of program committee (1993, 1995).

Lectures about the Pentium® Processor at Stanford University, University of Illinois, and Cornell (1993-1994).

Appeared in Stanford UVC Educational Video production about the Pentium Processor (1993).

IEEE International Symposium on Computer Architecture member of program committee (1996) and Session Chair for Cache Design (1989).

Teaching Experience

Faculty Associate at Arizona State University teaching undergraduate class on assembly language programming (Spring/2001) and computer architecture (Spring/2002).

Consulting Assistant Professor at Stanford University teaching undergraduate and graduate classes in computer architecture (Fall/1997, Winter/1998).

Consulting Professor at Tel Aviv University teaching graduate class in computer architecture (Spring/1987)

Peace Corps volunteer in a secondary school in Ghana, West Africa teaching mathematics and physics (1973-1975).

Don Alpert, "How Microprocessors Upset the Computer Industry," Microprocessor Report, December 11, 2006.

Don Alpert, "Evolving PC Chip-Set Topologies," Microprocessor Report, April 17, 2006.

Don Alpert, "Will Microprocessors Become Simpler?" Microprocessor Report, November 17, 2003.

Don Alpert, "Itanium Processor Status Report," Microprocessor Report, July 28, 2003.

Don Alpert, "Scalable MicroSupercomputers," Microprocessor Report, March 17, 2003.

Don Alpert, "ColdFire Goes Fully Superscalar," Microprocessor Report, October 28, 2002.

Donald Alpert and Alan Jay Smith, "Guest Editor Introduction," IEEE Micro special issue on Hot Chips VI, April 1995.

Donald Alpert and Dror Avnon, "Architecture of the Pentium Processor," IEEE Micro, June 1993, pp. 11-21. Selected best paper in IEEE Micro for 1993.

D. Alpert, A. Averbuch, and O. Danielli, "Performance Comparison of Load/Store and Symmetric Instruction Set Architectures," Proceedings, 18th International Symposium on Computer Architecture, June 1990, pp. 172-181.

Benjamin Maytal, Sorin Iacobovici, Donald Alpert, et al, "Design Considerations for a General-Purpose Microprocessor," Computer, January 1989, pp. 66-76.

Donald Alpert and Michael Flynn, "Performance Trade-offs for Microprocessor Cache Memories," IEEE Micro, August 1988, pp. 44-54.

D. Alpert, J. Levy, and B. Maytal, "Architecture of the NS32532 Microprocessor," Proceedings, 1987 IEEE International Conference on Computer Design, October 1987, pp. 168-172. Awarded best presenter at Design and Test sessions.

D. Alpert, D. Biran, L. Epstein, et al, "Trends in VLSI Microprocessor Design," Proceedings, First Annual Conference on Computer Technology, Systems and Applications (CompEuro '87), May 1987, pp. 564-567.

Y Sidi, D. Alpert, D. Biran, et al, "Design Considerations of an Advanced 32-Bit Microprocessor," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

D. Alpert, "Trends in VLSI Microprocessor Design," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

Donald Alpert, Michael J. Flynn, and Scott Wakefield, "Directly Executed Languages for VLSI Processor Design," Proceedings, 1983 IEEE International Conference on Computer Design, October 1983, pp. 609-612.

Donald Alpert, "Powerful 32-Bit Micro Includes Memory Management," Computer Design, October 1983, pp. 213-220.

Don Alpert, Dean Carberry, Mike Yamamura, et al, "32-Bit Processor Chip Integrates Major System Functions," Electronics, July 14, 1983, pp. 113-119.

Patents

4,802,085, Apparatus and method for detecting and handling memory-mapped I/O by a pipelined microprocessor
5,249,286, Selectively locking memory locations within a microprocessor's on-chip cache
5,263,153, Monitoring control flow in a microprocessor
5,416,913, Method and apparatus for dependency checking in a multi-pipelined microprocessor
5,442,756, Branch prediction and resolution apparatus for a superscalar computer processor
5,475,824, Microprocessor with apparatus for parallel execution of instructions
5,479,652, Microprocessor with an external command mode for diagnosis and debugging
5,481,751, Apparatus and method for storing partially-decoded instructions in the instruction cache of a CPU having multiple execution units
5,559,986, Interleaved cache for multiple accesses per clock cycle in a microprocessor
5,606,676, Branch prediction and resolution apparatus for a superscalar computer processor
5,617,554, Physical address size selection and page size selection in an address translator
5,621,886, Method and apparatus for providing efficient software debugging
5,638,525, Processor capable of executing programs that contain RISC and CISC instructions
5,657,253, Apparatus for monitoring the performance of a microprocessor
5,659,679, Method and apparatus for providing breakpoints on taken jumps and for providing software profiling in a computer system
5,669,011, Partially decoded instruction cache
5,671,435, Technique for software to identify features implemented in a processor
5,675,825, Apparatus and method for identifying a computer microprocessor
5,692,167, Method for verifying the correct processing of pipelined instructions including branch instructions and self-modifying code in a microprocessor
5,729,724, Adaptive 128-bit floating point load and store operations for quadruple precision compatibility
5,740,413, Method and apparatus for providing address breakpoints, branch breakpoints, and single stepping
5,764,959, Adaptive 128-bit floating point load and store instructions for quad-precision compatibility
5,774,686, Method and apparatus for providing two system architectures in a processor
5,790,834, Apparatus and method using an ID instruction to identify a computer microprocessor
5,802,605, Physical address selection and page size selection in an address translator
5,958,037, Apparatus and method for identifying features and the origin of a computer microprocessor
5,991,874, Conditional move using a compare instruction generating a condition field
6,052,801, Method and apparatus for providing breakpoints on a selectable address range
6,219,774, Address translation with/bypassing intermediate segmentation translation to accommodate two different instruction set architecture
6,385,718, Computer system and method for executing interrupt instructions in operating modes
6,408,386, Method and apparatus for providing event handling functionality in a computer system
6,584,558, Article for providing event handling functionality in a processor supporting different instruction sets
7,010,671, Computer system and method for executing interrupt instructions in two operating modes
7,389,403, Adaptive computing ensemble microprocessor architecture
7,802,073, Virtual core management