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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 July 2020 (16.07.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/142973 A1

(51) International Patent Classification:
G06N 3/02 (2006.01)

(21) International Application Number:
PCT/CN2019/071090

(22) International Filing Date:
10 January 2019 (10.01.2019)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **ALIBABA GROUP HOLDING LIMITED** [—/CN]; Fourth Floor, One Capital Place, P.O. Box 847, George Town, Grand Cayman (KY).

(72) Inventors: **LIU, Xiaoyong**; 525 Almanor Ave, 4th Floor, Sunnyvale, California 94085 (US). **CUI, Shiqiang**; Building 1, No.969 West Wen Yi Road, Yu Hang District, Hangzhou, Zhejiang 311121 (CN). **YANG, Yueming**; Building 1, No.969 West Wen Yi Road, Yu Hang District, Hangzhou, Zhejiang 311121 (CN).

(74) Agent: **BEIJING TSINGYUANHUI INTELLECTUAL PROPERTY LAW FIRM**; Room 362, 3rd Floor, Suzhou Street No.1, Haidian District, Beijing 100080 (CN).

(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, IR, IS, 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,

(54) Title: MATRIX-BASED INSTRUCTION SET ARCHITECTURE FOR NEURAL NETWORK

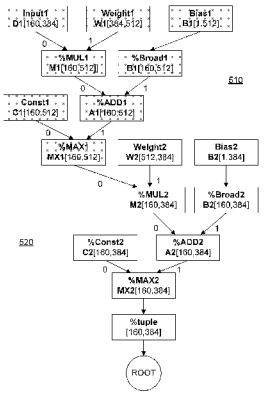

FIG. 5A

```
t1: MatLoad %R3[1, 512] sub{1,1,1,512;%R1[1,512]}
t2: MatLoad %R5[160, 384] sub{1,1;160,384;%D1[160,384]}
t3: MatLoad %R6[384, 128] sub{1,1;384,128;%W1[384,512]}
t4: MatMul %R7[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1;128,128;%R6[384,128]} %const 0
t5: MatMul %R7[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128;%R6[384,128]} %R7[160,128]
t6: MatMul %R7[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128;%R6[384,128]} %R7[160,128]
t7: MatMisc sub{1,1;160,128;%R4[160,512]} %R7[160,128] sub{1,1,1,128;%R3[1,512]} sub{0,%constDDR}
t8: MatLoad %R5[384, 128] sub{1,129;384,256;%W1[384,512]}
t9: MatMul%R9[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1,128,128;%R6[384,128]} %const 0
t10: MatMul %R9[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1,256,128;%R8[384,128]} %R9[160,128]
t11:MatMul %R9[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128;%R6[384,128]} %R9[160,128]
t12:MatMisc sub{1,129;160,256;%R11[160,512]} %D[160,128] sub{1,129,1,256 %R3[1,512]} sub{0,%constDDR}
```

FIG. 5B

```
t13: MatLoad %R10[384, 128] sub{1,257;384,384;%W1[384,512]}
t14: MatMul %R11[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1;128,128; %R10[384,128]} %const 0
t15: MatMul  %R11[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128; %R10[384,128]} %R11[160,128]
t16: MatMul %R11[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128; %R10[384,128]} %R11[160,128]
t17: MatMisc sub{1,257;160,384;%R4[160,512]} %R11[160,128] sub{1,257,1,384, %R3[1,512]} sub{0,%constDDR}
t18: MatLoad %R12[384, 128] sub{1,257;384,384;%W1[384,512]}
t19: MatMul %R13[160, 128] sub{1,1;160,128,%R5[160,384]} sub{1,1,128,128, %R12[384,128]} %const 0
t20: MatMul %R13[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128; %R12[384,128]} %R13[160,128]
t21: MatMul %R13[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128; %R12[384,128]} %R13[160,128]
t22: MatMisc sub{1,385;160,512;%R4[160,512]} %R13[160,128] sub{1,3854,1,512; %R3[1,512]} sub{0;%constDDR}
```

