(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 December 2022 (22.12.2022)

WIPO | PCT

(10) International Publication Number
WO 2022/263277 A1

(51) International Patent Classification:
G06F 9/30 (2018.01) *G06N 3/063* (2006.01)

(21) International Application Number:
PCT/EP2022/065660

(22) International Filing Date:
09 June 2022 (09.06.2022)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
17/350,550 17 June 2021 (17.06.2021) US

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, New York 10504 (US).

(71) Applicant *(for MG only)*: **IBM DEUTSCHLAND GMBH** [DE/DE]; IBM-Allee 1, 71139 Ehningen (DE).

(72) Inventors: **LICHTENAU, Cedric**; c/o IBM Deutschland Research & Development GmbH, Schoenaicher Strasse 220, 71032 Boeblingen (DE). **BRADBURY, Jonathan**; c/o IBM Corp., 2455 South Road, Poughkeepsie, New York 12601 (US). **ALBARAKAT, Laith**; c/o IBM Corp., 2455 South Road, Poughkeepsie, New York 12601 (US).

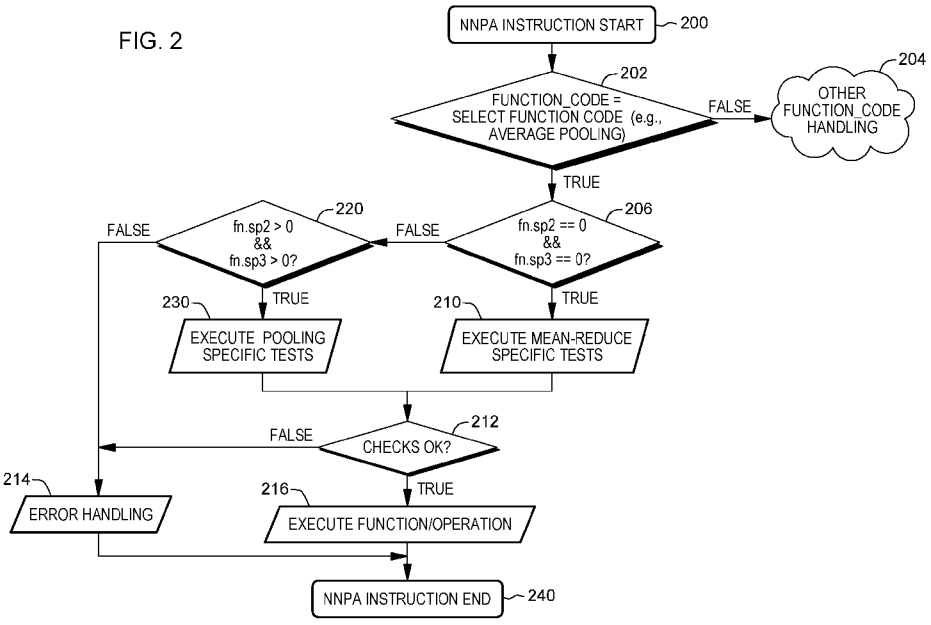
(74) Agent: **VETTER, Svenja**; c/o IBM Deutschland GmbH, Patentwesen und Urheberrecht, IBM-Allee 1, 71139 Ehningen (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM,

WO 2022/263277 A1

(54) Title: SINGLE FUNCTION TO PERFORM MULTIPLE OPERATIONS WITH DISTINCT OPERATION PARAMETER VALIDATION

FIG. 2



(57) Abstract: An indication of a function to be executed is obtained, in which the function is one function of an instruction and configured to perform multiple operations. A determination is made of an operation of the multiple operations to be performed, and a set of function-specific parameters is validated using a set of values and a corresponding set of relationships. The set of values and corresponding set of relationships are based on the operation to be performed. One set of values and corresponding set of relationships are to be used for the operation to be performed, and another set of values and corresponding set of relationships are to be used for another operation of the multiple operations.

[Continued on next page]

TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

SINGLE FUNCTION TO PERFORM MULTIPLE OPERATIONS WITH DISTINCT OPERATION PARAMETER VALIDATION

BACKGROUND

[0001] One or more aspects relate, in general, to facilitating processing within a computing environment, and in particular, to improving such processing.

[0002] In order to enhance processing in computing environments that are data and/or computational-intensive, co-processors are utilized, such as artificial intelligence accelerators (also referred to as neural network processors or neural network accelerators). Such accelerators provide a great deal of compute power used in performing, for instance, involved computations, such as computations on matrices or tensors.

[0003] Tensor computations, as an example, are used in complex processing, including deep learning, which is a subset of machine learning. Deep learning or machine learning, an aspect of artificial intelligence, is used in various technologies, including but not limited to, engineering, manufacturing, medical technologies, automotive technologies, computer processing, etc.

[0004] Deep learning uses various operations that operate on tensor data. Each of the operations is independently implemented, increasing development and verification efforts.

SUMMARY

[0005] Shortcomings of the prior art are overcome, and additional advantages are provided through the provision of a computer program product for facilitating processing within a computing environment. The computer program product includes one or more computer readable storage media and program instructions collectively stored on the one or more computer readable storage media to perform a method. The method includes obtaining an indication of a function to be executed, in which the function is one function of an instruction and configured to perform multiple operations. A determination is made of an operation of the multiple operations to be performed, and a set of function-specific parameters is validated using a set of values and a corresponding set of relationships. The set of values and corresponding set of relationships are based on the operation to be performed. As examples, one set of values and corresponding set of relationships are to be

used for the operation to be performed, and another set of values and corresponding set of relationships are to be used for another operation of the multiple operations.

[0006] Using a single function, e.g., a single function of an architected instruction, to perform multiple operations but with per operation parameter validation, code complexity, code duplication and/or verification efforts are reduced, improving system performance.

[0007] In one example, the determining the operation to be performed includes checking one or more function-specific parameters against at least one specific value. Based on the one or more function-specific parameters having a first select relationship with respect to the at least one specific value, the operation is one operation, and based on the one or more function-specific parameters having a second select relationship with respect to the at least one specific value, the operation is another operation.

[0008] By using the same function-specific parameters but different relationships to determine the operation to be performed, code complexity and verification efforts are reduced.

[0009] As an example, the function includes an average pool function, the one or more function-specific parameters include one or more stride values, in which a stride value is an amount that a sliding window moves over an input tensor when computing one or more adjacent output tensor elements, the at least one specific value includes zero, the first select relationship includes equal and the operation is a mean-reduce operation based on the one or more stride values being equal to zero.

[0010] Further, in one example, the second select relationship includes greater than and the operation is a pooling operation based on the one or more stride values being greater than zero.

[0011] As an example, the set of function-specific parameters includes one or more select-dimension window size values. A select-dimension window size value specifies a number of elements in the select dimension that a sliding window contains, and the sliding window is configured to move over an input tensor of the function to produce an output tensor.

[0012] In one example, the function includes an average pool function, the operation includes a mean-reduce operation, and the set of values and corresponding set of

relationships to be used to validate the set of function-specific parameters includes: one value of one dimension of a select input tensor and corresponding relationship of equal, another value of another dimension of the select input tensor and corresponding relationship of equal, and a select value and corresponding relationship of less than or equal.

[0013] The validating includes, for instance, checking a value of a dimension-2-window size is equal to a value of a dimension-2 of a first input tensor, a value of a dimension-3-window size is equal to a value of a dimension-3 of the first input tensor, the value of the dimension-2-window size is less than or equal to the select value, and the value of the dimension-3-window size is less than or equal to the select value.

[0014] In one example, the function includes an average pool function, the operation includes a pooling operation, and the set of values and corresponding set of relationships to be used to validate the set of function-specific parameters includes: one value of one dimension of a select input tensor and corresponding relationship of less than or equal, and another value of another dimension of the select input tensor and corresponding relationship of less than or equal.

[0015] The validating includes, for instance, checking that a value of a dimension-2-window size is less than or equal to a value of a dimension-2 of a first input tensor and a value of a dimension-3-window size is less than or equal to a value of a dimension-3 of the first input tensor.

[0016] In one example, a determination is made as to whether a type of padding is set to a particular type, in which the type of padding indicates which elements of a window are to be used in computing the output and, for one or more embodiments, based on the type of padding being set to the particular type, the validating is performed. Further, in one example, based on the type of padding not being set to the particular type, one or more checks are performed relating to one or more dimensions of an output tensor.

[0017] In one example, the determining the operation is based on at least one sliding window stride value of an input tensor, and the set of function-specific parameters includes at least one sliding window dimension of an input tensor.

[0018] Computer-implemented methods and systems relating to one or more aspects are also described and claimed herein. Further, services relating to one or more aspects are also described and may be claimed herein.

[0019] Additional features and advantages are realized through the techniques described herein. Other embodiments and aspects are described in detail herein and are considered a part of the claimed aspects.

BRIEF DESCRIPTION OF THE DRAWINGS

[0020] One or more aspects are particularly pointed out and distinctly claimed as examples in the claims at the conclusion of the specification. The foregoing and objects, features, and advantages of one or more aspects are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

FIG. 1A depicts one example of a computing environment to incorporate and use one or more aspects of the present invention;

FIG. 1B depicts further details of a processor of FIG. 1A, in accordance with one or more aspects of the present invention;

FIG. 2 depicts one example of processing associated with executing a single function of an instruction that is configured to perform multiple operations but able to check distinct parameter conditions for the multiple operations, in accordance with one or more aspects of the present invention;

FIG. 3A depicts one example of a format of a Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 3B depicts one example of a general register used by the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 3C depicts examples of function codes supported by the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 3D depicts one example of another general register used by the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 3E depicts one example of a parameter block used by a query function of the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 3F depicts one example of a parameter block used by one or more non-query functions of the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 3G depicts one example of a tensor descriptor used by the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIG. 4 depicts one example of a format of a Neural Network Processing (NNP)-data-type-1 data type, in accordance with one or more aspects of the present invention;

FIGS. 5A-5C depict examples of an input data layout used by the Neural Network Processing Assist instruction, in accordance with one or more aspects of the present invention;

FIGS. 6A-6C depict example output corresponding to the input data layout of FIGS. 5A-5C, in accordance with one or more aspects of the present invention;

FIGS. 7A-7C depict one example of facilitating processing within a computing environment, in accordance with one or more aspects of the present invention;

FIG. 8A depicts another example of a computing environment to incorporate and use one or more aspects of the present invention;

FIG. 8B depicts one example of further details of a memory of FIG. 8A, in accordance with one or more aspects of the present invention;

FIG. 8C depicts another example of further details of a memory of FIG. 8A, in accordance with one or more aspects of the present invention;

FIG. 9A depicts yet another example of a computing environment to incorporate and use one or more aspects of the present invention;

FIG. 9B depicts further details of the memory of FIG. 9A, in accordance with one or more aspects of the present invention;

FIG. 10 depicts one embodiment of a cloud computing environment, in accordance with one or more aspects of the present invention; and

FIG. 11 depicts one example of abstraction model layers, in accordance with one or more aspects of the present invention.

DETAILED DESCRIPTION

[0021] In accordance with one or more aspects of the present invention, a capability is provided to facilitate processing within a computing environment. As an example, an instruction is provided that is configured to implement multiple functions, and at least one function is configured to perform multiple operations with distinct parameter validation per operation. By using one function to perform the multiple operations but able to check parameter boundary conditions that differ between the multiple operations, code complexity, code duplication and/or verification efforts are reduced.

[0022] As an example, a function configured to implement multiple operations is an average pooling function and the multiple operations include a mean-reduce operation and a pooling operation used, for instance, in deep learning. The average pooling function performs different operations but is algorithmically reduced to a common algorithmic function using the same input tensors and function-specific-parameters but with different relative constraints.

[0023] In one example, the function configured to perform the multiple operations is initiated by an instruction. As an example, the instruction is a Neural Network Processing Assist instruction, which is a single instruction (e.g., a single architected hardware machine instruction at the hardware/software interface) configured to perform multiple functions. Each of the functions is configured as part of the single instruction (e.g., the single

architected instruction), reducing use of system resources and complexity, and improving system performance. Further, at least one of the functions, the AVGPOOL2D function, an example of which is described below, is configured to implement multiple operations (e.g., mean-reduce and pooling) based on input data, such as values of function-specific-parameters (e.g., function-specific-parameters 2 and 3, described below) provided by the instruction.

[0024] The instruction may be part of a general-purpose processor instruction set architecture (ISA), which is dispatched by a program on a processor, such as a general-purpose processor. It may be executed by the general-purpose processor and/or one or more functions of the instruction may be executed by a special-purpose processor, such as a co-processor configured for certain functions, that is coupled to or part of the general-purpose processor. Other variations are also possible.

[0025] One embodiment of a computing environment to incorporate and use one or more aspects of the present invention is described with reference to FIG. 1A. As an example, the computing environment is based on the z/Architecture® instruction set architecture, offered by International Business Machines Corporation, Armonk, New York. One embodiment of the z/Architecture instruction set architecture is described in a publication entitled, “z/Architecture Principles of Operation," IBM Publication No. SA22-7832-12, Thirteenth Edition, September 2019, which is hereby incorporated herein by reference in its entirety. The z/Architecture instruction set architecture, however, is only one example architecture; other architectures and/or other types of computing environments of International Business Machines Corporation and/or of other entities may include and/or use one or more aspects of the present invention. z/Architecture and IBM are trademarks or registered trademarks of International Business Machines Corporation in at least one jurisdiction.

[0026] Referring to FIG. 1A, a computing environment 100 includes, for instance, a computer system 102 shown, e.g., in the form of a general-purpose computing device. Computer system 102 may include, but is not limited to, one or more general-purpose processors or processing units 104 (e.g., central processing units (CPUs)), at least one special-purpose processor, such as a neural network processor 105, a memory 106 (a.k.a., system memory, main memory, main storage, central storage or storage, as examples), and one or more input/output (I/O) interfaces 108, coupled to one another via one or more buses

and/or other connections. For instance, processors 104, 105 and memory 106 are coupled to I/O interfaces 108 via one or more buses 110, and processors 104, 105 are coupled to one another via one or more buses 111.

[0027] Bus 111 is, for instance, a memory or cache coherence bus, and bus 110 represents, e.g., one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include the Industry Standard Architecture (ISA), the Micro Channel Architecture (MCA), the Enhanced ISA (EISA), the Video Electronics Standards Association (VESA) local bus, and the Peripheral Component Interconnect (PCI).

[0028] As examples, one or more special-purpose processors (e.g., neural network processors) may be separate from but coupled to one or more general-purpose processors and/or may be embedded within one or more general-purpose processors. Many variations are possible.

[0029] Memory 106 may include, for instance, a cache 112, such as a shared cache, which may be coupled to local caches 114 of processors 104 and/or to neural network processor 105, via, e.g., one or more buses 111. Further, memory 106 may include one or more programs or applications 116 and at least one operating system 118. An example operating system includes a z/OS® operating system, offered by International Business Machines Corporation, Armonk, New York. z/OS is a trademark or registered trademark of International Business Machines Corporation in at least one jurisdiction. Other operating systems offered by International Business Machines Corporation and/or other entities may also be used. Memory 106 may also include one or more computer readable program instructions 120, which may be configured to carry out functions of embodiments of aspects of the invention.

[0030] Moreover, in one or more embodiments, memory 106 includes processor firmware 122. Processor firmware includes, e.g., the microcode or millicode of a processor. It includes, for instance, the hardware-level instructions and/or data structures used in implementation of higher level machine code. In one embodiment, it includes, for instance, proprietary code that is typically delivered as microcode or millicode that includes trusted software, microcode or millicode specific to the underlying hardware and controls operating system access to the system hardware.

[0031] Computer system 102 may communicate via, e.g., I/O interfaces 108 with one or more external devices 130, such as a user terminal, a tape drive, a pointing device, a display, and one or more data storage devices 134, etc. A data storage device 134 may store one or more programs 136, one or more computer readable program instructions 138, and/or data, etc. The computer readable program instructions may be configured to carry out functions of embodiments of aspects of the invention.

[0032] Computer system 102 may also communicate via, e.g., I/O interfaces 108 with network interface 132, which enables computer system 102 to communicate with one or more networks, such as a local area network (LAN), a general wide area network (WAN), and/or a public network (e.g., the Internet), providing communication with other computing devices or systems.

[0033] Computer system 102 may include and/or be coupled to removable/non-removable, volatile/non-volatile computer system storage media. For example, it may include and/or be coupled to a non-removable, non-volatile magnetic media (typically called a "hard drive"), a magnetic disk drive for reading from and writing to a removable, non-volatile magnetic disk (e.g., a "floppy disk"), and/or an optical disk drive for reading from or writing to a removable, non-volatile optical disk, such as a CD-ROM, DVD-ROM or other optical media. It should be understood that other hardware and/or software components could be used in conjunction with computer system 102. Examples, include, but are not limited to: microcode or millicode, device drivers, redundant processing units, external disk drive arrays, RAID systems, tape drives, and data archival storage systems, etc.

[0034] Computer system 102 may be operational with numerous other general-purpose or special-purpose computing system environments or configurations. Examples of well-known computing systems, environments, and/or configurations that may be suitable for use with computer system 102 include, but are not limited to, personal computer (PC) systems, server computer systems, thin clients, thick clients, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputer systems, mainframe computer systems, and distributed cloud computing environments that include any of the above systems or devices, and the like.

[0035] In one example, a processor (e.g., processor 104 and/or processor 105) includes a plurality of functional components (or a subset thereof) used to execute instructions. As depicted in FIG. 1B, these functional components include, for instance, an instruction fetch component 150 to fetch instructions to be executed; an instruction decode unit 152 to decode the fetched instructions and to obtain operands of the decoded instructions; one or more instruction execute components 154 to execute the decoded instructions; a memory access component 156 to access memory for instruction execution, if necessary; and a write back component 158 to provide the results of the executed instructions. One or more of the components may access and/or use one or more registers 160 in instruction processing. Further, one or more of the components may, in accordance with one or more aspects of the present invention, include at least a portion of or have access to one or more other components used in performing multiple operations with distinct parameter checking based on executing a single function, and/or in performing neural network processing assist processing of, e.g., a Neural Network Processing Assist instruction (or other processing that may use one or more aspects of the present invention), as described herein. The one or more other components may include, for instance, a single function, multiple operations – distinct parameter validation component 170 and/or a neural network processing assist component 172 (and/or one or more other components).

[0036] In accordance with one or more aspects of the present invention, an instruction is executed which is able to perform multiple functions, and at least one function implements multiple operations with distinct parameter validation. An example of this processing is further described with reference to FIG. 2.

[0037] Referring to FIG. 2, in one example, an instruction, such as a Neural Network Processing Assist (NNPA) instruction (or another instruction), is initiated 200 on a processor, such as general-purpose processor 104. A determination is made as to the function to be performed. This is determined, for instance, by checking the function code of the instruction. If the function to be performed is not a select function code, such as, for instance, a function code specifying an average pool function (e.g., NNPA_AVGPOOL2D), then other processing is performed 204.

[0038] Returning to inquiry 202, if, however, the function code specifies the select function code, e.g., the function code specifying the average pool function, then processing continues, as described herein. In one example, the processing is performed by the general-

purpose processor initiating the instruction. However, in other embodiments, the processing may be performed by a special-purpose processor, such as a neural network accelerator (e.g., neural network accelerator 105) or by another general-purpose processor, special-purpose processor or other processor. Other variations are also possible.

[0039] In one example, based on specifying the AVGPOOL2D function, an input tensor, e.g., input tensor 1, is reduced by the specified operation to summarize windows of the input. A window is, for instance, a select portion of an input tensor having a defined size. The windows of the input are selected by moving a 2D sliding window over, e.g., dimensions 2 and 3 of the input tensor. A summary of the window is an element in the output tensor. The sliding window dimensions are described by, for instance, function-specific-parameters provided by the instruction, e.g., function-specific-parameter 4 and function-specific-parameter 5, examples of which are described herein.

[0040] In processing the function, in one embodiment, a determination is made of the operation to be performed, since the function is configured to perform multiple operations. In one example, the determination of the operation to be performed is made by checking select function-specific-parameters that are provided as an input to the instruction (e.g., in a parameter block used with the instruction). As an example, the select function-specific-parameters are function-specific-parameter 2 (a.k.a., fn.sp2) and function-specific-parameter 3 (a.k.a., fn.sp3), each of which specifies, for instance, a sliding window stride. The sliding window stride or stride is an amount that the sliding window moves over input tensor 1 when computing adjacent output tensor elements.

[0041] In one example, a determination is made as to whether values of function-specific-parameter 2 and function-specific-parameter 3 are equal to a select value, such as zero (206). If, for instance, values of the dimension-2-stride specified by fn.sp2 and the dimension-3-stride specified by fn.sp3 are equal to the select value (e.g., zero), then a mean-reduce operation is to be performed, and thus, mean-reduce specific tests are executed 210. These tests include, for instance, checking whether:

[0042] A padding type is equal to a select padding type, such a Valid. For instance, a check is made of a value of a particular function-specific-parameter, e.g., function-specific-parameter 1 (a.k.a., fn.sp1), provided as input to the function. If the specified padding type is Valid, all elements in the window are added to the collection used to compute the resulting output element.

[0043] A value of function-specific-parameter 4 (fn.sp4) is equal to a value of dimension-2 (e.g., e2) of a first input tensor (e.g., check fn.sp4==in1.e2). For instance, a dimension-2 window size (a.k.a., a sliding window value) in fn.sp4 is checked against a value of in1.e2.