FIG. 5C

```
t23: MatLoad %R3[1, 384] sub{1,1,1,384;%R2[1,384]}
t24: MatLoad %R6[512, 128] sub{1,1;512,128;%W2[512,384]}
t25: MatMul %R7[160, 128] sub{1,1;160,128;%R4[160,512]} sub{1,1 128,128;%t6[512,128]} %const 0
t26: MatMul %R7[160, 128] sub{1,129;160,256;%R4[160,512]} sub{129,1;256,128;%R6[512,128]} %R7[160,128]
t27: MatMul %R7[160, 128] sub{1,257;160,384 %R4[160,512]} sub{257,1;384,128 %t6[512,128]} %R7[160,128]
t28: MatMul %R7[160, 128] sub{1,385;160,512;%R4[160,512]} sub{385,1;512,128;%R6[512,128]} %R9[160,128]
t29: MatMisc sub{1 1 160,129;%R14[160,384]} %R7[160,128] sub{1,1:1,128 %R3[1,384]} sub{0 %constD30}
t30: MatLoad %R8[512, 128] sub{1,129;512,256;%W2[512,384]}
t31: MatMul %R9[160, 128] sub{1,1;160,128;%R4[160,512]} sub{1,1 128,128;%t6[512,128]} %const 0
t32: MatMul %R9[160, 128] sub{1,129,160,256;%R4[160,512]} sub{129,1,256,128;%t8[512,128]} %R9[160,128]
t33: MatMul %R9[160, 128] sub{1,257;160,384;%R4[160,512]} sub{257,1;384,128;%t8[512,128]} %R9[160,128]
t34: MatMul %R9[160, 128] sub{1,385;160,512,%R11[160,512]} sub{385,1;512,128,%t8[512,128]} %R9[160,128]
t35: MatMisc sub{1,129,160,256;%R14[160,384]} %R9[160,128] sub{1,129,1,256;%R31[1,384]} sub{0;%constDDR}
t36: MatLoad %R10[512, 128] sub{1,257;512,384;%W2[512,384]}
t37: MatMul %R11[160, 128] sub{1,1;160,128;%R4[160,512]} sub{1,1,128,128;%R10[512,128]} %const 0
t38: MatMul %R11[160, 128] sub{1,129;160,256;%R4[160,512]} sub{129,1;256,128;%R10[512,128]} %R11[160,128]
t39: MatMul %R11[160, 128] sub{1,257;160,384;%R4[160,512]} sub{257,1;384,128;%R10[512,128]} %R11[160,128]
t40: MatMul %R11[160, 128] sub{1,385;160,512;%R4[160,512]} sub{385,1;512,128;%R10[512,128]} %R11[160,128]
t41: MatMisc sub{1,257;160,381;%R11[160,384]} sub{1,257,1,381 %R3[1,384]} sub{0;%constDDR}
t42: MatStore sub{1,1;160,384; %D[R] %R14[160,384]
```

FIG. 5D

(57) Abstract: The present disclosure relates to a computing device for executing a set of instructions for a neural network model. The computing device comprises a computation module including a plurality of processing units performing at least one operation according to the set of instructions, a first memory module storing the set of instructions and input parameters for executing the set of instructions, and a second memory module including a plurality of memory blocks. The second memory module is configured to store a first matrix of the input parameters loaded from the first memory module in a first memory block of the plurality of memory blocks according to a first instruction of the set of instructions. The first memory block of the second memory module has a first memory cell array meeting dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

WO 2020/142973 A1

[Continued on next page]

SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, 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))*

MATRIX-BASED INSTRUCTION SET ARCHITECTURE FOR NEURAL NETWORK

BACKGROUND

[001] With the growth of machine learning and deep learning technologies, various types of accelerators for machine learning or deep learning have begun to emerge. A general machine learning hardware platform includes a general-purpose host device and one or more accelerators. The accelerator is usually responsible for processing machine learning specific operations. A typical machine learning or deep learning model may have thousands or even millions of variables and computation operations. Therefore, how to effectively utilize resources of the accelerators and how to quickly process operations on the accelerators have become an urgent priority to address.

SUMMARY

[002] Embodiments of the present disclosure provide a computing device for executing a set of instructions for a neural network model. The computing device comprises a computation module including a plurality of processing units performing at least one operation according to the set of instructions, a first memory module storing the set of instructions and input parameters for executing the set of instructions, and a second memory module including a plurality of memory blocks. The second memory module is configured to store a first matrix of the input parameters loaded from the first memory module in a first memory block of the plurality of memory blocks according to a first instruction of the set of instructions. The first memory block of the second memory module has a first memory cell array meeting dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

[003] Embodiments of the present disclosure also provide a method for executing a set of instructions for a neural network in a computing device. The computing device comprises a computation module including a plurality of processing units performing at least one operation according to the set of instructions, a first memory module storing the set of instructions and input parameters for executing the set of instructions; and a second memory module including a plurality of memory blocks. The method for executing a set of instructions for a neural network comprises loading a first matrix of the input parameters from the first memory module to the second memory module according to a first instruction of the set of instructions. The first matrix is stored in a first memory block of the plurality of memory blocks, the first memory block of the second memory module has a first memory cell array meeting dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

[004] Embodiments of the present disclosure also provide a non-transitory computer readable medium that stores a set of instructions that is executable by a computing device to cause the computing device to perform a method for executing a neural network. The computing device comprises a computation module including a plurality of processing units performing at least one operation according to the set of instructions, a first memory module storing input parameters; and a second memory module including a plurality of memory blocks. The method for executing a neural network comprises loading a first matrix of the input parameters from the first memory module to the second memory module according to a first instruction of the set of instructions. The first matrix is stored in a first memory block of the plurality of memory blocks, the first memory block of the second memory module has a first memory cell array meeting

dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

[005] The first instruction can define a range of the first matrix out of a second matrix stored in a first memory block of the first memory module to be loaded to the first memory module. The first memory block of the first memory module can have a second memory cell array meeting dimension of the second matrix, and each element of the second matrix is stored in a memory cell of a corresponding address index in the second memory cell array. The range of the first matrix can be defined by a first address index for a first element of the first matrix in the second matrix and a second address index for a second element of the first matrix in the second matrix. The first element is positioned at one corner of the first matrix and the second element is positioned at another corner farthest from the first element of the first matrix.

[006] The neural network model comprises two or more layers and one processing unit of the plurality of processing units can be configured to execute at least one of the two or more layers on matrix-based instructions. The set of instructions can include at least one of a matrix-based loading instruction, a matrix-based storing instruction, or a matrix-based operation instruction. The second memory module can comprise a first segment and a second segment. The first segment is configured to store data that is determined to interact with an off-chip device, and the second segment is configured to store data that is determined to be reused by the computation module within a preset time period.

BRIEF DESCRIPTION OF THE DRAWINGS

[007] **FIG. 1** illustrates an exemplary accelerator architecture, consistent with embodiments of the present disclosure.

[008] **FIG. 2** illustrates a schematic diagram of an exemplary neural network accelerator system, consistent with embodiments of the present disclosure.

[009] **FIG. 3** illustrates a schematic diagram of processing units and on-chip memory system, consistent with embodiments of the present disclosure.

[010] **FIG. 4** illustrates an example of a computational neural network.

[011] **FIG. 5A** illustrates an example of an intermediate representation associated with a computation graph having a 2-layer neural network.

[012] **FIG. 5B** to **FIG. 5D** illustrates a set of instructions for executing a 2-layer neural network of **FIG. 5A**

[013] **FIG. 6** illustrates an example for loading data on a matrix basis according to a matrix-based loading instruction, consistent with embodiments of the present disclosure.

[014] **FIG. 7A** illustrates a conventional data flow for processing a 3-layer neural network model.

[015] **FIG. 7B** illustrates an exemplary data flow for processing a 3-layer neural network model, consistent with embodiments of the present disclosure.

[016] **FIG. 8** illustrates an exemplary flow diagram for executing a neural network model in a neural network accelerator system, consistent with embodiments of the present disclosure.