[0044] A value of function-specific-parameter 5 (fn.sp5) is equal to a value of dimension-3 (e.g., e3) of a first input tensor (e.g., check fn.sp5==in1.e3). For instance, a dimension-3 window size (a.k.a., a sliding window value) in fn.sp5 is checked against a value of in1.e3.

[0045] A value of function-specific-parameter 4 is less than or equal to a select value (e.g., 1024) (e.g., fn.sp4 <=1024). For instance, a dimension-2 window size is compared to a select value, e.g., 1024.

[0046] A value of function-specific-parameter 5 is less than or equal to the select value (e.g., 1024) (e.g., fn.sp5 <= 1024). For instance, a dimension-3 window size is compared to a select value, e.g., 1024.

[0047] Additional, fewer and/or other tests may be performed.

[0048] If the tests are unsatisfactory 212, then error handling is performed 214. However, if the tests are satisfactory 212, then the function/selected operation (e.g., mean-reduce operation of an AVGPOOL2D function) is executed 216. For instance, in one embodiment, a general-purpose processor (e.g., general-purpose processor 104) initiates the Neural Network Processing Assist instruction and for certain functions, such as non-query functions, like the AVGPOOL2D function, the general-purpose processor provides information, such as an indication of the function/operation to be performed and memory address information for input data (e.g., one or more input tensors) to the special-purpose processor (e.g., neural network processor 105), such that the special-purpose processor can execute the function/operation, as described herein. When the function is complete, processing returns to the general-purpose processor to complete the instruction. In other embodiments, the general-purpose processor or the special-purpose processor initiates the instruction, performs the function/operation and completes the instruction. Other variations are possible.

[0049] Returning to inquiry 206, if, for instance, values of function-specific-parameter 2 and function-specific-parameter 3 are not equal to a select value, such as zero, then a further check is made as to whether values of fn.sp2 and fn.sp3 are, e.g., greater than a select value (e.g., zero) (220). If, for instance, values of the dimension-2-stride specified by fn.sp2 and the dimension-3-stride specified by fn.sp3 are not greater than the select value (e.g., 0), then error handling is performed 214. However, if values of the dimesion-2-stride specified by fn.sp2 and the dimension-3-stride specified by fn.sp3 are greater than the select value (e.g., zero), a pooling operation is to be performed, and thus, pooling specific tests are executed 230. Examples of these tests include, for instance, checking whether:

[0050] Values of fn.sp2 and fn.sp3 are less than or equal to a particular value, such as 30. For instance, dimension-2 and dimension-3 stride values are compared to a particular value, e.g., 30.

[0051] A padding type specified by function-specific-parameter 1 (fn.sp1) is equal to a select padding type, such as Valid. If the value in fn.sp1 is, e.g., Valid, then a check is made as to whether the sliding window value specified in fn.sp4 (also referred to as a dimension-2 window size) is less than or equal to a value of dimension-2 of a first input tensor (in1.e2) and whether the sliding window value specified in fn.sp5 (also referred to as a dimension-3 window size) is less than or equal to a value of dimension-3 of a first input tensor (in1.e3).

[0052] If, in one example, the padding type specified in fn.sp1 is not equal to the select padding type, such as Valid, then a check is made, for instance, as to whether a value of a dimension-2 (e2) of an output tensor (e.g., out.e2) is equal to a value of ceil (in1.e2/fn.sp4), and a check is made as to whether a value of a dimension-3 (e3) of the output tensor (e.g., out.e3) is equal to a value of ceil (in1.e3/fn.sp5). That is,

[0053] $$\mathrm{O1D2IS} = \left\lceil \frac{\mathrm{I1D2IS}}{\mathrm{D2S}} \right\rceil$$

[0054] $$\mathrm{O1D3IS} = \left\lceil \frac{\mathrm{I1D3IS}}{\mathrm{D3S}} \right\rceil$$

[0055] where:

[0056]	IxDyIS	Dimension-y-index-size of the input tensor x defined in tensor descriptor x.
[0057]	OxDyIS	Dimension-y-index-size of the output tensor x defined in tensor descriptor x.
[0058]	D2S	Dimension-2-stride.
[0059]	D3S	Dimension-3-stride.

[0060] Additional fewer and/or other tests may be performed.

[0061] If the tests are not satisfactory, then error handling is performed 214. However, if these tests are satisfactory 212, then the function/selected operation (e.g., pooling operation of an AVGPOOL2D function) is executed 216, as described herein.

[0062] Upon completion of the function/operation, processing returns to the general-purpose processor and the instruction completes 240.

[0063] As indicated, in one example the AVGPOOL2D function is implemented as part of an instruction, such as a Neural Network Processing Assist instruction. Further details relating to a Neural Network Processing Assist instruction, the AVGPOOL2D function, and the mean-reduce and pooling operations are described with reference to FIGS. 3A-3G. Referring initially to FIG. 3A, in one example, a Neural Network Processing Assist instruction 300 has an RRE format that denotes a register and register operation with an extended operation code (opcode). In one example, Neural Network Processing Assist instruction 300 includes an operation code (opcode) field 302 (e.g., bits 0-15) indicating a neural network processing assist operation. In one example, bits 16-31 of the instruction are reserved and are to contain zeros. In the description herein of the instruction, functions and/or operations of the instructions, specific locations, specific fields and/or specific sizes of the fields are indicated (e.g., specific bytes and/or bits). However, other locations, fields and/or sizes may be provided. Further, although the setting of a bit to a particular value, e.g., one or zero, may be specified, this is only an example. The bit, if set, may be set to a different value, such as the opposite value or to another value, in other examples. Many variations are possible.

[0064] In one example, the instruction uses a plurality of general registers implicitly specified by the instruction. For instance, Neural Network Processing Assist instruction 300 uses implied registers general register 0 and general register 1, examples of which are described with reference to FIGS. 3B and 3D, respectively.

[0065] Referring to FIG. 3B, in one example, general register 0 includes a function code field, and status fields which may be updated upon completion of the instruction. As an example, general register 0 includes a response code field 310 (e.g., bits 0-15), an exception flags field 312 (e.g., bits 24-31) and a function code field 314 (e.g., bits 56-63). Further, in one example, bits 16-23 and 32-55 of general register 0 are reserved and are to contain zeros. One or more fields are used by a particular function performed by the instruction. Not all fields are used by all of the functions, in one example. Each of the fields is described below:

[0066] Response Code (RC) 310: This field (e.g., bit positions 0-15) contains the response code. When execution of the Neural Network Processing Assist instruction completes with a condition code of, e.g., one, a response code is stored. When an invalid input condition is encountered, a non-zero value is stored to the response code field, which indicates the cause of the invalid input condition recognized during execution and a selected condition code, e.g., 1, is set. The codes stored to the response code field are defined, as follows, in one example:

	Response Code	Meaning
[0068]	0001	The format of the parameter block, as specified by the parameter block version number, is not supported by the model.
[0069]	0002	The specified function is not defined or installed on the machine.
[0070]	0010	A specified tensor data layout format is not supported.
[0071]	0011	A specified tensor data type is not supported.
[0072]	0012	A specified single tensor dimension is greater than the maximum dimension index size.

[0073] 0013 The size of a specified tensor is greater than the maximum tensor size.

[0074] 0014 The specified tensor address is not aligned on a 4 K-byte boundary.

[0075] 0015 The function-specific-save-area-address is not aligned on a 4 K-byte boundary.

[0076] F000-FFFF Function specific response codes. These response codes are defined for certain functions.

[0077] Exception Flags (EF) 312: This field (e.g., bit positions 24-31) includes the exception flags. If an exception condition is detected during execution of the instruction, the corresponding exception flag control (e.g., bit) will be set to, e.g., one; otherwise, the control remains unchanged. The exception flags field is to be initialized to zero prior to the first invocation of the instruction. Reserved flags are unchanged during execution of the instruction. The flags stored to the exception flags field are defined as follows, in one example:

[0078]

EF (Bit)	Meaning
[0079] 0	Range Violation. This flag is set when a non-numeric value was either detected in an input tensor or stored to the output tensor. This flag is, e.g., only valid when the instruction completes with condition code, e.g., 0.
[0080] 1-7	Reserved.

[0081] Function Code (FC) 314: This field (e.g., bit positions 56-63) includes the function code. Examples of assigned function codes for the Neural Network Processing Assist instruction are depicted in FIG. 3C. All other function codes are unassigned. If an unassigned or uninstalled function code is specified, a response code of, e.g., 0002 hex and a select condition code, e.g., 1, are set. This field is not modified during execution.

[0082] As indicated, in addition to general register 0, the Neural Network Processing Assist instruction also uses general register 1, an example of which is depicted in FIG. 3D. As examples, bits 40-63 in the 24-bit addressing mode, bits 33-63 in the 31-bit addressing

mode or bits 0-63 in the 64-bit addressing mode include an address of a parameter block 320. The contents of general register 1 specify, for instance, a logical address of a leftmost byte of the parameter block in storage. The parameter block is to be designated on a doubleword boundary; otherwise, a specification exception is recognized. For all functions, the contents of general register 1 are not modified.

[0083] In the access register mode, access register 1 specifies an address space containing the parameter block, input tensors, output tensors and the function specific save area, as an example.

[0084] In one example, the parameter block may have different formats depending on the function specified by the instruction to be performed. For instance, a query function of the instruction has a parameter block of one format and other functions of the instruction have a parameter block of another format. In another example, all functions use the same parameter block format. Other variations are also possible.

[0085] As examples, a parameter block and/or the information in the parameter block is stored in memory, in hardware registers and/or in a combination of memory and/or registers. Other examples are also possible.

[0086] One example of a parameter block used by a query function, such as a NNPA-Query Available Functions (QAF) operation, is described with reference to FIG. 3E. As shown, in one example, a NNPA-Query Available Functions parameter block 330 includes, for instance:

[0087] Installed Functions Vector 332: This field (e.g., bytes 0-31) of the parameter block includes the installed functions vector. In one example, bits 0-255 of the installed functions vector correspond to function codes 0-255, respectively, of the Neural Network Processing Assist instruction. When a bit is, e.g., one, the corresponding function is installed; otherwise, the function is not installed.

[0088] Installed Parameter Block Formats Vector 334: This field (e.g., bytes 32-47) of the parameter block includes the installed parameter block formats vector. In one example, bits 0-127 of the installed parameter block formats vector correspond to parameter block formats 0-127 for the non-query functions of the Neural Network Processing Assist

instruction. When a bit is, e.g., one, the corresponding parameter block format is installed; otherwise, the parameter block format is not installed.

[0089] Installed Data Types 336: This field (e.g., bytes 48-49) of the parameter block includes the installed data types vector. In one example, bits 0-15 of the installed data types vector correspond to the data types being installed. When a bit is, e.g., one, the corresponding data type is installed; otherwise, the data type is not installed. Example data types include (additional, fewer and/or other data types are possible):

[0090] Bit Data Type

[0091] 0 NNP-data-type-1

[0092] 1-15 Reserved

[0093] Installed Data Layout Formats 338: This field (e.g., bytes 52-55) of the parameter block includes the installed data layout formats vector. In one example, bits 0-31 of the installed data layout formats vector correspond to data layout formats being installed. When a bit is, e.g., one, the corresponding data layout format is installed; otherwise, the data layout format is not installed. Example data layout formats include (additional, fewer and/or other data types are possible):

[0094] Bit Data Layout Format

[0095] 0 4D-feature tensor

[0096] 1 4D-kernel tensor

[0097] 2-31 Reserved

[0098] Maximum Dimension Index Size 340: This field (e.g., bytes 60-63) of the parameter block includes, e.g., a 32-bit unsigned binary integer that specifies a maximum number of elements in a specified dimension index size for any specified tensor. In another example, the maximum dimension index size specifies a maximum number of bytes in a specified dimension index size for any specified tensor. Other examples are also possible.

[0099] Maximum Tensor Size 342: This field (e.g., bytes 64-71) of the parameter block includes, e.g., a 32-bit unsigned binary integer that specifies a maximum number of bytes in any specified tensor including any pad bytes required by the tensor format. In another

example, the maximum tensor size specifies a maximum number of total elements in any specified tensor including any padding required by the tensor format. Other examples are also possible.

[00100] Installed-NNP-Data-Type-1-Conversions Vector 344: This field (e.g., bytes 72-73) of the parameter block includes the installed-NNP-Data-Type-1-conversions vector. In one example, bits 0-15 of the installed-NNP-Data-Type-1-conversions vector correspond to installed data type conversion from/to NNP-data-type-1 format. When a bit is one, the corresponding conversion is installed; otherwise, the conversion is not installed. Additional, fewer and/or other conversions may be specified.

[00101] Bit Data Type

[00102] 0 Reserved

[00103] 1 BFP tiny format

[00104] 2 BFP short format

[00105] 3-15 Reserved

[00106] Although one example of a parameter block for a query function is described with reference to FIG. 3E, other formats of a parameter block for a query function, including the NNPA-Query Available Functions operation, may be used. The format may depend, in one example, on the type of query function to be performed. Further, the parameter block and/or each field of the parameter block may include additional, fewer and/or other information.

[00107] In addition to the parameter block for a query function, in one example, there is a parameter block format for non-query functions, such as non-query functions of the Neural-Network Processing Assist instruction. One example of a parameter block used by a non-query function, such as the AVGPOOL2D function of the Neural Network Processing Assist instruction, is described with reference to FIG. 3F.

[00108] As shown, in one example, a parameter block 350 employed by, e.g., the non-query functions of the Neural Network Processing Assist instruction includes, for instance:

[00109] Parameter Block Version Number 352: This field (e.g., bytes 0-1) of the parameter block specifies the version and size of the parameter block. In one example, bits 0-8 of the parameter block version number are reserved and are to contain zeros, and bits 9-15 of the parameter block version number contain an unsigned binary integer specifying the format of the parameter block. The query function provides a mechanism of indicating the parameter block formats available. When the size or format of the parameter block specified is not supported by the model, a response code of, e.g., 0001 hex is stored in general register 0 and the instruction completes by setting a condition code, e.g., condition code 1. The parameter block version number is specified by the program and is not modified during the execution of the instruction.

[00110] Model Version Number 354: This field (e.g., byte 2) of the parameter block is an unsigned binary integer identifying the model which executed the instruction (e.g., the particular non-query function). When a continuation flag (described below) is one, the model version number may be an input to the operation for the purpose of interpreting the contents of a continuation state buffer field (described below) of the parameter block to resume the operation.

[00111] Continuation Flag 356: This field (e.g., bit 63) of the parameter block, when, e.g., one, indicates the operation is partially complete and the contents of the continuation state buffer may be used to resume the operation. The program is to initialize the continuation flag to zero and not modify the continuation flag in the event the instruction is to be re-executed for the purpose of resuming the operation; otherwise, results are unpredictable.

[00112] If the continuation flag is set at the beginning of the operation and the contents of the parameter block have changed since the initial invocation, results are unpredictable.

[00113] Function-specific-save-area-address 358: This field (e.g., bytes 56-63) of the parameter block includes the logical address of the function specific save area. In one example, the function-specific-save-area-address is to be aligned on a 4 K-byte boundary; otherwise, a response code of, e.g., 0015 hex is set in general register 0 and the instruction completes with a condition code of, e.g., 1. The address is subject to the current addressing mode. The size of the function specific save area depends on the function code.

[00114] When the entire function specific save area overlaps the program event recording (PER) storage area designation, a PER storage alteration event is recognized, when applicable, for the function specific save area. When only a portion of the function specific save area overlaps the PER storage area designation, it is model-dependent which of the following occurs:

[00115] * A PER storage alteration event is recognized, when applicable, for the entire function specific save area.

[00116] * A PER storage alteration event is recognized, when applicable, for the portion of the function specific save area that is stored.

[00117] When the entire parameter block overlaps the PER storage area designation, a PER storage alteration event is recognized, when applicable, for the parameter block. When only a portion of the parameter block overlaps the PER storage area designation, it is model-dependent which of the following occurs:

[00118] * A PER storage alteration event is recognized, when applicable, for the entire parameter block.

[00119] * A PER storage alteration event is recognized, when applicable, for the portion of the parameter block that is stored.

[00120] A PER zero-address detection event is recognized, when applicable, for the parameter block. Zero address detection does not apply to the tensor addresses or the function-specific-save-area-address, in one example.

[00121] Output Tensor Descriptors (e.g., 1-2) 360/Input Tensor Descriptors (e.g., 1-3) 365: One example of a tensor descriptor is described with reference to FIG. 3G. In one example, a tensor descriptor 360, 365 includes:

[00122] Data Layout Format 382: This field (e.g., byte 0) of the tensor descriptor specifies the data layout format. Valid data layout formats include, for instance (additional, fewer and/or other data layout formats are possible):

[00123]

Format	Description	Alignment (bytes)

[00124]

0	4D-feature tensor	4096

[00125] 1 4D-kernel tensor 4096

[00126] 2-255 Reserved --

[00127] If an unsupported or reserved data layout format is specified, the response code of, e.g., 0010 hex, is stored in general register 0 and the instruction completes by setting condition code, e.g., 1.

[00128] Data Type 384: This field (e.g., byte 1) specifies the data type of the tensor. Examples of supported data types are described below (additional, fewer and/or other data types are possible):

[00129]

Value	Data Type	Data Size (bits)
[00130] 0	NNP data-type-1	16
[00131] 1-255	Reserved	--

[00132] If an unsupported or reserved data type is specified, a response code of, e.g., 0011 hex is stored in general register 0 and the instruction completes by setting condition code, e.g., 1.

[00133] Dimension 1-4 Index Size 386: Collectively, dimension index sizes one through four specify the shape of a 4D tensor. Each dimension index size is to be greater than zero and less than or equal to the maximum dimension index size (340, FIG. 3E); otherwise, a response code of, e.g., 0012 hex is stored in general register 0 and the instruction completes by setting condition code, e.g., 1. The total tensor size is to be less than or equal to the maximum tensor size (342, FIG. 3E); otherwise, a response code, e.g., 0013 hex is stored in general register 0 and the instruction completes by setting condition code, e.g., 1.

[00134] In one example, to determine the number of bytes in a 4D-feature tensor with elements of NNPA-data-type-1 (i.e., total tensor size), the following is used: dimension-index-4 * dimension-index-3 * ceil (dimension-index-2/32) * 32 * ceil (dimension-index-1/64) * 64 * 2.

[00135] Tensor Address 388: This field (e.g., bytes 24-31) of the tensor descriptor includes a logical address of the leftmost byte of the tensor. The address is subject to the current addressing mode.

[00136] If the address is not aligned on the boundary of the associated data layout format, a response code of, e.g., 0014 hex, is stored in general register 0 and the instruction completes by setting condition code, e.g., 1.

[00137] In the access register mode, access register 1 specifies the address space containing all active input and output tensors in storage.

[00138] Returning to FIG. 3F, parameter block 350 further includes, in one example, function- specific-parameters 1-5 (370), which may be used by specific functions, as described herein.

[00139] Further, parameter block 350 includes, in one example, a continuation state buffer field 375, which includes data (or a location of data) to be used if operation of this instruction is to be resumed.

[00140] As an input to the operation, reserved fields of the parameter block should contain zeros. When the operation ends, reserved fields may be stored as zeros or remain unchanged.

[00141] Although one example of a parameter block for a non-query function is described with reference to FIG. 3F, other formats of a parameter block for a non-query function, including a non-query function of the Neural Network Processing Assist instruction, may be used. The format may depend, in one example, on the type of function to be performed. Further, although one example of a tensor descriptor is described with reference to FIG. 3G, other formats may be used. Further, different formats for input and output tensors may be used. Other variations are possible.

[00142] Further details regarding various functions supported by one embodiment of the Neural Network Processing Assist instruction are described below:

[00143] Function Code 0: NNPA-QAF (Query Available Functions)

[00144] The Neural Network Processing Assist (NNPA) query function provides a mechanism to indicate selected information, such as, for instance, the availability of installed functions, installed parameter block formats, installed data types, installed data layout formats, maximum dimension index size and maximum tensor size. The information is obtained and placed in a selected location, such as a parameter block (e.g., parameter

block 330). When the operation ends, reserved fields of the parameter block may be stored as zeros or may remain unchanged.

[00145] In execution of one embodiment of the query function, a processor, such as general-purpose processor 104, obtains information relating to a specific processor, such as a specific model of a neural network processor, such as neural network processor 105. A specific model of a processor or machine has certain capabilities. Another model of the processor or machine may have additional, fewer and/or different capabilities and/or be of a different generation (e.g., a current or future generation) having additional, fewer and/or different capabilities. The obtained information is placed in a parameter block (e.g., parameter block 330) or other structure that is accessible to and/or for use with one or more applications that may use this information in further processing. In one example, the parameter block and/or information of the parameter block is maintained in memory. In other embodiments, the parameter block and/or information may be maintained in one or more hardware registers. As another example, the query function may be a privileged operation executed by the operating system, which makes available an application programming interface to make this information available to the application or non-privileged program. In yet a further example, the query function is performed by a special-purpose processor, such as neural network processor 105. Other variations are possible.

[00146] The information is obtained, e.g., by the firmware of the processor executing the query function. The firmware has knowledge of the attributes of the specific model of the specific processor (e.g., neural network processor). This information may be stored in, e.g., a control block, register and/or memory and/or otherwise be accessible to the processor executing the query function.