DETAILED DESCRIPTION

[017] Reference will now be made in detail to exemplary embodiments, examples of which are illustrated in the accompanying drawings. The following description refers to the accompanying drawings in which the same numbers in different drawings represent the same or

similar elements unless otherwise represented. The implementations set forth in the following description of exemplary embodiments do not represent all implementations consistent with the invention. Instead, they are merely examples of apparatuses and methods consistent with aspects related to the invention as recited in the appended claims.

[018] Generally, operations processed on accelerators are mainly based on multi-dimensional matrices. Conventional instructions for executing a machine learning model on accelerators, however, are based on scalars or one-dimensional vectors, which are not suitable for efficiently processing multi-dimensional matrix-based operations. This causes delay in processing operations on the accelerators and degrades overall performance of the machine learning system. The disclosed embodiments provide an instruction set architecture suitable for executing matrix-based instructions on accelerators. The disclosed embodiments also provide apparatuses and methods for executing matrix-based instructions. The disclosed embodiments also provide methods for optimizing data transmission between a host device and an accelerator device. The disclosed embodiments also provide a memory layout suitable for executing matrix-based instructions on an accelerator device. The disclosed embodiments also provide methods for optimizing hardware resource usage of an accelerator device.

[019] Moreover, some of the embodiments of the present disclosure can utilize instruction set architecture utilizing a memory (e.g., DDR memory) or buffer (e.g., cache memory) to enable not only a more efficient compilation optimization for matrix-oriented hardware architecture, but also high-speed memory loading and reuse. Some of the embodiments can utilize an optimization strategy to schedule the on-chip high speed memory such as cache memory in a static state or can reflect the lower level reconfigurable hardware.

[020] **FIG. 1** illustrates an exemplary accelerator architecture 100, consistent with embodiments of the present disclosure. An accelerator architecture 100 can include an on-chip communication system 110, an off-chip memory 120, a memory controller 130, a direct memory access (DMA) unit 140, a Joint Test Action Group (JTAG)/Test Access End (TAP) controller 150, peripheral interface 160, a bus 170, and the like. It is appreciated that on-chip communication system 110 can perform algorithmic operations based on communicated data. Moreover, while not shown, an accelerator architecture 100 can include an on-chip memory, generally, having less latency and smaller capacity than the off-chip memory 120. SRAM (Static Random-Access Memory) can be used as the on-chip memory.

[021] Chip communication system 110 can include a global manager 112 and a plurality of cores 116. Global manager 112 can include one or more task managers 114 configured to coordinate with one or more cores 116. Each task manager 114 can be associated with an array of cores 116 that provide synapse/neuron circuitry for the neural network. As shown in **FIG. 1**, global manager 112 can include two task managers 114 configured to coordinate with corresponding arrays of cores 116.

[022] Cores 116 can include one or more processing units configured to perform one or more operations (e.g., multiplication, addition, multiply-accumulate, etc.) on the communicated data under the control of global manager 112. To perform the operation on the communicated data packets, cores 116 can include one or more processing units for processing information in the data packets. On-chip memory can be shared by the one or more processing units. In some embodiments, core 116 can be considered a tile or the like.

[023] Off-chip memory 120 can be a DDR (Dual Data Rate) memory (e.g., DDR SDRAM (Synchronous Dynamic Random-Access Memory)), HBM (High Bandwidth Memory),

etc. HBM (e.g., HBM2) uses 3D stacking technology (e.g., allowing up to 8 dies in a stack, 4 packages, and 8GB per package) and is efficient from a power and area standpoint. Off-chip memory 120 can be configured to store a large amount of data with slower access speed and more consumption energy, compared to the on-chip memory having storage on the processor chip.

[024] Memory controller 130 can read, write, or refresh one or more memory devices. The memory devices can include on-chip memory and off-chip memory 120. For example, the memory device can be implemented as any type of volatile or non-volatile memory devices, or a combination thereof, such as a static random access memory (SRAM), an electrically erasable programmable read-only memory (EEPROM), an erasable programmable read-only memory (EPROM), a programmable read-only memory (PROM), a read-only memory (ROM), a magnetic memory, a flash memory, or a magnetic or optical disk. Moreover, while one memory controller is shown in **FIG. 1**, it is appreciated that more than one memory controller can be provided in NPU architecture 100.