[00147] The obtained information includes, for instance, model-dependent detailed information regarding at least one or more data attributes of the specific processor, including, for instance, one or more installed or supported data types, one or more installed or supported data layout formats and/or one or more installed or supported data sizes of the selected model of the specific processor. This information is model-dependent in that other models (e.g., previous models and/or future models) may not support the same data attributes, such as the same data types, data sizes and/or data layout formats. When execution of the query function (e.g., NNPA-QAF function) completes, condition code 0, as

an example, is set. Condition codes 1, 2 and 3 are not applicable to the query function, in one example. Further information relating to the obtained information is described below.

[00148] As indicated, in one example, the obtained information includes model-dependent information about one or more data attributes of, e.g., a particular model of a neural network processor. One example of a data attribute is installed data types of the neural network processor. For instance, a particular model of a neural network processor (or other processor) may support one or more data types, such as a NNP-data-type-1 data type (also referred to as a neural network processing-data-type-1 data type) and/or other data types, as examples. The NNP-data-type-1 data type is a 16-bit floating-point format that provides a number of advantages for deep learning training and inference computations, including, for instance: preserves the accuracy of deep learning networks; eliminates the subnormal format which simplifies rounding modes and handling of corner cases; automatic rounding to the nearest value for arithmetic operations; and special entities of infinity and not-a-number (NaN) are combined into one value (NINF), which is accepted and handled by arithmetic operations. NINF provides better defaults for exponent overflow and invalid operations (such as division by zero). This allows many programs to continue running without hiding such errors and without using specialized exception handlers. Other model-dependent data types are also possible.

[00149] One example of a format of the NNP-data-type-1 data type is depicted in FIG. 4. As depicted, in one example, NNP-data-type-1 data may be represented in a format 400, which includes, for instance, a sign 402 (e.g., bit 0), an exponent + 31 404 (e.g., bits 1-6) and a fraction 406 (e.g., bits 7-15).

[00150] Example properties of the NNP-data-type-1 format are depicted below:

	Property	NNP-data-type-1
[00152]	Format length (bits)	16 bits
[00153]	Biased-exponent length (bits)	6 bits
[00154]	Fraction length (bits)	9 bits
[00155]	Precision (p)	10 bits
[00156]	Maximum left-units-view exponent (Emax)	32

[00157] Minimum left-units-view exponent (Emin) -31

[00158] Left-units-view (LUV) bias 31

[00159] Nmax $(1-2^{-9}) \times 2^{33} \approx 8.6 \times 10^{9}$

[00160] Nmin $(1+2^{-9}) \times 2^{-31} \approx 4.6 \times 10^{-10}$

[00161] Dmin ---

Where ≈ indicates that the value is approximate, Nmax is largest (in magnitude) representable finite number, and Nmin is smallest (in magnitude) representable number.

[00162] Further details relating to the NNP-data-type-1 data type are described below:

[00163] Biased Exponent: The bias that is used to allow exponents to be expressed as unsigned numbers is shown above. Biased exponents are similar to characteristics of the binary floating-point format, except that no special meanings are attached to biased exponents of all zeros and all ones, as described below with reference to the classes of the NNP-data-type-1 data type.

[00164] Significand: The binary point of a NNP-data-type-1 number is considered to be to the left of the leftmost fraction bit. To the left of the binary point there is an implied unit bit, which is considered to be one for normal numbers and zero for zeros. The fraction with the implied unit bit appended on the left is the significand of the number.

[00165] The value of a normal NNP-data-type-1 is the significand multiplied by the radix 2 raised to the power of the unbiased exponent.

[00166] Values of Non-Zero Numbers: The values of non-zero numbers are shown below:

[00167] <u>Number Class</u> <u>Value</u>

[00168] Normal numbers $\pm 2^{e-31} \times (1.f)$

Where e is biased exponent shown in decimal, and f is fraction in binary.

[00169] In one embodiment, there are three classes of NNP-data-type-1 data, including numeric and related non-numeric entities. Each data item includes a sign, an exponent and a significand. The exponent is biased such that all biased exponents are non-negative unsigned numbers and the minimum biased exponent is zero. The significand includes an explicit fraction and an implicit unit bit to the left of the binary point. The sign bit is zero for plus and one for minus.

[00170] All non-zero finite numbers permitted have a unique NNP-data-type-1 representation. There are no subnormal numbers, which numbers might allow multiple representations for the same values, and there are no subnormal arithmetic operations. The three classes include, for instance:

	Data Class	Sign	Biased Exponent	Unit Bit*	Fraction
[00172]	Zero	±	0	0	0
[00173]	Normal numbers	±	0	1	Not 0
[00174]	Normal numbers	±	Not 0, not all ones	1	Any
[00175]	Normal numbers	±	All ones	-	Not all ones
[00176]	NINF	±	All ones	-	All ones

Where: – indicates does not apply, * indicates the unit bit is implied, NINF is not a number or infinity.

[00177] Further details regarding each of the classes are described below:

[00178] Zeros: Zeros have a biased exponent of zero and a zero fraction. The implied unit bit is zero.

[00179] Normal Numbers: Normal numbers may have a biased exponent of any value. When the biased exponent is 0, the fraction is to be non-zero. When the biased exponent is all ones, the fraction is not to be all ones. Other biased exponent values may have any fraction value. The implied unit bit is one for all normal numbers.

[00180] NINF: A NINF is represented by a biased exponent of all ones and a fraction of all ones. A NINF represents a value not in the range of representable values in NNP-data-

type-1 (i.e., 16-bit floating point designed for deep learning that has 6 exponent bits and 9 fraction bits). Normally, NINFs are just propagated during computations so that they will remain visible at the end.

[00181] Although the NNP-data-type-1 data type is supported in one example, other specialized or non-standard data types may be supported, as well as one or more standard data types including, but not limited to: IEEE 754 short precision, binary floating-point 16-bit, IEEE half precision floating point, 8-bit floating point, 4-bit integer format and/or 8-bit integer format, to name a few. These data formats have different qualities for neural network processing. As an example, smaller data types (e.g., less bits) can be processed faster and use less cache/memory, and larger data types provide greater result accuracy in the neural network. A data type to be supported may have one or more assigned bits in the query parameter block (e.g., in installed data types field 336 of parameter block 330). For instance, specialized or non-standard data types supported by a particular processor are indicated in the installed data types field but standard data types are not indicated. In other embodiments, one or more standard data types are also indicated. Other variations are possible.

[00182] In one particular example, bit 0 of installed data types field 336 is reserved for the NNP-data-type-1 data type, and when it is set to, e.g., 1, it indicates that the processor supports NNP-data-type-1. As an example, the bit vector of installed data types is configured to represent up to 16 data types, in which a bit is assigned to each data type. However, a bit vector in other embodiments may support more or fewer data types. Further, a vector may be configured in which one or more bits are assigned to a data type. Many examples are possible and/or additional, fewer and/or other data types may be supported and/or indicated in the vector.

[00183] In one example, the query function obtains an indication of the data types installed on the model-dependent processor and places the indication in the parameter block by, e.g., setting one or more bits in installed data types field 336 of parameter block 330. Further, in one example, the query function obtains an indication of installed data layout formats (another data attribute) and places the information in the parameter block by, e.g., setting one or more bits in installed data layout formats field 338. Example data layout formats include, for instance, a 4D-feature tensor layout and a 4D-kernel tensor layout. The 4D-feature tensor layout is used, in one example, by the functions indicated herein, and in

one example, the convolution function uses the 4D-kernel tensor layout. These data layout formats arrange data in storage for a tensor in a way that increases processing efficiency in execution of the functions of the Neural Network Processing Assist instruction. For instance, to operate efficiently, the Neural Network Processing Assist instruction uses input tensors provided in particular data layout formats. Although example layouts are provided, additional, fewer and/or other layouts may be provided for the functions described herein and/or other functions.

[00184] The use or availability of layouts for a particular processor model is provided by the vector of installed data layout formats (e.g., field 338 of parameter block 330). The vector is, for instance, a bit vector of installed data layout formats that allows the CPU to convey to applications which layouts are supported. For instance, bit 0 is reserved for the 4D-feature tensor layout, and when it is set to, e.g., 1, it indicates that the processor supports a 4D-feature tensor layout; and bit 1 is reserved for the 4D-kernel tensor layout, and when it is set to, e.g., 1, it indicates that the processor supports a 4D-kernel tensor layout. In one example, the bit vector of installed data layout formats is configured to represent up to 16 data layouts, in which a bit is assigned to each data layout. However, a bit vector in other embodiments may support more or fewer data layouts. Further, a vector may be configured in which one or more bits are assigned to data layouts. Many examples are possible. Further details regarding the 4D-feature tensor layout and the 4D-kernel tensor layout are described below. Again, other layouts may be used now or in the future to optimize performance.

[00185] In one example, the Neural Network Processing Assist instruction operates with 4D-tensors, i.e., tensors with 4 dimensions. These 4D-tensors are obtained from generic input tensors described herein in, e.g., row-major, i.e., when enumerating the tensor elements in increasing memory address order, the inner dimension called E1 will be stepped up first through the E1-index-size values starting with 0 through the E1-index-size -1, before the index of the E2 dimension will be increased and the stepping through the E1 dimension is repeated. The index of the outer dimension called the E4 dimension is increased last.

[00186] Tensors that have a lower number of dimensions (e.g., 3D- or 1D-tensors) will be represented as 4D-tensors with one or more dimensions of the 4D-tensor exceeding the original tensor dimensions set to 1.

[00187] The transformation of a row-major generic 4D-tensor with dimensions E4, E3, E2, E1 into a 4D-feature tensor layout (also referred to herein as NNPA data layout format 0 4D-feature tensor) is described herein:

[00188] A resulting tensor can be represented, for instance, as a 4D-tensor of, e.g., 64-element vectors or a 5D-tensor with dimensions:

[00189] $E4, \lceil E1/64 \rceil, E3, \lceil E2/32 \rceil * 32, 64$, where $\lceil\ \rceil$ refers to a ceil function. (Stated another way: E4 * E3 * ceil (E2/32) * 32 * ceil (E1/64) * 64 elements.)

[00190] An element [e4][e3][e2][e1] of the generic tensor may be mapped to the following element of the resulting 5D-tensor:

[00191] $[e4][\lfloor e1/64 \rfloor][e3][e2][e1\ \mathrm{MOD}\ 64]$, where $\lfloor\ \rfloor$ is a floor function and mod is modulo. (Stated another way: element (E3 * e2_limit * e1_limit * e4x) + (e2_limit * e3x * 64) + (e2x * 64) + ($\lfloor$ e1x/64$\rfloor$ * e2_limit * E3 * 64) + (e1x mod 64), where e2_limit = $\lceil E2/32 \rceil$ * 32 and e1_limit = $\lceil E1/64 \rceil$ * 64.)

[00192] The resulting tensor may be larger than the generic tensor. Elements of the resulting tensor with no corresponding elements in the generic tensor are called pad elements.

[00193] Consider the element [fe4][fe1][fe3][fe2][fe0] of a NNPA data layout format 0 4D-feature tensor of a 64-element vector or its equivalent representation as a 5D-tensor of elements. This element is either a pad element or its corresponding element in the generic 4D tensor with dimensions E4, E3, E2, E1 can be determined with the following formula:

[00194] · if fe2 ≥ E2 then this is an E2 (or page)-pad element

[00195] · else if fe1*64+fe0 ≥ E1 then this is an E1 (or row)-pad element

[00196] · else corresponding element in generic 4D tensor is:

[00197] [fe4][fe3][fe2][fe1*64+fe0]

[00198] For convolutional neural network based artificial intelligence models, the meaning of the 4 dimensions of a feature tensor can generally be mapped to:

[00199] · E4: N – Size of mini-batch

[00200] · E3: H – Height of the 3D-tensor/image

[00201] · E2: W – Width of the 3D-tensor/image

[00202] · E1: C – Channels or classes of the 3D-tensor

[00203] For machine learning or recurrent neural network based artificial intelligence models, the meaning of the 4 dimensions of a 4D-feature tensor may generally be mapped to:

[00204] · E4: T – Number of time-steps or models

[00205] · E3: Reserved, generally set to 1

[00206] · E2: N_{mb} – Minibatch size

[00207] · E1: L – Features

[00208] The NNPA data layout format 0 provides, e.g., two dimensional data locality with 4k-Bytes blocks of data (pages) as well as 4k-Byte block data alignment for the outer dimensions of the generated tensor.

[00209] Pad element bytes are ignored for the input tensors and unpredictable for output tensors. PER storage-alteration on pad bytes is unpredictable.

[00210] One example of an input data layout for a 4D-feature tensor layout, which has dimensions E1, E2, E3 and E4, is shown in FIGS. 5A-5C, and an example output for the 4D-feature tensor layout is depicted in FIGS. 6A-6C. Referring to FIG. 5A, a 3D-tensor 500 is shown, which has dimensions E1, E2 and E3. In one example, each 3D-tensor includes a plurality of 2D-tensors 502. The numbers in each 2D-tensor 502 describe memory offsets of where each of its elements would be in memory. The inputs are used to lay-out the data of the original tensor (e.g., original 4D-tensor of FIGS. 5A-5C) in memory, as shown in FIGS. 6A-6C, which correspond to FIGS. 5A-5C.

[00211] In FIG. 6A, as an example, a unit of memory 600 (e.g., a memory page) includes a pre-selected number (e.g., 32) of rows 602, each of which is identified by, e.g., e2_page_idx; and each row has a pre-selected number (e.g., 64) of elements 604, each identified by, e.g., e1_page_idx. If a row does not include the pre-selected number of elements, it is padded 606, referred to as row or E1 padding; and if the memory unit does

not have a pre-selected number of rows, it is padded 608, referred to as page or E2 padding. As examples, the row padding is e.g., zeros or other values, and the page padding is, e.g., existing values, zeros, or other values.

[00212] In one example, output elements of a row are provided in memory (e.g., in a page) based on element positions in the E1 direction of its corresponding input. For instance, referring to FIG. 5A, element positions 0, 1 and 2 of the three matrices shown (e.g., element positions at a same location in each matrix) are shown in row 0 of page 0 of FIG. 6A, etc. In this example, the 4D-tensor is small and all of the elements of each 2D-tensor representing the 4D-tensor fits in one page. However, this is only one example. A 2D-tensor may include one or more pages. If a 2D-tensor is created based on a reformatting of a 4D-tensor, then the number of pages of the 2D-tensor is based on the size of the 4D-tensor. In one example, one or more ceil functions are used to determine the number of rows in a 2D-tensor and the number of elements in each row, which will indicate how many pages are to be used. Other variations are possible.

[00213] In addition to the 4D-feature tensor layout, in one example, a neural network processor may support a 4D-kernel tensor, which re-arranges the elements of a 4D-tensor to reduce the number of memory accesses and data gathering steps when executing certain artificial intelligence (e.g., neural network processing assist) operations, such as a convolution. As an example, a row-major generic 4D-tensor with dimensions E4, E3, E2, E1 is transformed into a NNPA data layout format 1 4D-kernel tensor (4D-kernel tensor), as described herein:

[00214] A resulting tensor can be represented as a 4D-tensor of, e.g., 64-element vectors or a 5D-tensor with dimensions:

[00215] $\lceil E1/64 \rceil, E4, E3, \lceil E2/32 \rceil * 32, 64$, where $\lceil\ \rceil$ refers to a ceil function. (Stated another way: E4 * E3 * ceil (E2/32) * 32 * ceil (E1/64) * 64 elements.)

[00216] An element [e4][e3][e2[e1] of the generic tensor may be mapped to the following element of the resulting 5D-tensor:

[00217] $[\lfloor e1/64 \rfloor][e4][e3][e2][e1\ \mathrm{MOD}\ 64]$, where $\lfloor\ \rfloor$ refers to a floor function and mod is modulo. Stated another way: element $(\lfloor e1x/64 \rfloor$ * E4 * E3 * e2_limit * 64) + (e4x

* E3 * e2_limit * 64) + (e3x * e2_limit * 64) + (e2x * 64) + (e1x mod 64), where e2_limit = $\lceil E2/32 \rceil$ * 32 and e1_limit = $\lceil E1/64 \rceil$ * 64.

[00218] The resulting tensor may be larger than the generic tensor. Elements of the resulting tensor with no corresponding elements in the generic tensor are called pad elements.

[00219] Consider the element [fe1][fe4][fe3][fe2][fe0] of a NNPA data layout format 1 4D-feature tensor of 64-element vectors or its equivalent representation as a 5D-tensor of elements. This element is either a pad element or its corresponding element in the generic 4D tensor with dimensions E4, E3, E2, E1 can be determined with the following formula:

[00220] · if fe2 ≥ E2 then this is an E2 (or page)-pad element

[00221] · else if fe1*64+fe0 ≥ E1 then this is an E1 (or row)-pad element

[00222] · else corresponding element in generic 4D tensor is

[00223] [fe4][fe3][fe2][fe1*64+fe0]

[00224] For convolutional neural network based artificial intelligence models, the meaning of the 4 dimensions of a kernel tensor can generally be mapped to:

[00225] · E4: H – Height of the 3D-tensor/image

[00226] · E3: W – Width of the 3D-tensor/image

[00227] · E2: C – Number of Channels of the 3D-tensor

[00228] · E1: K – Number of Kernels

[00229] The NNPA data layout format 1 provides, e.g., two dimensional kernel parallelism within 4k-Byte blocks of data (pages) as well as 4k-Byte block data alignment for the outer dimensions of the generate tensor for efficient processing.

[00230] Pad bytes are ignored for the input tensors. PER storage-alteration on pad bytes is unpredictable.

[00231] Again, although example data layout formats include a 4D-feature tensor layout and a 4D-kernel tensor layout, other data layout formats may be supported by the processor (e.g., neural network processor 105). An indication of supported data layouts is obtained and placed in the query parameter block by setting one or more bits in, e.g., field 338.

[00232] The query parameter block also includes, in accordance with one or more aspects of the present invention, other data attribute information, which includes, e.g., supported size information for the data. A processor, such as a neural network processor, typically has limitations based on internal buffer sizes, processing units, data bus structures, firmware limitations, etc. that can limit the maximum size of tensor dimensions and/or the overall size of a tensor. Therefore, the query function provides fields to convey these limits to applications. For instance, the processor, based on executing the query function, obtains various data sizes, such as a maximum dimension index size (e.g., 65,536 elements) and a maximum tensor size (e.g., 8 GB), and includes this information in fields 340 and 342, respectively, of the parameter block (e.g., parameter block 330). Additional, fewer and/or other size information may also be supported by the processor (e.g., neural network processor 105), and thus, obtained and placed in the parameter block, e.g., fields 340, 342 and/or other fields. In other embodiments, the limitations could be smaller or larger, and/or the sizes may be in other units, such as bytes instead of elements, elements instead of bytes, etc. Further, other embodiments allow for different maximum sizes of each dimension, rather than the same maximum for all dimensions. Many variations are possible.

[00233] In accordance with one or more aspects of the present invention, a query function is provided that conveys detailed information relating to a specific model of a selected processor (e.g., neural network processor 105). The detailed information includes, for instance, model-dependent information relating to a specific processor. (A processor may also support standard data attributes, such as standard data types, standard data layouts, etc., which are implied and not necessarily presented by the query function; although, in other embodiments, the query function may indicate all or various selected subsets of data attributes, etc.) Although example information is provided, other information may be provided in other embodiments. The obtained information, which may be different for different models of a processor and/or of different processors, is used to perform artificial intelligence and/or other processing. The artificial intelligence and/or other processing may employ one or more non-query functions of, e.g., the Neural Network Processing Assist instruction. A specific non-query function employed in the processing is performed by

executing the Neural Network Processing Assist instruction one or more times and specifying the non-query specific function.

[00234] Examples of non-query functions supported by the Neural Network Processing Assist instruction include an AVGPOOL2D function and a MAXPOOL2D function, each of which is described below (additional, fewer and/or other functions are supported in one or more embodiments).

[00235] Function Code 80: NNPA-MAXPOOL2D

Function Code 81: NNPA-AVGPOOL2D

[00236] When either the NNPA-MAXPOOL2D or the NNPA-AVGPOOL2D function is specified, input tensor 1, described by the input tensor 1 descriptor (e.g., see FIG. 3G), is reduced by the specified operation to summarize windows of the input. The windows of the input are selected by moving a 2D sliding window over dimensions 2 and 3. A summary of the window is an element in the output tensor. The sliding window dimensions are described by, e.g., function-specific-parameter 4 and function-specific-parameter 5. The amount that the sliding window moves over the input tensor 1 when computing adjacent output tensor elements is called the stride. The sliding window stride is specified by, e.g., function-specific-parameter 2 and function-specific-parameter 3. When the NNPA-MAXPOOL2D operation is specified, the Max operation defined below is performed on the window. When the NNPA-AVGPOOL2D operation is specified, the AVG operation defined below is performed on the window. If the specified padding type is Valid, all elements in the window are added to the collection used to compute the resulting output element. If the specified padding type is Same, depending on the location of the window, only a subset of elements from the window may be added to the collection used to compute the resulting output element (e.g., those elements outside the bounds of the tensor may be ignored).

[00237] In one example, a CollectElements operation adds an element to the collection of elements and increments the number of elements in the collection. Each time the window start position moves, the collection is emptied. It is unpredictable whether elements not required to perform the operations are accessed.

[00238] Max Operation: In one example, the maximum value of the collection of elements in the window is computed by comparing all elements in the collection to each other and returning the largest value.

[00239] AVG (Average) Operation: In one example, the average value of the collection of elements in the window is computed as the summation of, e.g., all elements in the collection divided by the number of elements in the collection.

[00240] In one example, fields are allocated as follows:

[00241] * A pooling function-specific-parameter 1 controls the padding type. For instance, bits 29-31 of function-specific-parameter 1 include a PAD field that specifies the padding type. Example types include, for instance:

	PAD	Padding Type
[00242]		
[00243]	0	Valid
[00244]	1	Same
[00245]	2-7	Reserved

[00246] If a reserved value is specified for the PAD field, a response code of, e.g., F000 hex is reported and the operation completes with condition code, e.g., 1.

[00247] In one example, bit positions 0-28 of function-specific-parameter 1 are reserved and are to contain zeros.

[00248] * Function-specific-parameter 2 contains, e.g., a 32-bit unsigned binary integer that specifies the dimension-2-stride (D2S) which specifies the number of elements the sliding window moves in dimension 2 (also referred to as e2).

[00249] * Function-specific-parameter 3 contains, e.g., a 32-bit unsigned binary integer that specifies the dimension-3-stride (D3S) which specifies the number of elements the sliding window moves in dimension 3 (also referred to as e3).

[00250] * Function-specific-parameter 4 contains, e.g., a 32-bit unsigned binary integer that specifies the dimension-2-window-size (D2WS) which specifies the number of elements in dimension 2 the sliding window contains.

[00251] * Function-specific-parameter 5 contains, e.g., a 32-bit unsigned binary integer that specifies the dimension-3-window-size (D3WS) which specifies the number of elements in dimension 3 the sliding window contains.

[00252] In one example, the specified values in function-specific-parameters 2-5 are to be less than or equal to the maximum dimension index size, and the specified values in function-specific-parameters 4-5 are to be greater than, e.g., zero; otherwise, response code, e.g., 0012 hex is reported and the operation completes with condition code, e.g., 1.

[00253] If the dimension-2-stride and the dimension-3-stride are both zero and either the dimension-2-window size or the dimension-3-window size is greater than, e.g., 1024, response code, e.g., F001 hex is stored. If the dimension-2-stride and the dimension-3-stride are both greater than, e.g., zero and either the dimension-2-window-size or the dimension-3-window-size is greater than, e.g., 64, response code, e.g., F002 hex is stored. If the dimension-2-stride and the dimension-3-stride are both greater than, e.g., zero, and either the dimension-2 stride or the dimension-3 stride is greater than, e.g., 30, response code, e.g., F003 hex is stored. If the dimension-2-stride and the dimension-3-stride are both greater than, e.g., zero and either the input tensor dimension-2-index-size or the input tensor dimension-3-index-size is greater than, e.g., 1024, response code, e.g., F004 hex is stored. For all of the above conditions, the instruction completes with condition code, e.g., 1.

[00254] In one example, if the specified data layout in any of the specified tensor descriptors does not specify a 4D-feature tensor (e.g., data-layout = 0) or if the data-type in any specified tensor descriptor does not specify NNP-data-type-1 (e.g., data-type = 0), response code, e.g., 0010 hex or 0011 hex, respectively, is set in general register 0 and the instruction completes with condition code, e.g., 1.

[00255] In one example, the following conditions are to be true, otherwise, a general operand data exception is recognized:

[00256] * The dimension-4-index-sizes and dimension-1-index-sizes of the input tensor and the output tensor are to be the same.

[00257] * The data layout and the data type of the input tensor and the output tensor are to be the same.

[00258] * If the dimension-2-stride and the dimension-3-stride are both zero (specifying, e.g., a mean-reduce operation of the AVGPOOL2D function), the following additional conditions are to be true, in one example:

[00259] * The input tensor dimension-2-index-size is to be equal to the dimension-2-window size.

[00260] * The input tensor dimension-3-index-size is to be equal to the dimension-3-window-size.

[00261] * The dimension-2-index-size and the dimension-3-index-size of the output tensor are to be one.

[00262] * The specified padding is to be valid

[00263] * If either the dimension-2-stride or the dimension-3-stride is non-zero, then both strides are to be non-zero, in one example.

[00264] * If the dimension-2-stride and the dimension-3-stride are both greater than zero (specifying, e.g., a pooling operation of the AVGPOOL2D function), the following additional conditions are to be true, in one example:

[00265] * When the specified padding is Valid, the dimension-2-window-size is to be less than or equal to the dimension-2-index-size of the input tensor.

[00266] * When the specified padding is Valid, the dimension-3-window-size is to be less than or equal to the dimension-3-index-size of the input tensor.

[00267] * When the specified padding is Same, the following relationships between the dimension-2-index-size and dimension-3-index size of

the input and output tensors are to be satisfied (Pooling Same Padding):

[00268] $$O1D2IS = \left\lceil \frac{I1D2IS}{D2S} \right\rceil$$

[00269] $$O1D3IS = \left\lceil \frac{I1D3IS}{D3S} \right\rceil$$

[00270] where:

[00271] IxDyIS Dimension-y-index-size of the input tensor x defined in tensor descriptor x.

[00272] OxDyIS Dimension-y-index-size of the output tensor x defined in tensor descriptor x.

[00273] D2S Dimension-2-stride.

[00274] D3S Dimension-3-stride.

[00275] * When the specified padding is Valid, the following relationships between the dimension-2-index-size and dimension-3-index-size of the input and output tensors are to be satisfied (Pooling Valid Padding):

[00276] $$O1D2IS = \left\lceil \frac{(I1D2IS - D2WS + 1)}{D2S} \right\rceil$$

[00277] $$O1D3IS = \left\lceil \frac{(I1D3IS - D3WS + 1)}{D3S} \right\rceil$$

[00278] where D2WS is dimension-2-window size and D3WS is dimension-3-window size.

[00279] The output tensor descriptor 2, input tensor descriptors 2 and 3, and function-specific-save-area-address field are ignored.

[00280] For the Neural Network Processing Assist instruction, in one embodiment, if the output tensor overlaps with any input tensor or the parameter block, results are unpredictable.

[00281] A specification exception is recognized when execution of the Neural Network Processing Assist instruction is attempted and the parameter block is not designated on, e.g., a doubleword boundary, as an example.

[00282] A general operand data exception is recognized when execution of the Neural Network Processing Assist instruction is attempted and there are, for instance, tensor descriptor inconsistencies.

[00283] Resulting Condition Codes for the Neural Network Processing Assist instruction include, for instance: 0 – Normal completion; 1 – Response code is set; 2 --; 3 – CPU-determined amount of data processed.

[00284] In one embodiment, the priority of execution for the Neural Network Processing Assist instruction includes, for instance:

[00285] 1.-7. Exceptions with the same priority as the priority of program interruption conditions for the general case.

[00286] 8.A Condition code 1 due to an unassigned or uninstalled function code specified.

[00287] 8.B Specification exception due to parameter block not designated on doubleword boundary.

[00288] 9. Access exceptions for an access to the parameter block.

[00289] 10. Condition code 1 due to specified format of the parameter block not supported by the model.

[00290] 11.A Condition code 1 due to the specified tensor data layouts are not supported.

[00291] 11.B General operand data exception due to differing data layouts between tensor descriptors.

[00292] 12.A Condition code 1 due to conditions other than those included in items 8.A, 10 and 11.A above and 12.B.1 below.

[00293] 12.B.1 Condition code 1 due to invalid output tensor data type for NNPA-RELU and NNPA-CONVOLUTION.

[00294] 12.B.2 General operand data exception for invalid value for NNPA-RELU function-specific-parameter 1 and NNPA-CONVOLUTION function-specific-parameter 4.

[00295] 13.A Access exceptions for an access to the output tensor.

[00296] 13.B Access exceptions for an access to the input tensors.

[00297] 13.C Access exceptions for an access to the function-specific-save-area.

[00298] 14. Condition code 0.

[00299] As described herein, a single instruction (e.g., the Neural Network Processing Assist instruction) is configured to perform a plurality of functions, including a query function and a plurality of non-query functions. At least one non-query function, the AVGPOOL2D function, is configured to implement a plurality of operations (e.g., mean-reduce and pooling). By using one function to perform multiple operations, duplicate coding and verification, as examples, are eliminated. The specific operation to be performed depends on the values of select input parameters to the function. The same input parameters are used for both operations of the function but different values of select parameters indicate the operation to be performed leading to different bound checking of other input parameters. Although mean-reduce and pooling operations execute differently, they can be algorithmically reduced to a common algorithmic operation with the same input tensors and function-specific-parameters but having different relative constraints. The one difference is checking of the conditions (e.g., stride, window size) that differ between the two operations. As described herein, a single function of an instruction is provided that executes multiple operations and implements different bound checking for both operations on some of the parameters. This reduces, at the very least, code complexity, code duplication and verification efforts.

[00300] One or more aspects of the present invention are inextricably tied to computer technology and facilitate processing within a computer, improving performance thereof. The use of a single architected machine instruction configured to perform various functions improves performance within the computing environment by reducing complexity, reducing

use of resources and increasing processing speed. The use of a single function to implement multiple operations reduces complexity, use of resources, coding and/or verification efforts, and improves system performance. The instruction, function, and/or operations may be used in many technical fields, such as in computer processing, medical processing, engineering, automotive technologies, manufacturing, etc. By providing optimizations, these technical fields are improved by, e.g., reducing errors and/or execution time.

[00301] Further details of one embodiment of facilitating processing within a computing environment, as it relates to one or more aspects of the present invention, are described with reference to FIGS. 7A-7C.

[00302] Referring to FIG. 7A, an indication of a function to be executed is obtained, in which the function is one function of an instruction and configured to perform multiple operations 700. A determination is made of an operation of the multiple operations to be performed 702, and a set of function-specific parameters is validated using a set of values and a corresponding set of relationships 704. The set of values and corresponding set of relationships are based on the operation to be performed 706. As examples, one set of values and corresponding set of relationships are to be used for the operation to be performed 708, and another set of values and corresponding set of relationships are to be used for another operation of the multiple operations 710.

[00303] Using a single function, e.g., a single function of an architected instruction, to perform multiple operations but with per operation parameter validation, code complexity, code duplication and/or verification efforts are reduced, improving system performance.

[00304] In one example, the determining the operation to be performed includes checking one or more function-specific parameters against at least one specific value 720. Based on the one or more function-specific parameters having a first select relationship with respect to the at least one specific value, the operation is one operation 722, and based on the one or more function-specific parameters having a second select relationship with respect to the at least one specific value, the operation is another operation 724.

[00305] By using the same function-specific parameters but different relationships to determine the operation to be performed, code complexity and verification efforts are reduced.

[00306] As an example, the function includes an average pool function, the one or more function-specific parameters include one or more stride values, in which a stride value is an amount that a sliding window moves over an input tensor when computing one or more adjacent output tensor elements, the at least one specific value includes zero, the first select relationship includes equal and the operation is a mean-reduce operation based on the one or more stride values being equal to zero 726.

[00307] Further, in one example, referring to FIG. 7B, the second select relationship includes greater than and the operation is a pooling operation based on the one or more stride values being greater than zero 728.

[00308] As an example, the set of function-specific parameters includes one or more select-dimension window size values 730. A select-dimension window size value specifies a number of elements in the select dimension that a sliding window contains 732, and the sliding window is configured to move over an input tensor of the function to produce an output tensor 734.

[00309] In one example, the function includes an average pool function, the operation includes a mean-reduce operation, and the set of values and corresponding set of relationships to be used to validate the set of function-specific parameters includes: one value of one dimension of a select input tensor and corresponding relationship of equal, another value of another dimension of the select input tensor and corresponding relationship of equal, and a select value and corresponding relationship of less than or equal 740.

[00310] The validating includes, for instance, checking a value of a dimension-2-window size is equal to a value of a dimension-2 of a first input tensor, a value of a dimension-3-window size is equal to a value of a dimension-3 of the first input tensor, the value of the dimension-2-window size is less than or equal to the select value, and the value of the dimension-3-window size is less than or equal to the select value 746.

[00311] In one example, referring to FIG. 7C, the function includes an average pool function, the operation includes a pooling operation, and the set of values and corresponding set of relationships to be used to validate the set of function-specific parameters includes: one value of one dimension of a select input tensor and corresponding relationship of less than or equal, and another value of another dimension of the select input tensor and corresponding relationship of less than or equal 750.

[00312] The validating includes, for instance, checking that a value of a dimension-2-window size is less than or equal to a value of a dimension-2 of a first input tensor and that a value of a dimension-3-window size is less than or equal to a value of a dimension-3 of the first input tensor 756.

[00313] In one example, a determination is made as to whether a type of padding is set to a particular type, in which the type of padding indicates which elements of a window are to be used in computing the output and, for one or more embodiments, e.g., the pooling operation, based on the type of padding being set to the particular type, the validating is performed 760. Further, in one example, based on the type of padding not being set to the particular type, one or more checks are performed relating to one or more dimensions of an output tensor 762.

[00314] In one example, the determining the operation is based on at least one sliding window stride value of an input tensor 770, and the set of function-specific parameters includes at least one sliding window dimension of an input tensor 772.

[00315] Other variations and embodiments are possible.

[00316] Aspects of the present invention may be used by many types of computing environments. Another example of a computing environment to incorporate and use one or more aspects of the present invention is described with reference to FIG. 8A. As an example, the computing environment of FIG. 8A is based on the z/Architecture® instruction set architecture offered by International Business Machines Corporation, Armonk, New York. The z/Architecture instruction set architecture, however, is only one example architecture. Again, the computing environment may be based on other architectures, including, but not limited to, the Intel® x86 architectures, other architectures of International Business Machines Corporation, and/or architectures of other companies. Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

[00317] In one example, a computing environment 10 includes a central electronics complex (CEC) 11. Central electronics complex 11 includes a plurality of components, such as, for instance, a memory 12 (a.k.a., system memory, main memory, main storage, central storage, storage) coupled to one or more processors, such as one or more general-purpose processors (a.k.a., central processing units (CPUs) 13) and one or more special-

purpose processors (e.g., neural network processor 31), and to an input/output (I/O) subsystem 14.

[00318] As examples, the one or more special-purpose processors may be separate from the one or more general-purpose processors and/or at least one special-purpose processor may be embedded within at least one general-purpose processor. Other variations are also possible.

[00319] I/O subsystem 14 can be a part of the central electronics complex or separate therefrom. It directs the flow of information between main storage 12 and input/output control units 15 and input/output (I/O) devices 16 coupled to the central electronics complex.

[00320] Many types of I/O devices may be used. One particular type is a data storage device 17. Data storage device 17 can store one or more programs 18, one or more computer readable program instructions 19, and/or data, etc. The computer readable program instructions can be configured to carry out functions of embodiments of aspects of the invention.

[00321] Central electronics complex 11 can include and/or be coupled to removable/non-removable, volatile/non-volatile computer system storage media. For example, it can include and/or be coupled to a non-removable, non-volatile magnetic media (typically called a "hard drive"), a magnetic disk drive for reading from and writing to a removable, non-volatile magnetic disk (e.g., a "floppy disk"), and/or an optical disk drive for reading from or writing to a removable, non-volatile optical disk, such as a CD-ROM, DVD-ROM or other optical media. It should be understood that other hardware and/or software components could be used in conjunction with central electronics complex 11. Examples include, but are not limited to: microcode or millicode, device drivers, redundant processing units, external disk drive arrays, RAID systems, tape drives, and data archival storage systems, etc.

[00322] Further, central electronics complex 11 can be operational with numerous other general-purpose or special-purpose computing system environments or configurations. Examples of well-known computing systems, environments, and/or configurations that may be suitable for use with central electronics complex 11 include, but are not limited to, personal computer (PC) systems, server computer systems, thin clients, thick clients,

handheld or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputer systems, mainframe computer systems, and distributed cloud computing environments that include any of the above systems or devices, and the like.

[00323] Central electronics complex 11 provides in one or more embodiments logical partitioning and/or virtualization support. In one embodiment, as shown in FIG. 8B, memory 12 includes, for example, one or more logical partitions 20, a hypervisor 21 that manages the logical partitions, and processor firmware 22. One example of hypervisor 21 is the Processor Resource/System Manager (PR/SM™), offered by International Business Machines Corporation, Armonk, New York. PR/SM is a trademark or registered trademark of International Business Machines Corporation in at least one jurisdiction.

[00324] Each logical partition 20 is capable of functioning as a separate system. That is, each logical partition can be independently reset, run a guest operating system 23 such as the z/OS® operating system, offered by International Business Machines Corporation, Armonk, New York, or other control code 24, such as coupling facility control code (CFCC), and operate with different programs 25. An operating system or application program running in a logical partition appears to have access to a full and complete system, but in reality, only a portion of it is available. Although the z/OS operating system is offered as an example, other operating systems offered by International Business Machines Corporation and/or other companies may be used in accordance with one or more aspects of the present invention.

[00325] Memory 12 is coupled to, e.g., CPUs 13 (FIG. 8A), which are physical processor resources that can be allocated to the logical partitions. For instance, a logical partition 20 may include one or more logical processors, each of which represents all or a share of a physical processor resource 13 that can be dynamically allocated to the logical partition.

[00326] In yet a further embodiment, the central electronics complex provides virtual machine support (either with or without logical partitioning support). As shown in FIG. 8C, memory 12 of central electronics complex 11 includes, for example, one or more virtual machines 26, a virtual machine manager, such as a hypervisor 27, that manages the virtual machines, and processor firmware 28. One example of hypervisor 27 is the z/VM® hypervisor, offered by International Business Machines Corporation, Armonk, New York.

The hypervisor is sometimes referred to as a host. z/VM is a trademark or registered trademark of International Business Machines Corporation in at least one jurisdiction.

[00327] The virtual machine support of the central electronics complex provides the ability to operate large numbers of virtual machines 26, each capable of operating with different programs 29 and running a guest operating system 30, such as the Linux® operating system. Each virtual machine 26 is capable of functioning as a separate system. That is, each virtual machine can be independently reset, run a guest operating system, and operate with different programs. An operating system or application program running in a virtual machine appears to have access to a full and complete system, but in reality, only a portion of it is available. Although z/VM and Linux are offered as examples, other virtual machine managers and/or operating systems may be used in accordance with one or more aspects of the present invention. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

[00328] Another embodiment of a computing environment to incorporate and use one or more aspects of the present invention is described with reference to FIG. 9A. In this example, a computing environment 36 includes, for instance, a native central processing unit (CPU) 37, a memory 38, and one or more input/output devices and/or interfaces 39 coupled to one another via, for example, one or more buses 40 and/or other connections. As examples, computing environment 36 may include a PowerPC® processor offered by International Business Machines Corporation, Armonk, New York; an HP Superdome with Intel® Itanium® II processors offered by Hewlett Packard Co., Palo Alto, California; and/or other machines based on architectures offered by International Business Machines Corporation, Hewlett Packard, Intel Corporation, Oracle, and/or others. PowerPC is a trademark or registered trademark of International Business Machines Corporation in at least one jurisdiction. Itanium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

[00329] Native central processing unit 37 includes one or more native registers 41, such as one or more general purpose registers and/or one or more special purpose registers used during processing within the environment. These registers include information that represents the state of the environment at any particular point in time.

[00330] Moreover, native central processing unit 37 executes instructions and code that are stored in memory 38. In one particular example, the central processing unit executes emulator code 42 stored in memory 38. This code enables the computing environment configured in one architecture to emulate another architecture. For instance, emulator code 42 allows machines based on architectures other than the z/Architecture instruction set architecture, such as PowerPC processors, HP Superdome servers or others, to emulate the z/Architecture instruction set architecture and to execute software and instructions developed based on the z/Architecture instruction set architecture.

[00331] Further details relating to emulator code 42 are described with reference to FIG. 9B. Guest instructions 43 stored in memory 38 comprise software instructions (e.g., correlating to machine instructions) that were developed to be executed in an architecture other than that of native CPU 37. For example, guest instructions 43 may have been designed to execute on a processor based on the z/Architecture instruction set architecture, but instead, are being emulated on native CPU 37, which may be, for example, an Intel Itanium II processor. In one example, emulator code 42 includes an instruction fetching routine 44 to obtain one or more guest instructions 43 from memory 38, and to optionally provide local buffering for the instructions obtained. It also includes an instruction translation routine 45 to determine the type of guest instruction that has been obtained and to translate the guest instruction into one or more corresponding native instructions 46. This translation includes, for instance, identifying the function to be performed by the guest instruction and choosing the native instruction(s) to perform that function.

[00332] Further, emulator code 42 includes an emulation control routine 47 to cause the native instructions to be executed. Emulation control routine 47 may cause native CPU 37 to execute a routine of native instructions that emulate one or more previously obtained guest instructions and, at the conclusion of such execution, return control to the instruction fetch routine to emulate the obtaining of the next guest instruction or a group of guest instructions. Execution of the native instructions 46 may include loading data into a register from memory 38; storing data back to memory from a register; or performing some type of arithmetic or logic operation, as determined by the translation routine.