[025] Memory controller 130 can generate memory addresses and initiate memory read or write cycles. Memory controller 130 can contain several hardware registers that can be written and read by the one or more processors. The registers can include a memory address register, a byte-count register, one or more control registers, and other types of registers. These registers can specify some combination of the source, the destination, the direction of the transfer (reading from the input/output (I/O) device or writing to the I/O device), the size of the transfer unit, the number of bytes to transfer in one burst, and/or other typical features of memory controllers.

[026] DMA unit 140 can assist with transferring data between off-chip memory 120 and memory associated with a host device. In addition, DMA unit 140 can assist with transferring

data between multiple accelerators (e.g., accelerator 100). DMA unit 140 can allow off-chip devices to access both on-chip and off-chip memory without causing a CPU interruption. Thus, DMA unit 140 can also generate memory addresses and initiate memory read or write cycles. DMA unit 140 can contain several hardware registers that can be written and read by the one or more processors, including a memory address register, a byte-count register, one or more control registers, and other types of registers. These registers can specify some combination of the source, the destination, the direction of the transfer (reading from the input/output (I/O) device or writing to the I/O device), the size of the transfer unit, and/or the number of bytes to transfer in one burst. It is appreciated that an accelerator architecture 100 can include a second DMA unit, which can be used to transfer data between other accelerator architectures to allow multiple accelerator architectures to communicate directly without involving the host CPU.

[027] JTAG/TAP controller 150 can specify a dedicated debug port implementing a serial communications interface (e.g., a JTAG interface) for low-overhead access without requiring direct external access to the system address and data buses. The JTAG/TAP controller 150 can also specify an on-chip test access interface (e.g., a TAP interface) that implements a protocol to access a set of test registers that present chip logic levels and device capabilities of various parts.

[028] Peripheral interface 160 (such as a PCIe interface), if present, serves as an (and typically the) inter-chip bus, providing communication between the accelerator and other devices.

[029] Bus 170 includes both intra-chip bus and inter-chip buses. The intra-chip bus connects all internal components to one another as called for by the system architecture. While not all components are connected to every other component, all components do have some

connection to other components they need to communicate with. The inter-chip bus connects the accelerator with other devices, such as the off-chip memory or peripherals. Typically, if there is a peripheral interface 160 (e.g., the inter-chip bus), bus 170 is solely concerned with intra-chip buses, though in some implementations it could still be concerned with specialized inter-bus communications.

[030] While the disclosed embodiments are described with respect to an accelerator architecture 100 for accelerating some applications such as deep learning, it is appreciated that the embodiments of the present disclosure could be applied to, for example, GPU (Graphics Processing Unit), FPGA (Field Programmable Gate Array), CPU (Central Processing Unit), or other types of neural network accelerators for deep learning.

[031] **FIG. 2** illustrates a schematic diagram of an exemplary neural network accelerator system 200, consistent with embodiments of the present disclosure. Neural network accelerator system 200 may include a host device 210 and a target device 220 having, for example, the accelerator architecture 100 of **FIG. 1**. Here, the target device 220 can be designed for processing neural network models. The target device 220 may be connected to the host device 210 through a peripheral interface 160 or an inter-chip bus 170. NPU (Neural Processing Unit) 221 is the key computing device of the target device 220. CPU 211 of the host device 210 can issue instructions or commands to the target device 220, which performs the computation according to the instructions or commands and sends the results back to the host device 210. The host device 210 can be associated with its own memory device 212 and the target device 220 can be associated with its own memory devices 222 and 223. In some embodiments, the target device 220 can be implemented as a heterogenous accelerator where processing units do not have equal processing performance with each other.