[00333] Each routine is, for instance, implemented in software, which is stored in memory and executed by native central processing unit 37. In other examples, one or more of the routines or operations are implemented in firmware, hardware, software or some

combination thereof. The registers of the emulated processor may be emulated using registers 41 of the native CPU or by using locations in memory 38. In embodiments, guest instructions 43, native instructions 46 and emulator code 42 may reside in the same memory or may be disbursed among different memory devices.

[00334] An instruction that may be emulated includes the Neural Network Assist Processing instruction described herein, in accordance with one or more aspects of the present invention. Further, other instructions, functions, operations and/or one or more aspects of neural network processing may be emulated, in accordance with one or more aspects of the present invention.

[00335] The computing environments described above are only examples of computing environments that can be used. Other environments, including but not limited to, non-partitioned environments, partitioned environments, cloud environments and/or emulated environments, may be used; embodiments are not limited to any one environment. Although various examples of computing environments are described herein, one or more aspects of the present invention may be used with many types of environments. The computing environments provided herein are only examples.

[00336] Each computing environment is capable of being configured to include one or more aspects of the present invention.

[00337] One or more aspects may relate to cloud computing.

[00338] It is to be understood that although this disclosure includes a detailed description on cloud computing, implementation of the teachings recited herein are not limited to a cloud computing environment. Rather, embodiments of the present invention are capable of being implemented in conjunction with any other type of computing environment now known or later developed.

[00339] Cloud computing is a model of service delivery for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, network bandwidth, servers, processing, memory, storage, applications, virtual machines, and services) that can be rapidly provisioned and released with minimal management effort or interaction with a provider of the service. This cloud model may

include at least five characteristics, at least three service models, and at least four deployment models.

[00340] Characteristics are as follows:

[00341] On-demand self-service: a cloud consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with the service's provider.

[00342] Broad network access: capabilities are available over a network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

[00343] Resource pooling: the provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to demand. There is a sense of location independence in that the consumer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter).

[00344] Rapid elasticity: capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

[00345] Measured service: cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

[00346] Service Models are as follows:

[00347] Software as a Service (SaaS): the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based e-mail). The consumer does not manage or control the underlying cloud

infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

[00348] Platform as a Service (PaaS): the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

[00349] Infrastructure as a Service (IaaS): the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

[00350] Deployment Models are as follows:

[00351] Private cloud: the cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on-premises or off-premises.

[00352] Community cloud: the cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on-premises or off-premises.

[00353] Public cloud: the cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

[00354] Hybrid cloud: the cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

[00355] A cloud computing environment is service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability. At the heart of cloud computing is an infrastructure that includes a network of interconnected nodes.

[00356] Referring now to FIG. 10, illustrative cloud computing environment 50 is depicted. As shown, cloud computing environment 50 includes one or more cloud computing nodes 52 with which local computing devices used by cloud consumers, such as, for example, personal digital assistant (PDA) or cellular telephone 54A, desktop computer 54B, laptop computer 54C, and/or automobile computer system 54N may communicate. Nodes 52 may communicate with one another. They may be grouped (not shown) physically or virtually, in one or more networks, such as Private, Community, Public, or Hybrid clouds as described hereinabove, or a combination thereof. This allows cloud computing environment 50 to offer infrastructure, platforms and/or software as services for which a cloud consumer does not need to maintain resources on a local computing device. It is understood that the types of computing devices 54A-N shown in FIG. 10 are intended to be illustrative only and that computing nodes 52 and cloud computing environment 50 can communicate with any type of computerized device over any type of network and/or network addressable connection (e.g., using a web browser).

[00357] Referring now to FIG. 11, a set of functional abstraction layers provided by cloud computing environment 50 (FIG. 10) is shown. It should be understood in advance that the components, layers, and functions shown in FIG. 11 are intended to be illustrative only and embodiments of the invention are not limited thereto. As depicted, the following layers and corresponding functions are provided:

[00358] Hardware and software layer 60 includes hardware and software components. Examples of hardware components include: mainframes 61; RISC (Reduced Instruction Set Computer) architecture based servers 62; servers 63; blade servers 64; storage devices 65; and networks and networking components 66. In some embodiments, software components include network application server software 67 and database software 68.

[00359] Virtualization layer 70 provides an abstraction layer from which the following examples of virtual entities may be provided: virtual servers 71; virtual storage 72; virtual networks 73, including virtual private networks; virtual applications and operating systems 74; and virtual clients 75.

[00360] In one example, management layer 80 may provide the functions described below. Resource provisioning 81 provides dynamic procurement of computing resources and other resources that are utilized to perform tasks within the cloud computing environment. Metering and Pricing 82 provide cost tracking as resources are utilized within the cloud computing environment, and billing or invoicing for consumption of these resources. In one example, these resources may include application software licenses. Security provides identity verification for cloud consumers and tasks, as well as protection for data and other resources. User portal 83 provides access to the cloud computing environment for consumers and system administrators. Service level management 84 provides cloud computing resource allocation and management such that required service levels are met. Service Level Agreement (SLA) planning and fulfillment 85 provide pre-arrangement for, and procurement of, cloud computing resources for which a future requirement is anticipated in accordance with an SLA.

[00361] Workloads layer 90 provides examples of functionality for which the cloud computing environment may be utilized. Examples of workloads and functions which may be provided from this layer include: mapping and navigation 91; software development and lifecycle management 92; virtual classroom education delivery 93; data analytics processing 94; transaction processing 95; and neural network processing assist processing 96.

[00362] Aspects of the present invention may be a system, a method, and/or a computer program product at any possible technical detail level of integration. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[00363] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only

memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[00364] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[00365] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, configuration data for integrated circuitry, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++, or the like, and procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-

programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[00366] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[00367] These computer readable program instructions may be provided to a processor of a computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[00368] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[00369] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the

specified logical function(s). In some alternative implementations, the functions noted in the blocks may occur out of the order noted in the Figures. For example, two blocks shown in succession may, in fact, be accomplished as one step, executed concurrently, substantially concurrently, in a partially or wholly temporally overlapping manner, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

[00370] In addition to the above, one or more aspects may be provided, offered, deployed, managed, serviced, etc. by a service provider who offers management of customer environments. For instance, the service provider can create, maintain, support, etc. computer code and/or a computer infrastructure that performs one or more aspects for one or more customers. In return, the service provider may receive payment from the customer under a subscription and/or fee agreement, as examples. Additionally or alternatively, the service provider may receive payment from the sale of advertising content to one or more third parties.

[00371] In one aspect, an application may be deployed for performing one or more embodiments. As one example, the deploying of an application comprises providing computer infrastructure operable to perform one or more embodiments.

[00372] As a further aspect, a computing infrastructure may be deployed comprising integrating computer readable code into a computing system, in which the code in combination with the computing system is capable of performing one or more embodiments.

[00373] As yet a further aspect, a process for integrating computing infrastructure comprising integrating computer readable code into a computer system may be provided. The computer system comprises a computer readable medium, in which the computer medium comprises one or more embodiments. The code in combination with the computer system is capable of performing one or more embodiments.

[00374] Although various embodiments are described above, these are only examples. For instance, computing environments of other architectures can be used to incorporate

and/or use one or more aspects. Further, different instructions, functions and/or operations may be used. Additionally, different types of registers and/or different registers may be used. Further, other data formats, data layouts and/or data sizes may be supported. In one or more embodiments, one or more general-purpose processors, one or more special-purpose processors or a combination of general-purpose and special-purpose processors may be used. Many variations are possible.

[00375] Various aspects are described herein. Further, many variations are possible without departing from a spirit of aspects of the present invention. It should be noted that, unless otherwise inconsistent, each aspect or feature described herein, and variants thereof, may be combinable with any other aspect or feature.

[00376] Further, other types of computing environments can benefit and be used. As an example, a data processing system suitable for storing and/or executing program code is usable that includes at least two processors coupled directly or indirectly to memory elements through a system bus. The memory elements include, for instance, local memory employed during actual execution of the program code, bulk storage, and cache memory which provide temporary storage of at least some program code in order to reduce the number of times code must be retrieved from bulk storage during execution.

[00377] Input/Output or I/O devices (including, but not limited to, keyboards, displays, pointing devices, DASD, tape, CDs, DVDs, thumb drives and other memory media, etc.) can be coupled to the system either directly or through intervening I/O controllers. Network adapters may also be coupled to the system to enable the data processing system to become coupled to other data processing systems or remote printers or storage devices through intervening private or public networks. Modems, cable modems, and Ethernet cards are just a few of the available types of network adapters.

[00378] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting. As used herein, the singular forms "a", "an" and "the" are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms "comprises" and/or "comprising", when used in this specification, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components and/or groups thereof.

[00379] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below, if any, are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of one or more embodiments has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain various aspects and the practical application, and to enable others of ordinary skill in the art to understand various embodiments with various modifications as are suited to the particular use contemplated.

CLAIMS

1. A computer program product for facilitating processing within a computing environment, the computer program product comprising:

one or more computer readable storage media and program instructions collectively stored on the one or more computer readable storage media to perform a method comprising:

obtaining an indication of a function to be executed, the function being one function of an instruction and configured to perform multiple operations;

determining an operation of the multiple operations to be performed; and

validating a set of function-specific parameters using a set of values and a corresponding set of relationships, wherein the set of values and corresponding set of relationships are based on the operation to be performed, wherein one set of values and corresponding set of relationships are to be used for the operation to be performed and another set of values and corresponding set of relationships are to be used for another operation of the multiple operations.

2. The computer program product of claim 1, wherein the determining the operation to be performed comprises checking one or more function-specific parameters against at least one specific value, wherein based on the one or more function-specific parameters having a first select relationship with respect to the at least one specific value, the operation is one operation, and based on the one or more function-specific parameters having a second select relationship with respect to the at least one specific value, the operation is another operation.

3. The computer program product of claim 2, wherein the function comprises an average pool function, the one or more function-specific parameters comprise one or more stride values, in which a stride value is an amount that a sliding window moves over an input tensor when computing one or more adjacent output tensor elements, the at least one specific value comprises zero, the first select relationship comprises equal and the

operation is a mean-reduce operation based on the one or more stride values being equal to zero.

4. The computer program product of claim 3, wherein the second select relationship comprises greater than and the operation is a pooling operation based on the one or more stride values being greater than zero.

5. The computer program product according to any of the previous claims , wherein the set of function-specific parameters comprises one or more select-dimension window size values, and wherein a select-dimension window size value specifies a number of elements in the select dimension that a sliding window contains, the sliding window configured to move over an input tensor of the function to produce an output tensor.

6. The computer program product of claim 5, wherein the function comprises an average pool function, the operation comprises a mean-reduce operation, and the set of values and corresponding set of relationships to be used to validate the set of function-specific parameters includes: one value of one dimension of a select input tensor and corresponding relationship of equal, another value of another dimension of the select input tensor and corresponding relationship of equal, and a select value and corresponding relationship of less than or equal.

7. The computer program product of claim 6, wherein the validating comprises checking a value of a dimension-2-window size is equal to a value of a dimension-2 of a first input tensor, a value of a dimension-3-window size is equal to a value of a dimension-3 of the first input tensor, the value of the dimension-2-window size is less than or equal to the select value, and the value of the dimension-3-window size is less than or equal to the select value.

8. The computer program product of claim 5, wherein the function comprises an average pool function, the operation comprises a pooling operation, and the set of values and corresponding set of relationships to be used to validate the set of function-specific parameters includes: one value of one dimension of a select input tensor and corresponding relationship of less than or equal, and another value of another dimension of the select input tensor and corresponding relationship of less than or equal.

9. The computer program product of claim 8, wherein the validating comprises checking that a value of a dimension-2-window size is less than or equal to a value of a

dimension-2 of a first input tensor and that a value of a dimension-3-window size is less than or equal to a value of a dimension-3 of the first input tensor.

10. The computer program product of claim 8, wherein the method further comprises determining whether a type of padding is set to a particular type, wherein the type of padding indicates which elements of a window are to be used in computing the output, and based on the type of padding being set to the particular type, the validating is performed.

11. The computer program product of claim 10, wherein based on the type of padding not being set to the particular type, one or more checks are performed relating to one or more dimensions of an output tensor.

12. The computer program product according to any of the previous claims , wherein the determining the operation is based on at least one sliding window stride value of an input tensor, and wherein the set of function-specific parameters includes at least one sliding window dimension of an input tensor.

13. A computer system for facilitating processing within a computing environment, the computer system comprising:

a memory; and

at least one processor in communication with the memory, wherein the computer system is configured to perform a method, said method comprising:

obtaining an indication of a function to be executed, the function being one function of an instruction and configured to perform multiple operations;

determining an operation of the multiple operations to be performed; and

validating a set of function-specific parameters using a set of values and a corresponding set of relationships, wherein the set of values and corresponding set of relationships are based on the operation to be performed, wherein one set of values and corresponding set of relationships are to be used for the operation to be performed and another set of values and

corresponding set of relationships are to be used for another operation of the multiple operations.

14. The computer system of claim 13, wherein the determining the operation to be performed comprises checking one or more function-specific parameters against at least one specific value, wherein based on the one or more function-specific parameters having a first select relationship with respect to the at least one specific value, the operation is one operation, and based on the one or more function-specific parameters having a second select relationship with respect to the at least one specific value, the operation is another operation.

15. The computer system of claim 14, wherein the function comprises an average pool function, the one or more function-specific parameters comprise one or more stride values, in which a stride value is an amount that a sliding window moves over an input tensor when computing one or more adjacent output tensor elements, the at least one specific value comprises zero, the first select relationship comprises equal and the operation is a mean-reduce operation based on the one or more stride values being equal to zero, and wherein the second select relationship comprises greater than and the operation is a pooling operation based on the one or more stride values being greater than zero.

16. The computer system according to any of the previous claims 13 to 15 , wherein the determining the operation is based on at least one sliding window stride value of an input tensor, and wherein the set of function-specific parameters includes at least one sliding window dimension of an input tensor.

17. A computer-implemented method of facilitating processing within a computing environment, the computer-implemented method comprising:

obtaining an indication of a function to be executed, the function being one function of an instruction and configured to perform multiple operations;

determining an operation of the multiple operations to be performed; and

validating a set of function-specific parameters using a set of values and a corresponding set of relationships, wherein the set of values and corresponding set of relationships are based on the operation to be performed, wherein one set of values and corresponding set of relationships are to be used for the operation to be performed and another set of values and corresponding set of relationships are to be used for another operation of the multiple operations.

18. The computer-implemented method of claim 17, wherein the determining the operation to be performed comprises checking one or more function-specific parameters against at least one specific value, wherein based on the one or more function-specific parameters having a first select relationship with respect to the at least one specific value, the operation is one operation, and based on the one or more function-specific parameters having a second select relationship with respect to the at least one specific value, the operation is another operation.

19. The computer-implemented method of claim 18, wherein the function comprises an average pool function, the one or more function-specific parameters comprise one or more stride values, in which a stride value is an amount that a sliding window moves over an input tensor when computing one or more adjacent output tensor elements, the at least one specific value comprises zero, the first select relationship comprises equal and the operation is a mean-reduce operation based on the one or more stride values being equal to zero, and wherein the second select relationship comprises greater than and the operation is a pooling operation based on the one or more stride values being greater than zero.

20. The computer-implemented method according to any of the previous claims 17 to 19 , wherein the determining the operation is based on at least one sliding window stride value of an input tensor, and wherein the set of function-specific parameters includes at least one sliding window dimension of an input tensor.

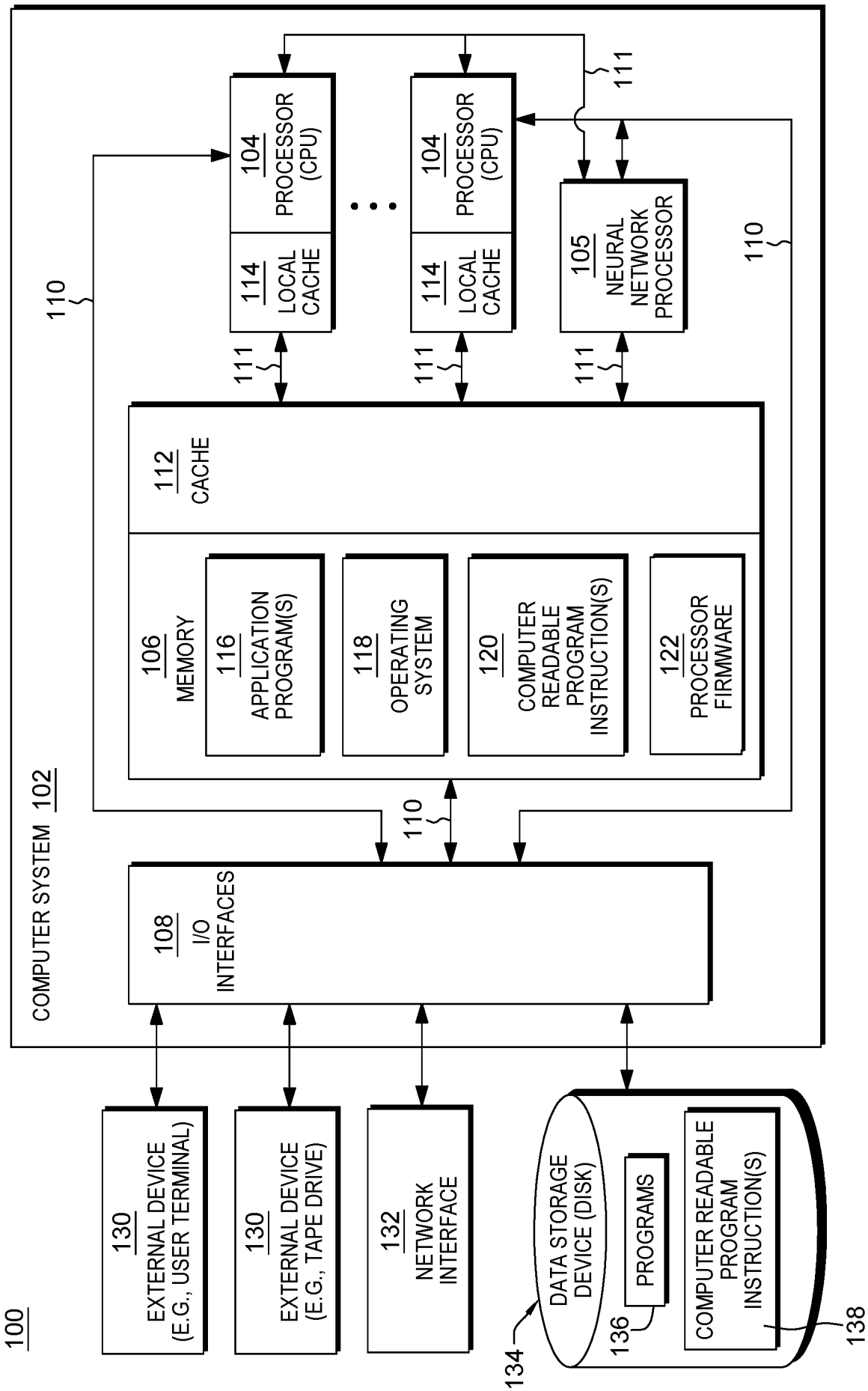

100
COMPUTER SYSTEM 102
104 PROCESSOR (CPU)
114 LOCAL CACHE
104 PROCESSOR (CPU)
114 LOCAL CACHE
105 NEURAL NETWORK PROCESSOR
111
110
112 CACHE
106 MEMORY
116 APPLICATION PROGRAM(S)
118 OPERATING SYSTEM
120 COMPUTER READABLE PROGRAM INSTRUCTION(S)
122 PROCESSOR FIRMWARE
108 I/O INTERFACES
130 EXTERNAL DEVICE (E.G., USER TERMINAL)
130 EXTERNAL DEVICE (E.G., TAPE DRIVE)
132 NETWORK INTERFACE
DATA STORAGE DEVICE (DISK)
PROGRAMS
COMPUTER READABLE PROGRAM INSTRUCTION(S)
134
136
138

FIG. 1A

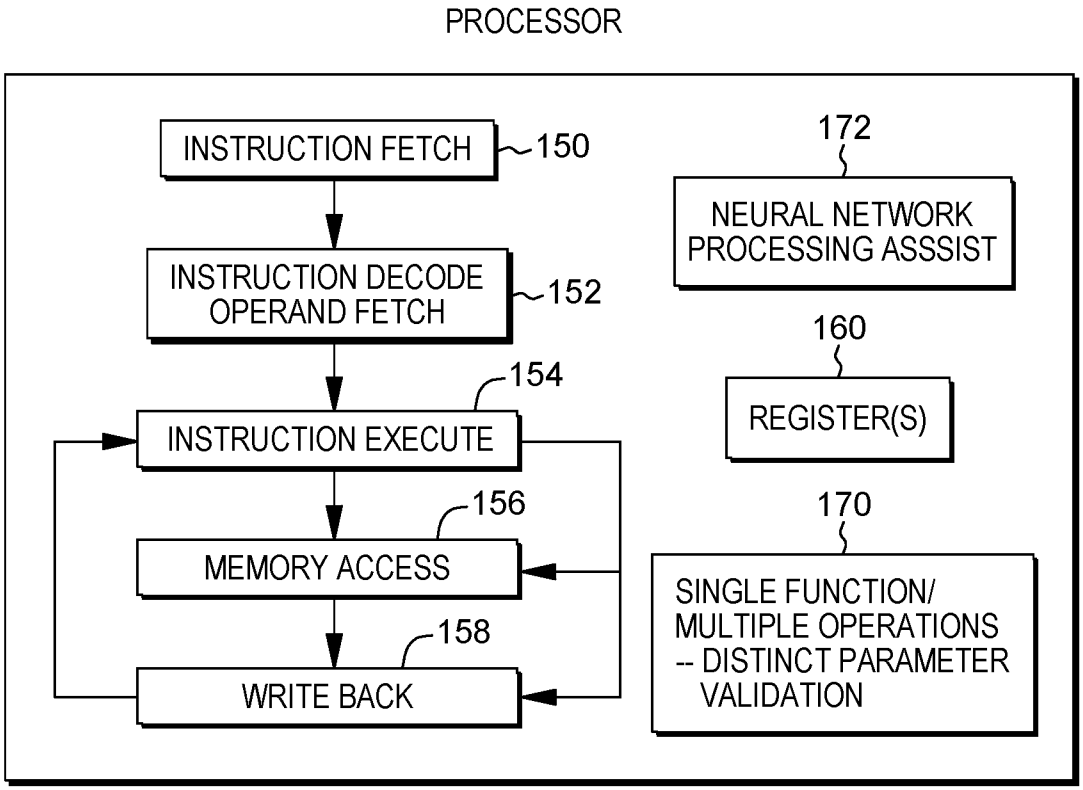

PROCESSOR
INSTRUCTION FETCH
150
INSTRUCTION DECODE
OPERAND FETCH
152
154
INSTRUCTION EXECUTE
156
MEMORY ACCESS
158
WRITE BACK
172
NEURAL NETWORK
PROCESSING ASSSIST
160
REGISTER(S)
170
SINGLE FUNCTION/
MULTIPLE OPERATIONS
-- DISTINCT PARAMETER
VALIDATION

FIG. 1B

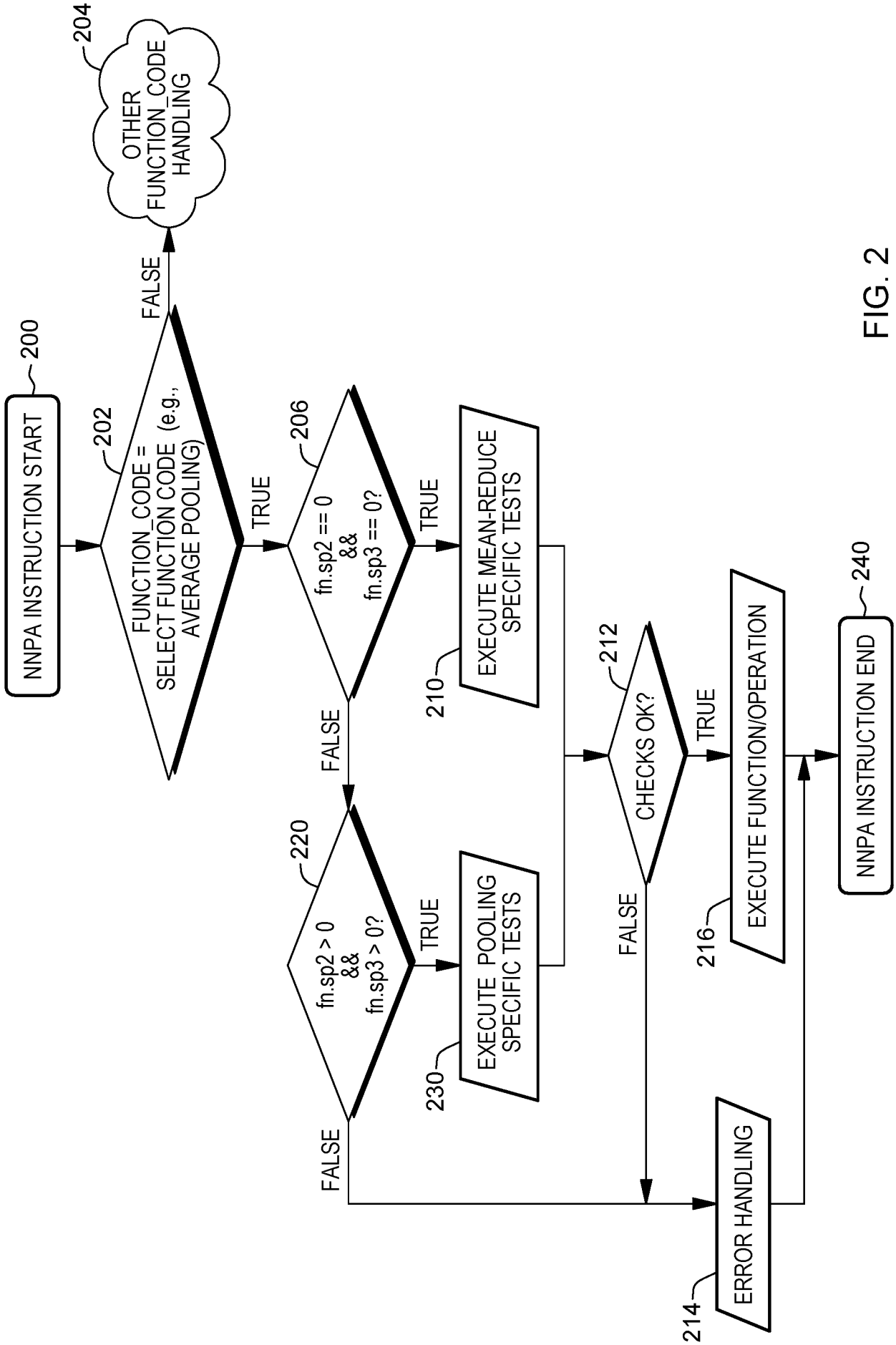

NNPA INSTRUCTION START
200
FUNCTION_CODE = SELECT FUNCTION CODE (e.g., AVERAGE POOLING)
202
FALSE
OTHER FUNCTION_CODE HANDLING
204
TRUE
fn.sp2 == 0 && fn.sp3 == 0?
206
TRUE
EXECUTE MEAN-REDUCE SPECIFIC TESTS
210
FALSE
fn.sp2 > 0 && fn.sp3 > 0?
220
TRUE
EXECUTE POOLING SPECIFIC TESTS
230
FALSE
CHECKS OK?
212
TRUE
EXECUTE FUNCTION/OPERATION
216
FALSE
ERROR HANDLING
214
NNPA INSTRUCTION END
240

FIG. 2

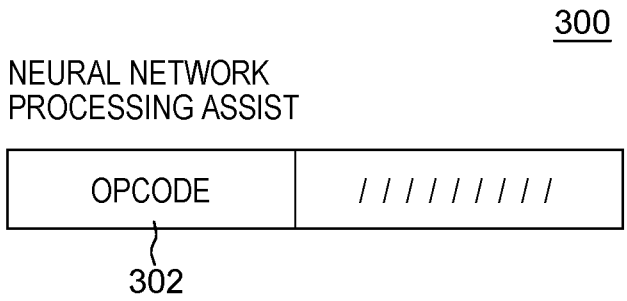

300
NEURAL NETWORK
PROCESSING ASSIST
OPCODE
/ / / / / / / / /
302

FIG. 3A

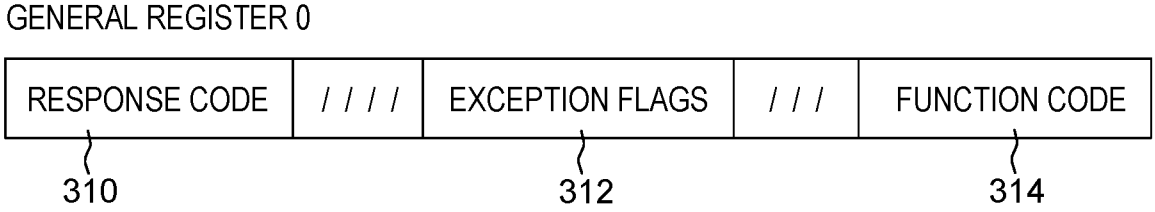

GENERAL REGISTER 0
RESPONSE CODE
/ / / /
EXCEPTION FLAGS
/ / /
FUNCTION CODE
310
312
314

FIG. 3B

314

CODE (DEC)	CODE (HEX)	FUNCTION	PARAMETER BLOCK SIZE (BYTES)
0	0	NNPA - QAF	256
16	10	NNPA - ADD	4096
17	11	NNPA - SUB	4096
18	12	NNPA - MUL	4096
19	13	NNPA - DIV	4096
20	14	NNPA - MIN	4096
21	15	NNPA - MAX	4096
32	20	NNPA - LOG	4096
33	21	NNPA - EXP	4096
49	31	NNPA - RELU	4096
50	32	NNPA - TANH	4096
51	33	NNPA - SIGMOID	4096
52	34	NNPA - SOFTMAX	4096
64	40	NNPA - BATCHNORM	4096
80	50	NNPA - MAXPOOL2D	4096
81	51	NNPA - AVGPOOL2D	4096
96	60	NNPA - LSTMACT	4096
97	61	NNPA - GRUACT	4096
112	70	NNPA - CONVOLUTION	4096
113	71	NNPA - MATMUL-OP	4096
114	72	NNPA - MATMUL-OP-BCAST23	4096

FIG. 3C

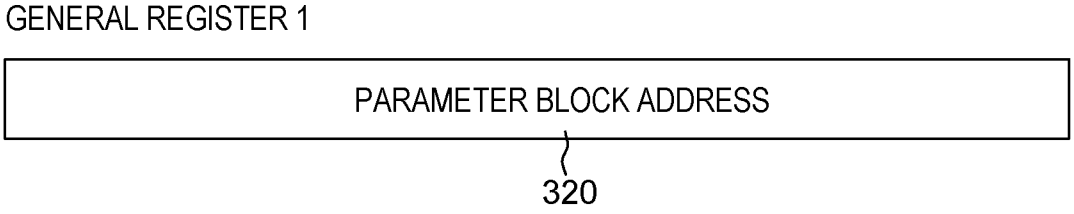

GENERAL REGISTER 1
PARAMETER BLOCK ADDRESS
320

FIG. 3D

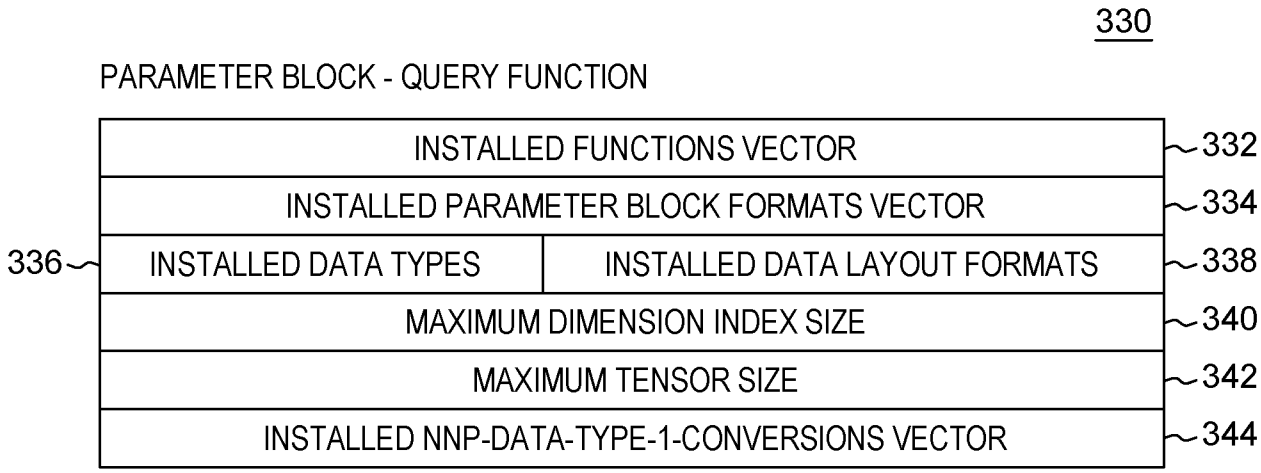

330
PARAMETER BLOCK - QUERY FUNCTION
INSTALLED FUNCTIONS VECTOR
332
INSTALLED PARAMETER BLOCK FORMATS VECTOR
334
336
INSTALLED DATA TYPES
INSTALLED DATA LAYOUT FORMATS
338
MAXIMUM DIMENSION INDEX SIZE
340
MAXIMUM TENSOR SIZE
342
INSTALLED NNP-DATA-TYPE-1-CONVERSIONS VECTOR
344

FIG. 3E

350

PARAMETER BLOCK - NON-QUERY FUNCTIONS

352	PARAMETER BLOCK VERSION NUMBER	MODEL VERSION NUMBER 354	CONTINUATION FLAG	356
	FUNCTION-SPECIFIC-SAVE-AREA-ADDRESS			358
	OUTPUT TENSOR DESCRIPTOR 1			360
	OUTPUT TENSOR DESCRIPTOR 2			
	INPUT TENSOR DESCRIPTOR 1			365
	INPUT TENSOR DESCRIPTOR 2			
	INPUT TENSOR DESCRIPTOR 3			
	FUNCTION-SPECIFIC-PARAMETER 1	FUNCTION-SPECIFIC-PARAMETER 2		370
	FUNCTION-SPECIFIC-PARAMETER 3	FUNCTION-SPECIFIC-PARAMETER 4		
	FUNCTION-SPECIFIC-PARAMETER 5			
	CONTINUATION STATE BUFFER			375

FIG. 3F

360, 365

TENSOR DESCRIPTOR

382	DATA LAYOUT FORMAT	DATA TYPE	384
	DIMENSION-4 INDEX SIZE	DIMENSION-3 INDEX SIZE	386
	DIMENSION-2 INDEX SIZE	DIMENSION-1 INDEX SIZE	
	TENSOR ADDRESS		388

FIG. 3G

400

NNP-DATA-TYPE-1

S	EXPONENT + 31	FRACTION
402	404	406

FIG. 4

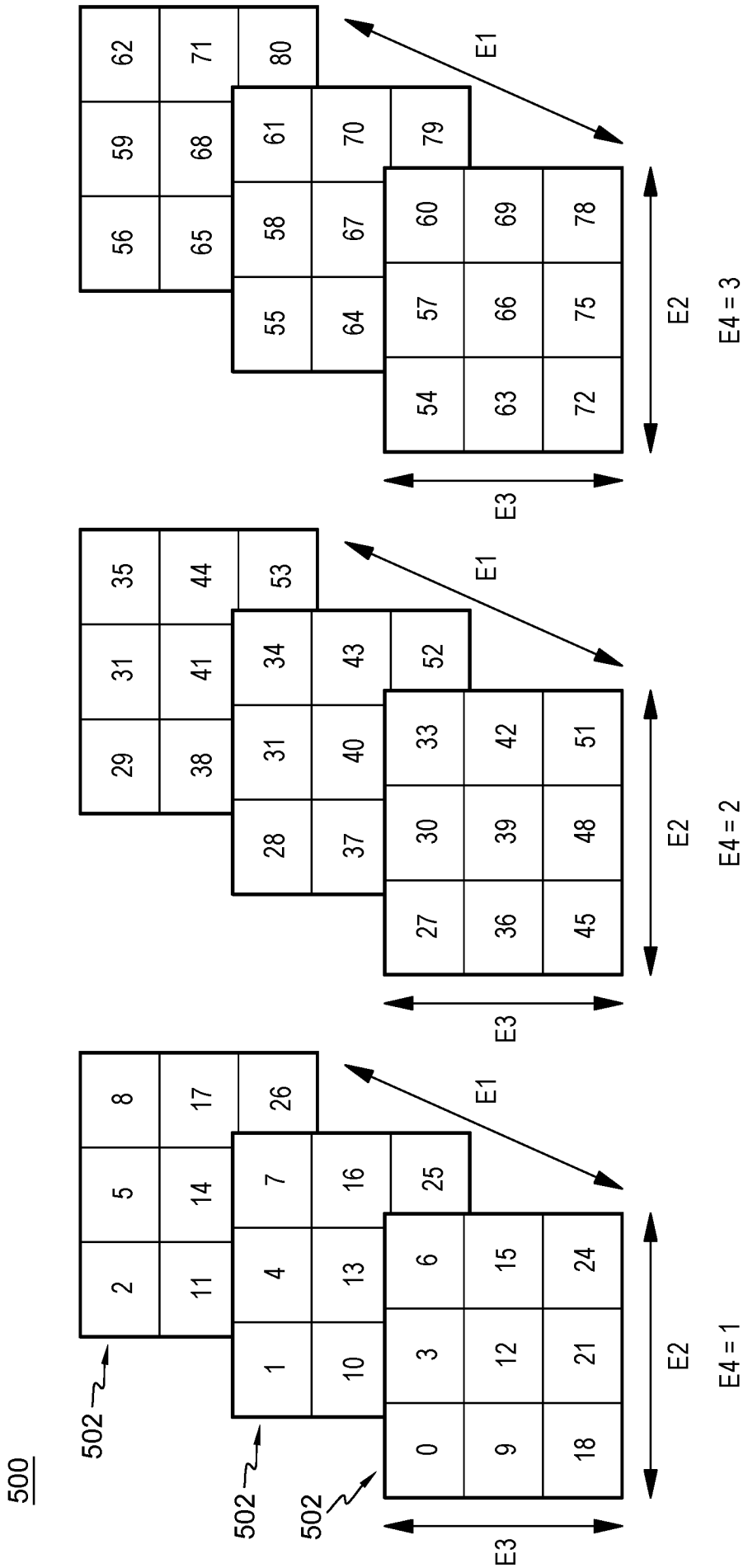


FIG. 5A

FIG. 5B

FIG. 5C

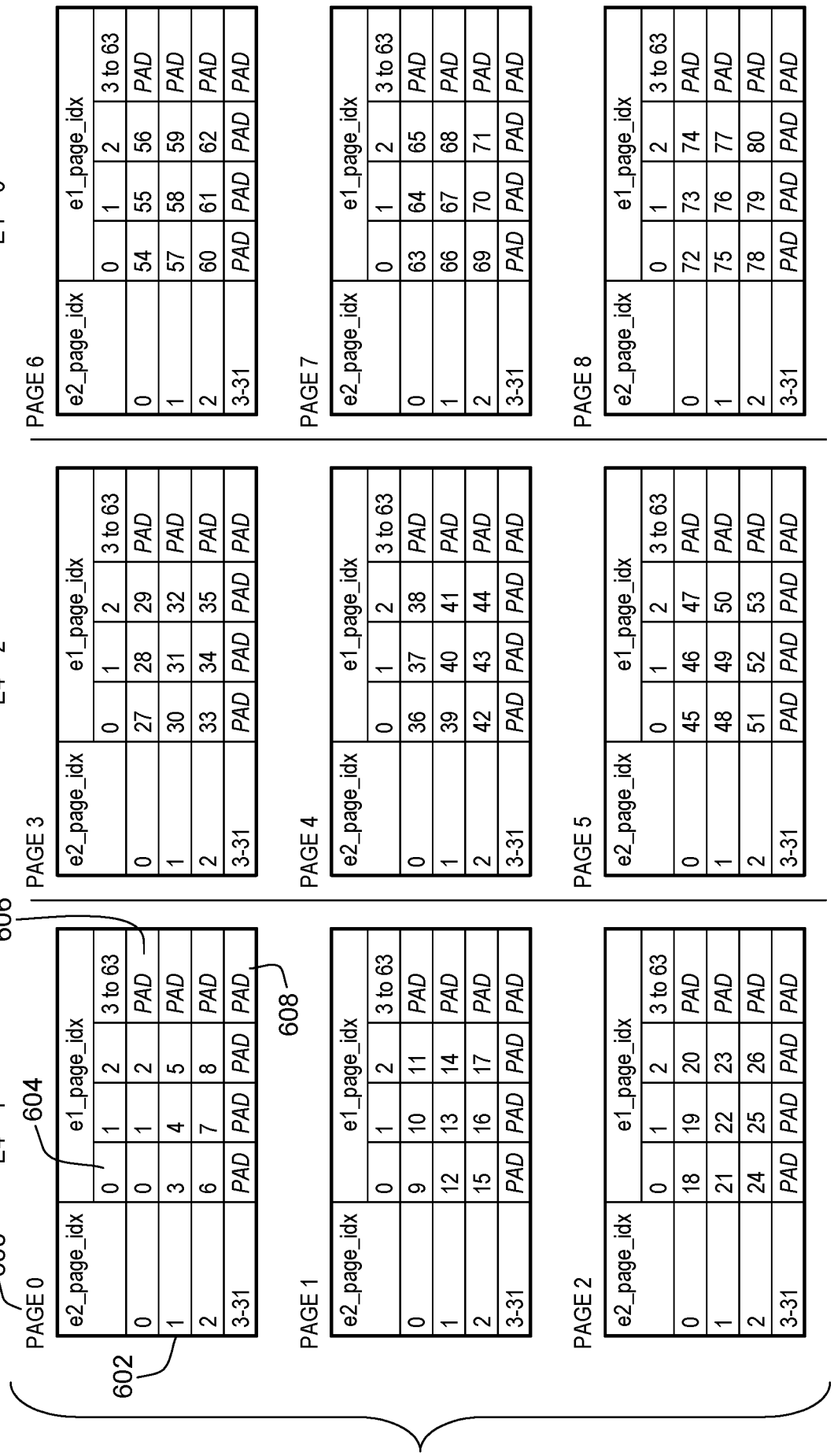


FIG. 6A

FIG. 6B

FIG. 6C

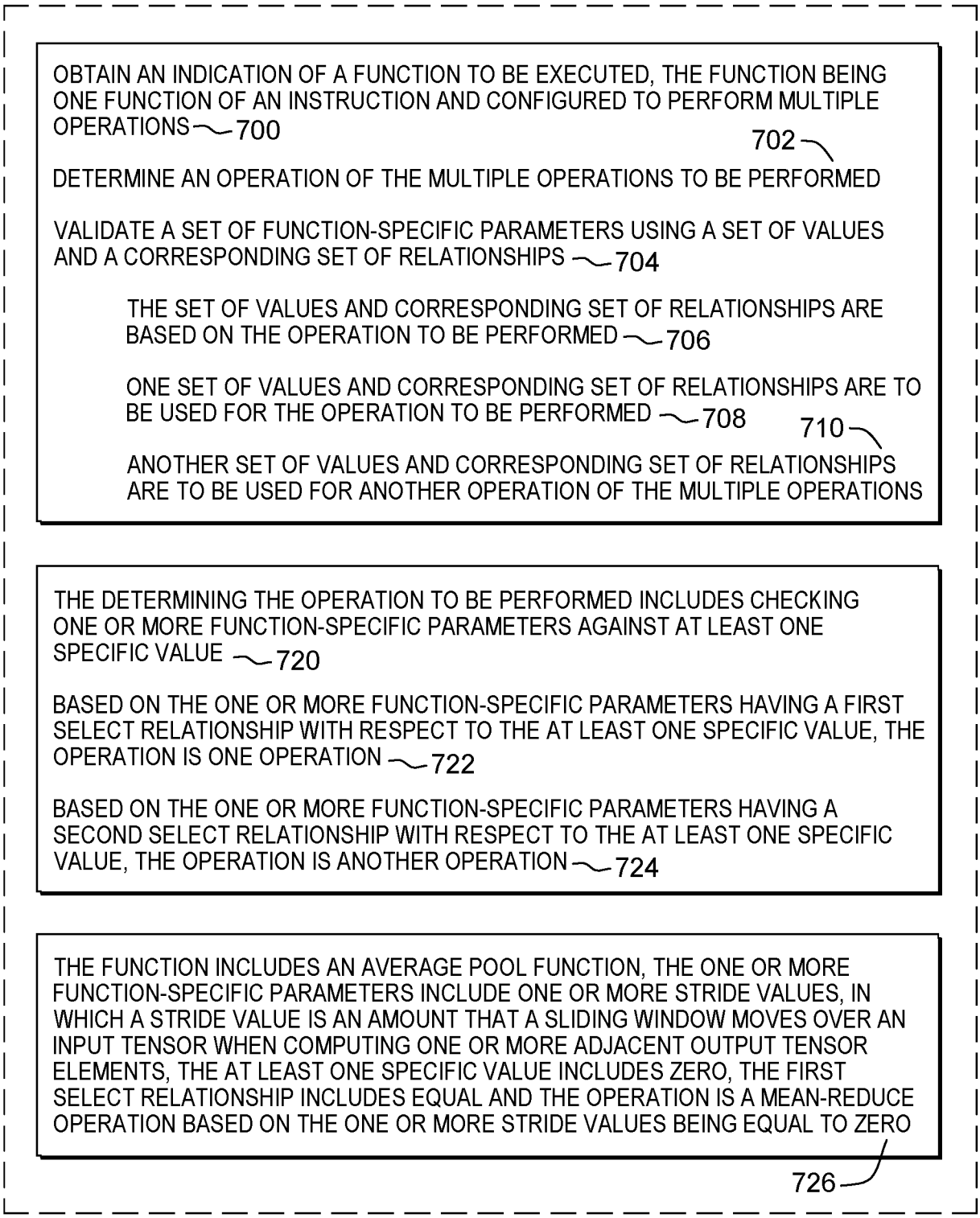

OBTAIN AN INDICATION OF A FUNCTION TO BE EXECUTED, THE FUNCTION BEING ONE FUNCTION OF AN INSTRUCTION AND CONFIGURED TO PERFORM MULTIPLE OPERATIONS 700
702
DETERMINE AN OPERATION OF THE MULTIPLE OPERATIONS TO BE PERFORMED
VALIDATE A SET OF FUNCTION-SPECIFIC PARAMETERS USING A SET OF VALUES AND A CORRESPONDING SET OF RELATIONSHIPS 704
THE SET OF VALUES AND CORRESPONDING SET OF RELATIONSHIPS ARE BASED ON THE OPERATION TO BE PERFORMED 706
ONE SET OF VALUES AND CORRESPONDING SET OF RELATIONSHIPS ARE TO BE USED FOR THE OPERATION TO BE PERFORMED 708
710
ANOTHER SET OF VALUES AND CORRESPONDING SET OF RELATIONSHIPS ARE TO BE USED FOR ANOTHER OPERATION OF THE MULTIPLE OPERATIONS
THE DETERMINING THE OPERATION TO BE PERFORMED INCLUDES CHECKING ONE OR MORE FUNCTION-SPECIFIC PARAMETERS AGAINST AT LEAST ONE SPECIFIC VALUE 720
BASED ON THE ONE OR MORE FUNCTION-SPECIFIC PARAMETERS HAVING A FIRST SELECT RELATIONSHIP WITH RESPECT TO THE AT LEAST ONE SPECIFIC VALUE, THE OPERATION IS ONE OPERATION 722
BASED ON THE ONE OR MORE FUNCTION-SPECIFIC PARAMETERS HAVING A SECOND SELECT RELATIONSHIP WITH RESPECT TO THE AT LEAST ONE SPECIFIC VALUE, THE OPERATION IS ANOTHER OPERATION 724
THE FUNCTION INCLUDES AN AVERAGE POOL FUNCTION, THE ONE OR MORE FUNCTION-SPECIFIC PARAMETERS INCLUDE ONE OR MORE STRIDE VALUES, IN WHICH A STRIDE VALUE IS AN AMOUNT THAT A SLIDING WINDOW MOVES OVER AN INPUT TENSOR WHEN COMPUTING ONE OR MORE ADJACENT OUTPUT TENSOR ELEMENTS, THE AT LEAST ONE SPECIFIC VALUE INCLUDES ZERO, THE FIRST SELECT RELATIONSHIP INCLUDES EQUAL AND THE OPERATION IS A MEAN-REDUCE OPERATION BASED ON THE ONE OR MORE STRIDE VALUES BEING EQUAL TO ZERO
726

FIG. 7A

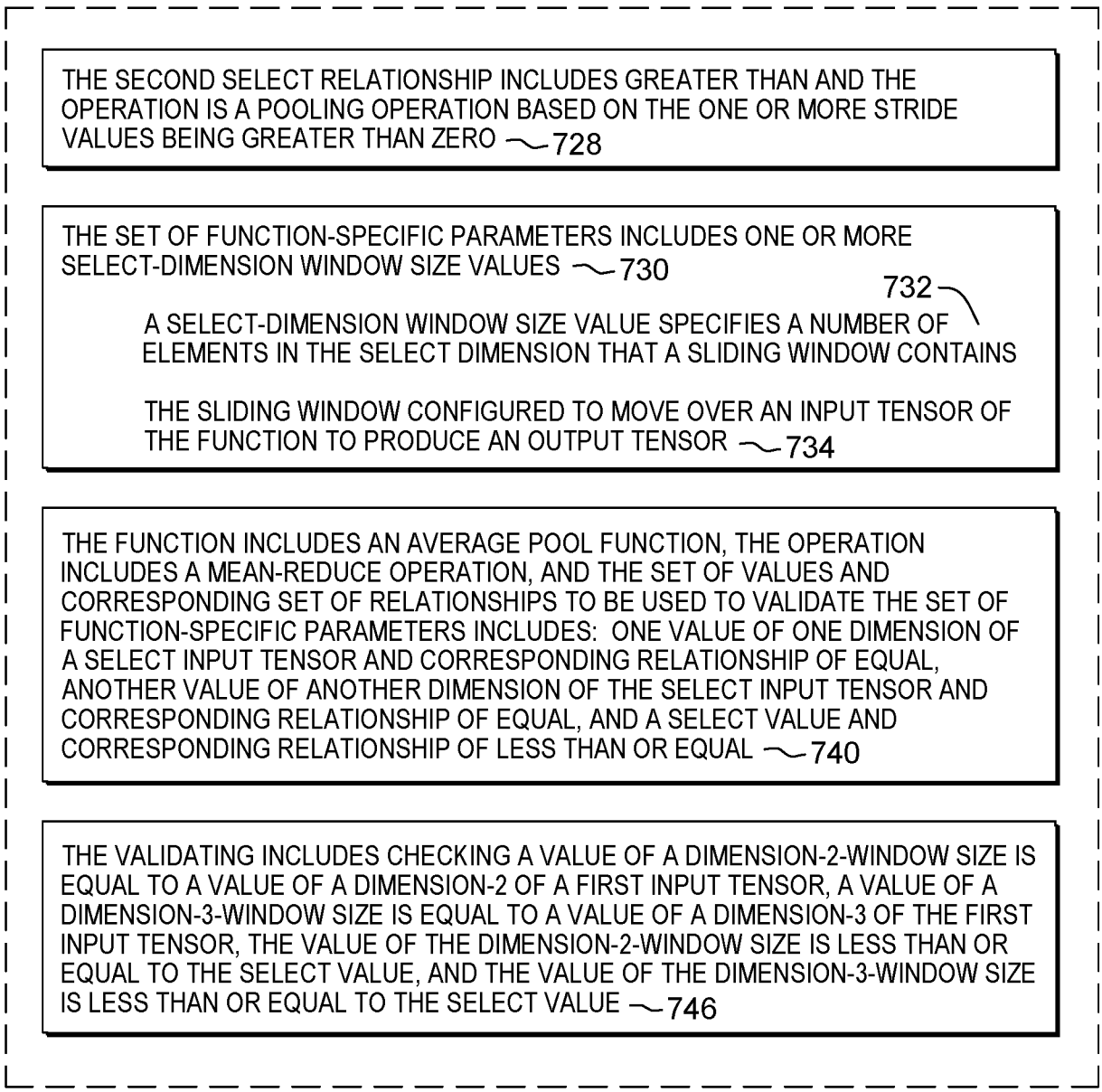

THE SECOND SELECT RELATIONSHIP INCLUDES GREATER THAN AND THE OPERATION IS A POOLING OPERATION BASED ON THE ONE OR MORE STRIDE VALUES BEING GREATER THAN ZERO 728
THE SET OF FUNCTION-SPECIFIC PARAMETERS INCLUDES ONE OR MORE SELECT-DIMENSION WINDOW SIZE VALUES 730
732
A SELECT-DIMENSION WINDOW SIZE VALUE SPECIFIES A NUMBER OF ELEMENTS IN THE SELECT DIMENSION THAT A SLIDING WINDOW CONTAINS
THE SLIDING WINDOW CONFIGURED TO MOVE OVER AN INPUT TENSOR OF THE FUNCTION TO PRODUCE AN OUTPUT TENSOR 734
THE FUNCTION INCLUDES AN AVERAGE POOL FUNCTION, THE OPERATION INCLUDES A MEAN-REDUCE OPERATION, AND THE SET OF VALUES AND CORRESPONDING SET OF RELATIONSHIPS TO BE USED TO VALIDATE THE SET OF FUNCTION-SPECIFIC PARAMETERS INCLUDES: ONE VALUE OF ONE DIMENSION OF A SELECT INPUT TENSOR AND CORRESPONDING RELATIONSHIP OF EQUAL, ANOTHER VALUE OF ANOTHER DIMENSION OF THE SELECT INPUT TENSOR AND CORRESPONDING RELATIONSHIP OF EQUAL, AND A SELECT VALUE AND CORRESPONDING RELATIONSHIP OF LESS THAN OR EQUAL 740
THE VALIDATING INCLUDES CHECKING A VALUE OF A DIMENSION-2-WINDOW SIZE IS EQUAL TO A VALUE OF A DIMENSION-2 OF A FIRST INPUT TENSOR, A VALUE OF A DIMENSION-3-WINDOW SIZE IS EQUAL TO A VALUE OF A DIMENSION-3 OF THE FIRST INPUT TENSOR, THE VALUE OF THE DIMENSION-2-WINDOW SIZE IS LESS THAN OR EQUAL TO THE SELECT VALUE, AND THE VALUE OF THE DIMENSION-3-WINDOW SIZE IS LESS THAN OR EQUAL TO THE SELECT VALUE 746

FIG. 7B

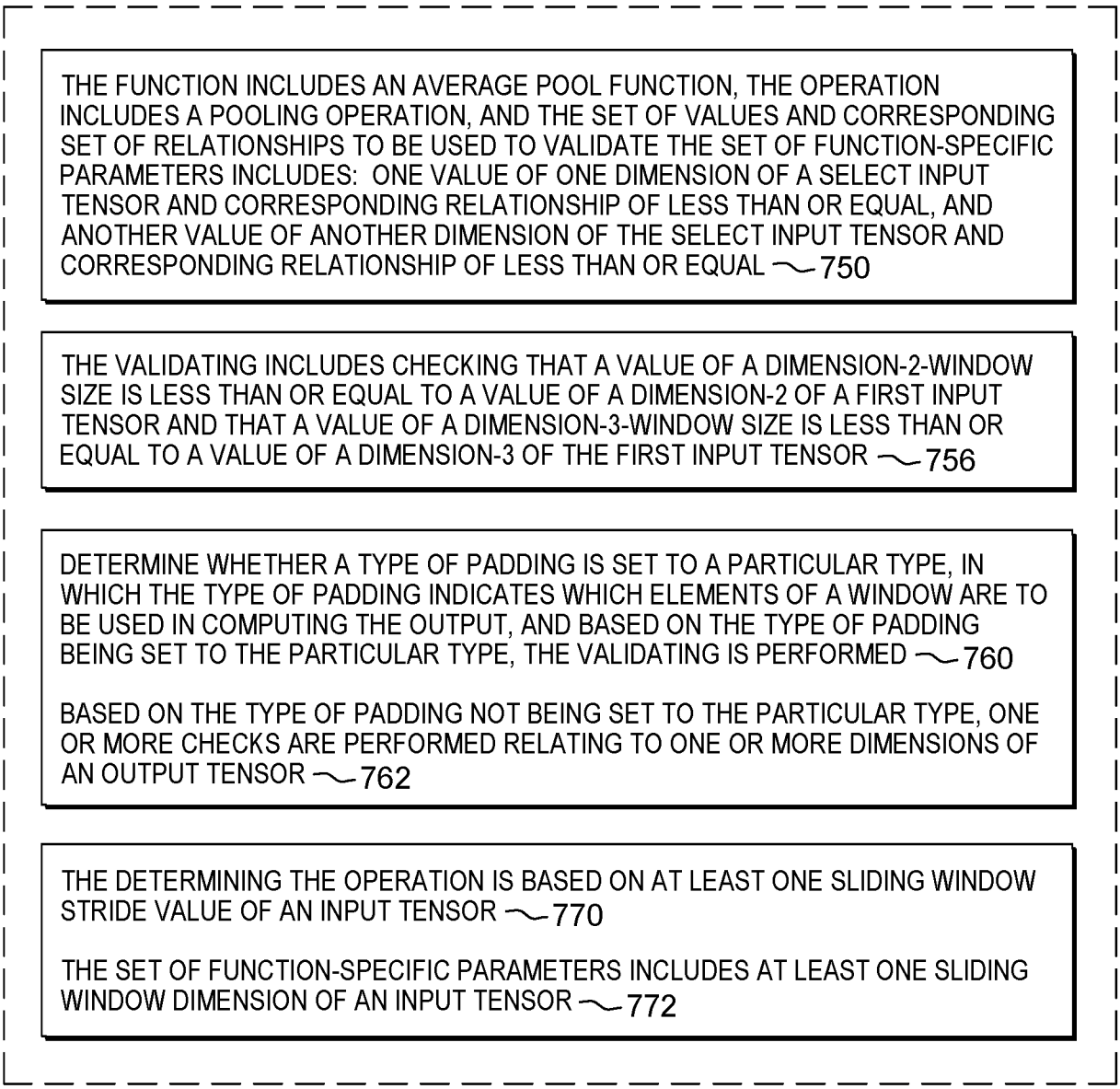

THE FUNCTION INCLUDES AN AVERAGE POOL FUNCTION, THE OPERATION INCLUDES A POOLING OPERATION, AND THE SET OF VALUES AND CORRESPONDING SET OF RELATIONSHIPS TO BE USED TO VALIDATE THE SET OF FUNCTION-SPECIFIC PARAMETERS INCLUDES: ONE VALUE OF ONE DIMENSION OF A SELECT INPUT TENSOR AND CORRESPONDING RELATIONSHIP OF LESS THAN OR EQUAL, AND ANOTHER VALUE OF ANOTHER DIMENSION OF THE SELECT INPUT TENSOR AND CORRESPONDING RELATIONSHIP OF LESS THAN OR EQUAL 750
THE VALIDATING INCLUDES CHECKING THAT A VALUE OF A DIMENSION-2-WINDOW SIZE IS LESS THAN OR EQUAL TO A VALUE OF A DIMENSION-2 OF A FIRST INPUT TENSOR AND THAT A VALUE OF A DIMENSION-3-WINDOW SIZE IS LESS THAN OR EQUAL TO A VALUE OF A DIMENSION-3 OF THE FIRST INPUT TENSOR 756
DETERMINE WHETHER A TYPE OF PADDING IS SET TO A PARTICULAR TYPE, IN WHICH THE TYPE OF PADDING INDICATES WHICH ELEMENTS OF A WINDOW ARE TO BE USED IN COMPUTING THE OUTPUT, AND BASED ON THE TYPE OF PADDING BEING SET TO THE PARTICULAR TYPE, THE VALIDATING IS PERFORMED 760
BASED ON THE TYPE OF PADDING NOT BEING SET TO THE PARTICULAR TYPE, ONE OR MORE CHECKS ARE PERFORMED RELATING TO ONE OR MORE DIMENSIONS OF AN OUTPUT TENSOR 762
THE DETERMINING THE OPERATION IS BASED ON AT LEAST ONE SLIDING WINDOW STRIDE VALUE OF AN INPUT TENSOR 770
THE SET OF FUNCTION-SPECIFIC PARAMETERS INCLUDES AT LEAST ONE SLIDING WINDOW DIMENSION OF AN INPUT TENSOR 772

FIG. 7C

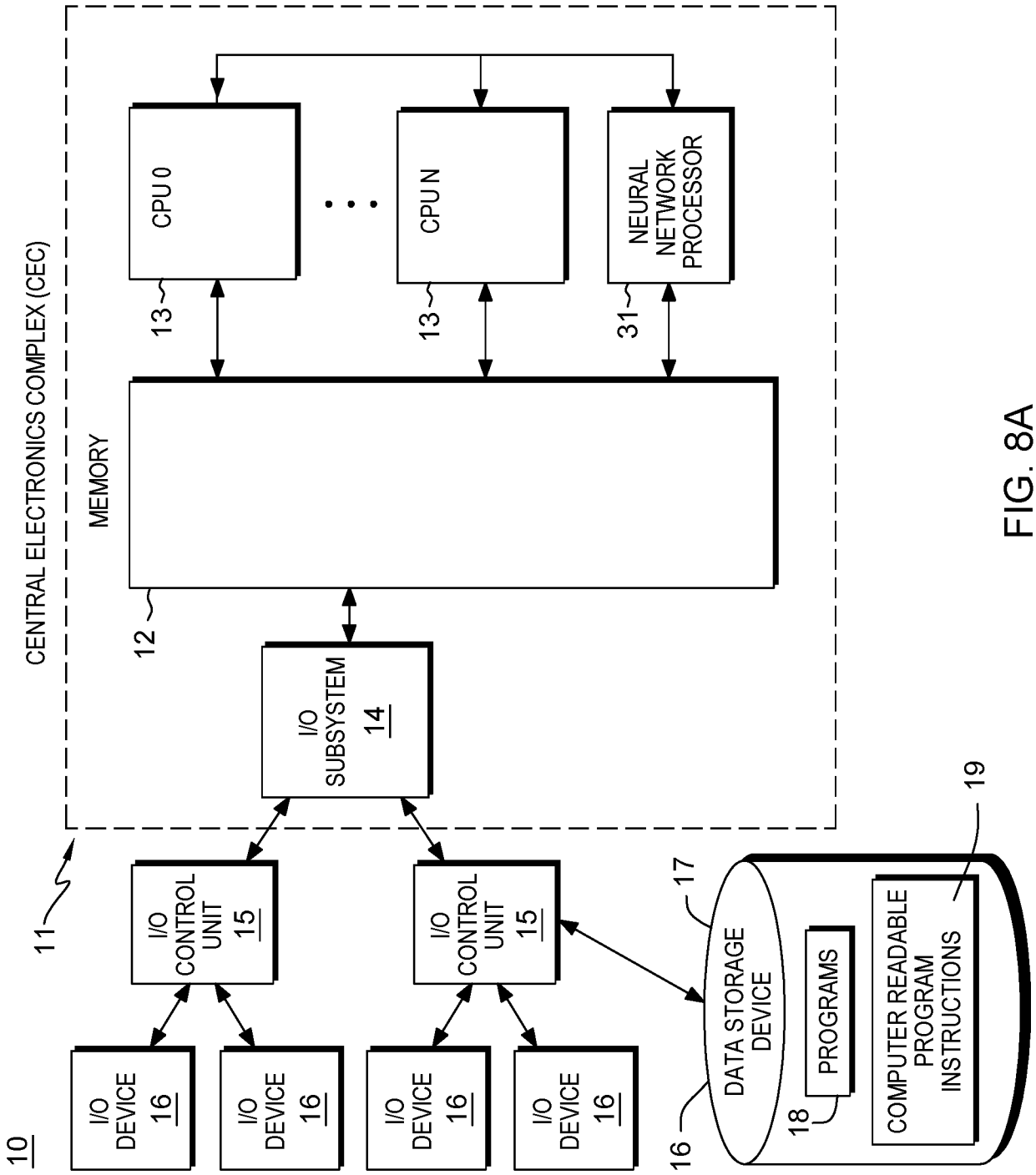

10
11
CENTRAL ELECTRONICS COMPLEX (CEC)
MEMORY
12
CPU 0
CPU N
NEURAL NETWORK PROCESSOR
13
13
31
I/O SUBSYSTEM 14
I/O CONTROL UNIT 15
I/O CONTROL UNIT 15
I/O DEVICE 16
I/O DEVICE 16
I/O DEVICE 16
I/O DEVICE 16
16
DATA STORAGE DEVICE
17
PROGRAMS
18
COMPUTER READABLE PROGRAM INSTRUCTIONS
19

FIG. 8A

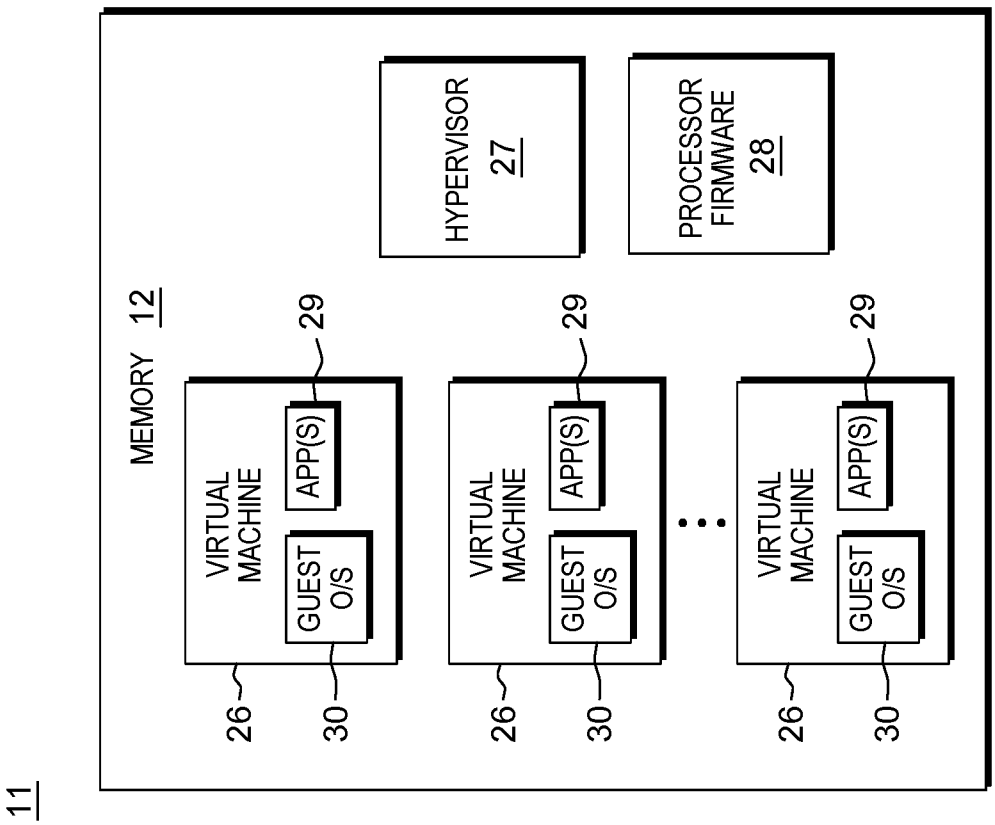

11
MEMORY 12
VIRTUAL MACHINE
APP(S)
GUEST O/S
29
26
30
HYPERVISOR 27
PROCESSOR FIRMWARE 28

FIG. 8C

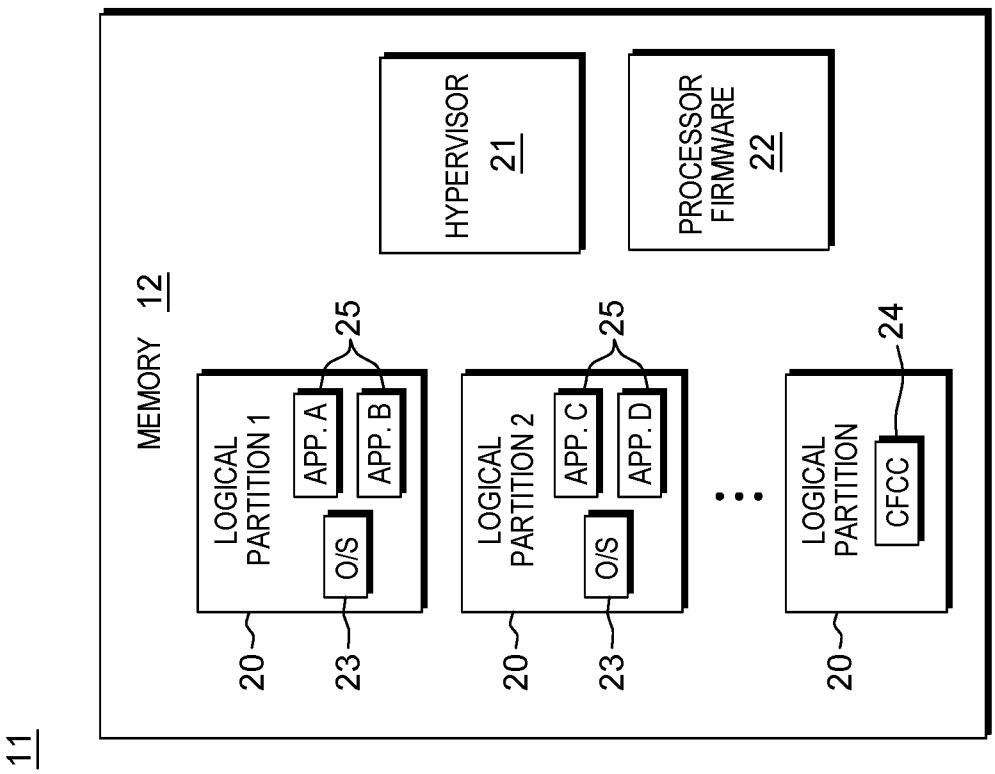

11
MEMORY 12
LOGICAL PARTITION 1
APP. A
APP. B
O/S
25
20
23
LOGICAL PARTITION 2
APP. C
APP. D
O/S
25
20
23
LOGICAL PARTITION
CFCC
24
20
HYPERVISOR 21
PROCESSOR FIRMWARE 22

FIG. 8B

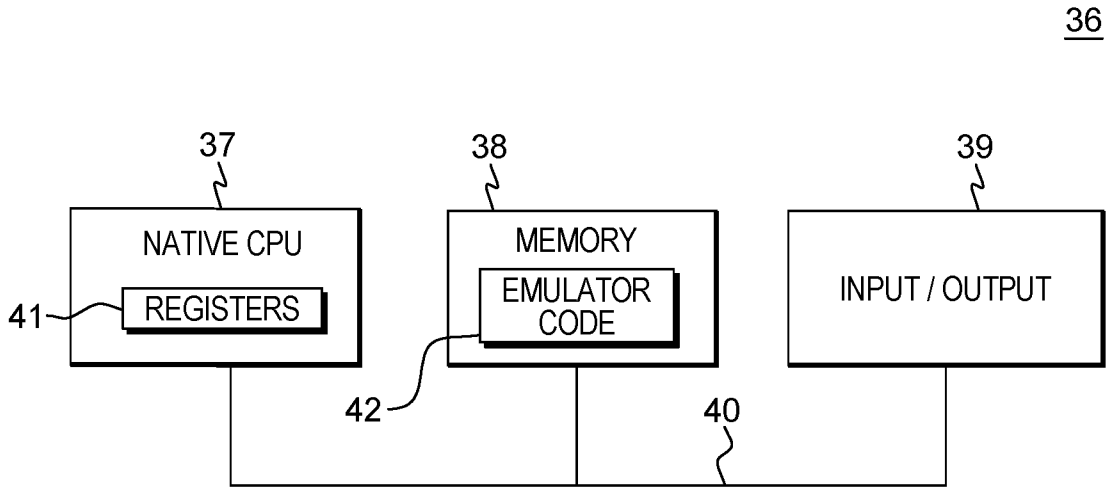

36
37
38
39
NATIVE CPU
REGISTERS
41
MEMORY
EMULATOR CODE
42
INPUT / OUTPUT
40

FIG. 9A

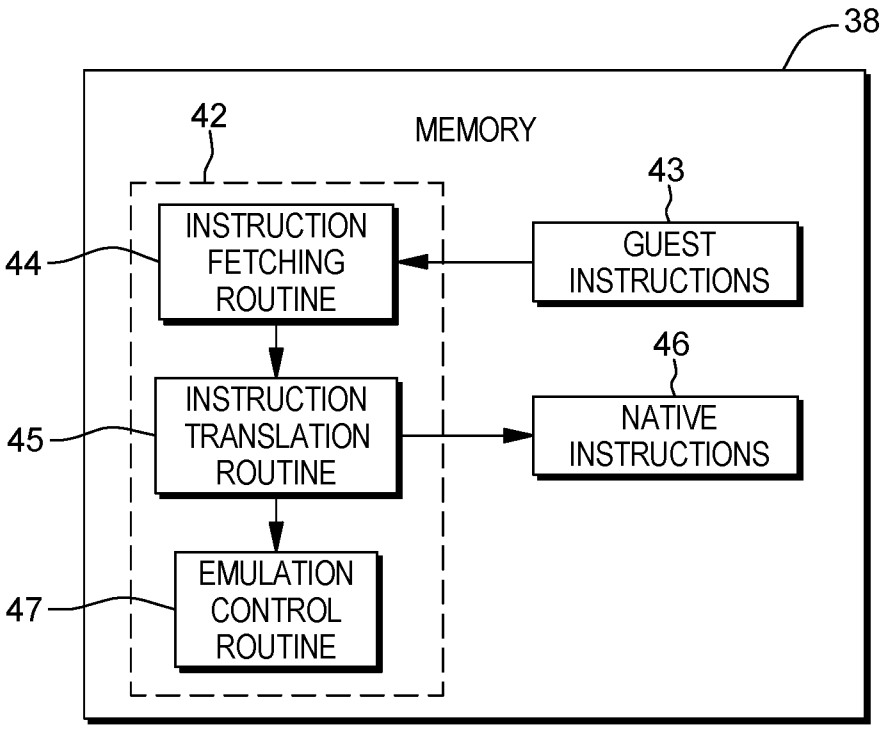

38
42
MEMORY
43
INSTRUCTION FETCHING ROUTINE
44
GUEST INSTRUCTIONS
46
INSTRUCTION TRANSLATION ROUTINE
45
NATIVE INSTRUCTIONS
EMULATION CONTROL ROUTINE
47

FIG. 9B

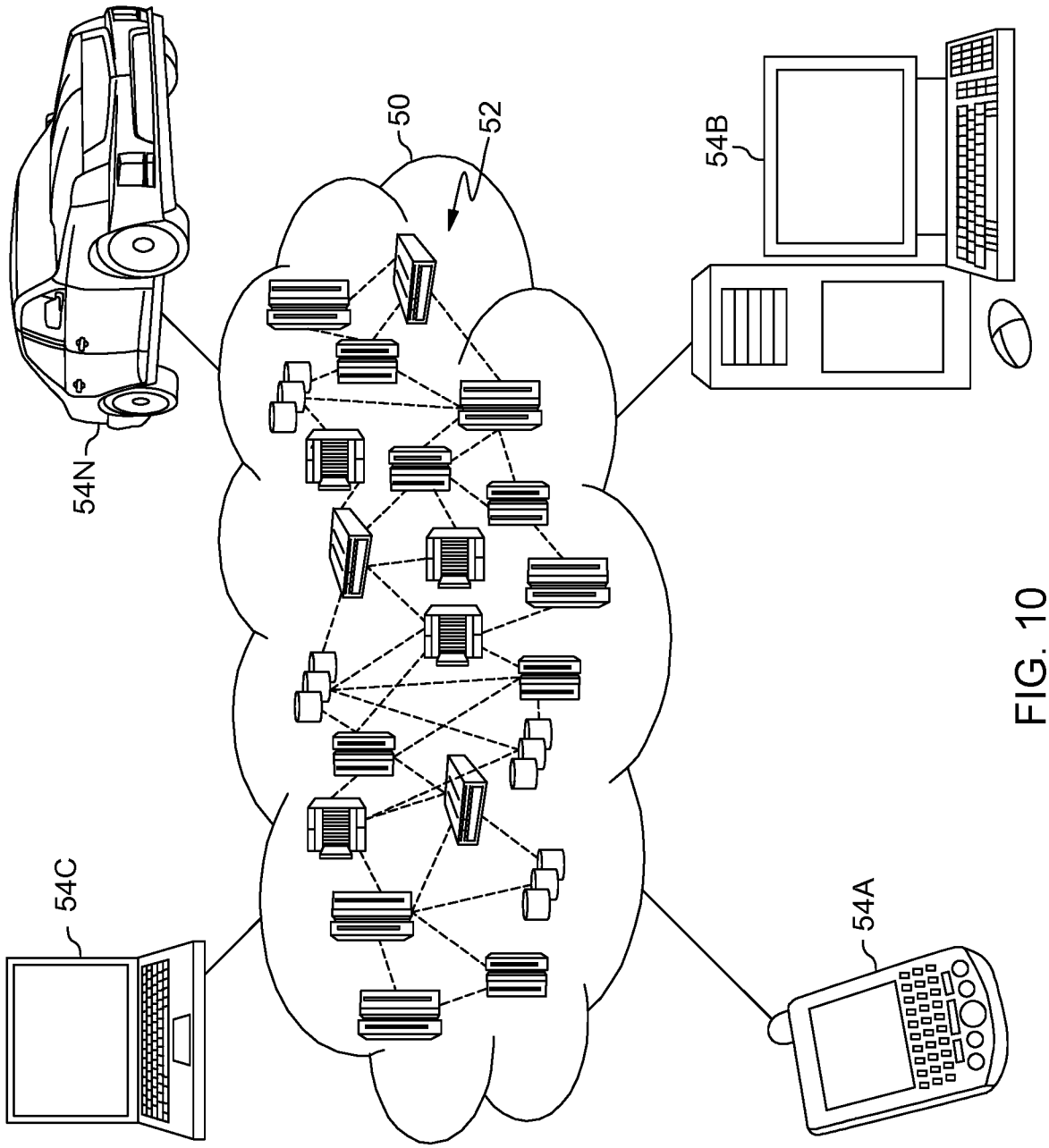


FIG. 10

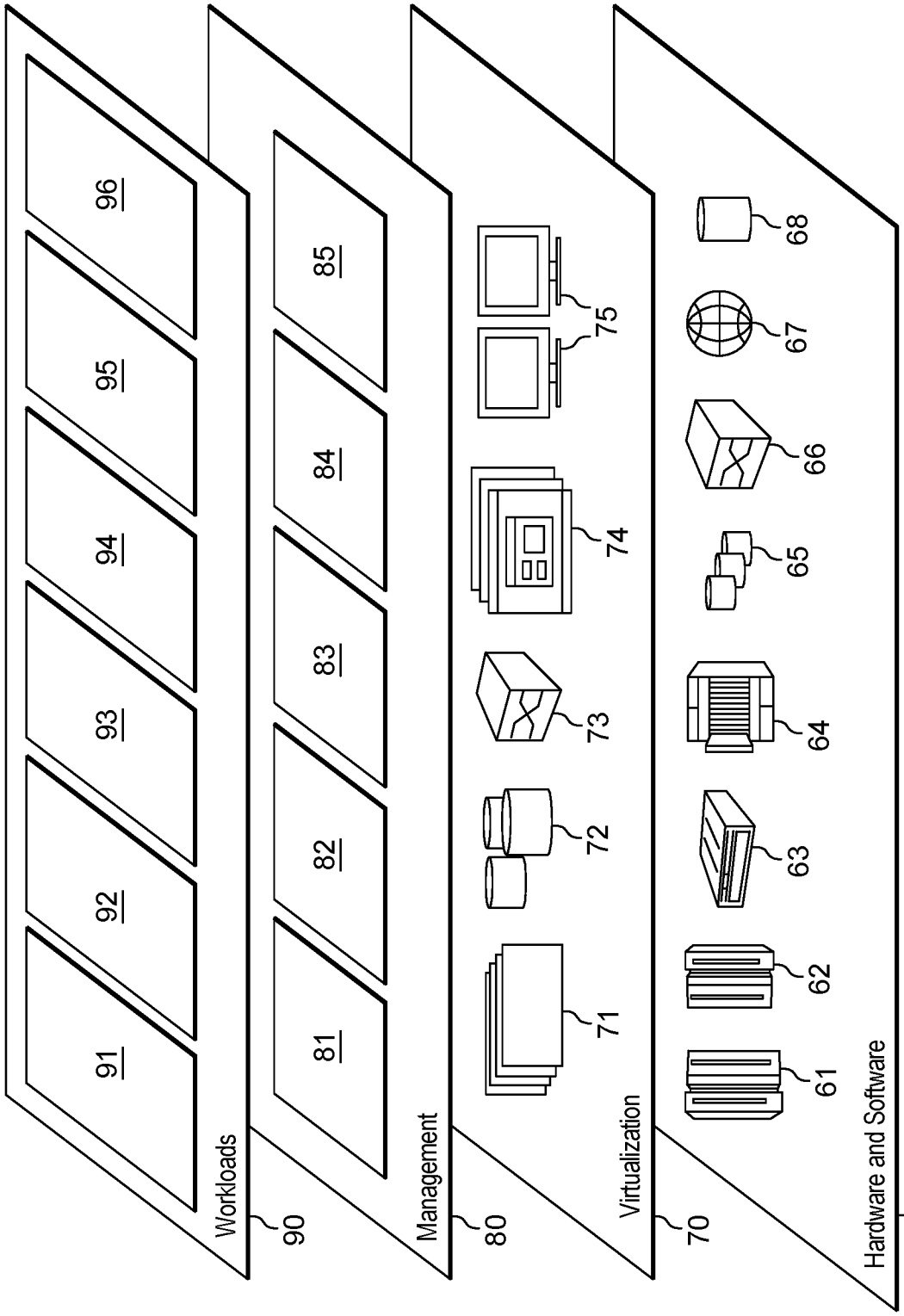

Workloads
90
91
92
93
94
95
96
Management
80
81
82
83
84
85
Virtualization
70
71
72
73
74
75
Hardware and Software
60
61
62
63
64
65
66
67
68

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2022/065660

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/30 G06N3/063
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F G06N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Amd: ""RDNA 2" Instruction Set Architecture", , 30 November 2020 (2020-11-30), pages 1-291, XP055964979, Retrieved from the Internet: URL:https://developer.amd.com/wp-content/resources/RDNA2_Shader_ISA_November2020.pdf [retrieved on 2022-09-26] page 16 - page 20 page 80 - page 82 page 222 ----- -/--	1-20

[x] Further documents are listed in the continuation of Box C.

[x] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance;; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance;; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
27 September 2022	06/10/2022

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Alecu, Mihail

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2022/065660

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2018/165577 A1 (YOUNG REGINALD CLIFFORD [US] ET AL) 14 June 2018 (2018-06-14) paragraph [0005] - paragraph [0010] paragraph [0032] paragraph [0035] paragraph [0040] paragraph [0066] - paragraph [0072] paragraph [0075] - paragraph [0077] -----	1-20
A	ABDELFATTAH MOHAMED S ET AL: "DLA: Compiler and FPGA Overlay for Neural Network Inference Acceleration", 2018 28TH INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL), IEEE, 27 August 2018 (2018-08-27), pages 411-4117, XP033462516, DOI: 10.1109/FPL.2018.00077 [retrieved on 2018-11-12] page 416, left-hand column - right-hand column -----	1-20

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2022/065660

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2018165577 A1	14-06-2018	CN 108615072 A	02-10-2018
		CN 114239797 A	25-03-2022
		DE 102017121257 A1	14-06-2018
		DE 202017105528 U1	18-12-2017
		EP 3555814 A1	23-10-2019
		GB 2557703 A	27-06-2018
		JP 6900487 B2	07-07-2021
		JP 2020506454 A	27-02-2020
		KR 20190089204 A	30-07-2019
		KR 20210127813 A	22-10-2021
		SG 10201707701P A	30-07-2018
		SG 10201805259X A	30-07-2018
		US 2018165577 A1	14-06-2018
		US 2018300628 A1	18-10-2018
		US 2019354863 A1	21-11-2019
		WO 2018111357 A1	21-06-2018