[032] Reference is still made to **FIG. 2** to explain a memory layout for a target device 220, consistent with embodiments of the present disclosure. As shown in **FIG. 2**, the target device 220 according to embodiments of the present disclosure can include a first memory module 222 and a second memory module 223. NPU 221 can be implemented as a chip comprising a computation module 224 incorporated therein. The second memory module 223 can be called an on-chip memory in that the second memory module 223 is mounted on the chip of NPU 221 having the computation module 224. The first memory module 222 can be called an off-chip memory in that the first memory module 222 is placed off the chip of NPU 221. The first memory module 222 can serve as a main memory for a target device 220. Instructions, input parameters, and other data for executing a program such as a neural network model can be transferred from a host device 210 to the first memory module 222. The second memory module 223 is used by a computation module 224 of a target device 220 to reduce the average cost (e.g., time and energy) to access data from/to the first memory module 222. The second memory module 223 can be smaller and faster than the first memory module 222. The second memory module 223 can store copies of the data such as instructions, input parameters, etc. from the first memory module 222. The second memory module 223 can also store intermediate results from the computation module 224 during executing a computer program such as a neural network model. In some embodiments, the second memory module 223 can store data in a matrix form and has a relatively large size. For example, the second memory module 223 can be implemented to have a capacity of 100 MB (megabyte) and more.

[033] In some embodiments, task manager 114 can assign any number of tasks to one or more cores (e.g., core 116) according to commands from the host device 210. Here, the computation module 224 can be implemented to comprise one or more cores and each of the

cores include a plurality of processing units. Some of the commands can instruct DMA unit 140 to load instructions and data from a host memory 212 to the first memory module 222. The loaded instructions can then be distributed to each core assigned with the corresponding task, and the one or more cores can process these instructions. According to the first few instructions, data from the first memory module 222 can be loaded/stored into the second memory module 223. Each core can then initiate the instruction pipeline, which involves fetching the instruction from the second memory module 223, decoding the instruction and generating memory addresses of data (e.g., corresponding to an operand) to be used for executing operations, reading the data from the second memory module 223, executing the operations, and then writing back results to the second memory module 223.

[034] In some embodiments of the present disclosure, a first memory module 222 can be configured to include a plurality of memory segments 222A to 222D. A memory address may comprise a segment identifier of a corresponding segment among the plurality of memory segments 222A to 222D and an offset within the segment. That is, a memory address can include a value that identifies a corresponding segment. It is illustrated in **FIG. 2**that the first memory module 222 includes four memory segments 222A to 222D, however, it is appreciated that any number of memory segments can be included in the first memory module 222 according to embodiments. The first memory module 222 can include a first memory segment 222A configured to store a set of instructions for executing a program on the target device 220. The set of instructions can be transferred from a memory 212 of the host device 210.

[035] The second memory segment 222B is configured to store data that is interactive with a host device 210. For example, the second memory segment 222B preserves data transferred from a host device 210 and results to be transferred to the host device 210. On the

other hand, the third memory segment 222C is configured to store data that is used internally in the target device 220. For example, the third memory segment 222C can store data that is determined to be used within a preset time period by the computation module 224. The preset time can be determined by taking capacities of the first and second memory modules 222 and 223 and operations and instructions to be processed into account. For example, the third memory segment 222C can store data that is internally used in the target device 220 during a time period for executing a corresponding program on the target device 220. By dividing the first memory module 222 to include a plurality of memory segments such as a second memory segment 222B and third memory segment 222C, it is possible to reduce time and energy for memory accessing and data storing into the first memory module 222. The first memory module 222 can include a fourth memory segment 222D for other data used for executing a program on the target device 220. According to embodiments of the present disclosure, the size of each of the plurality of memory segments 222A to 222D can be adjusted according to the needs of the system.

[036] Consistent with embodiments of the present disclosure, a target device 222 can includes a computation module 224. The computation module 224 is the electric circuitry within a target device 222 that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling and input/output operations specified by the instructions. The computation module 224 can includes a plurality of processing units configured to perform one or more operations (e.g., multiplication, addition, multiply-accumulate, etc.). A processing unit can be a fundamental building block of a computing circuit. For example, a processing unit can be an ALU (Arithmetic Logic Unit) implemented as a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. A processing unit can also be a MAC (Multiplier-Accumulator Unit) configured to perform a MAC operation, which

can be performed with floating point numbers. A processing unit can be configured to perform one or more operations according to embodiments of the present disclosure. It is also possible that a processing unit can be implemented by modifying an existing processing unit to perform one or more operations according to embodiments of the present disclosure.

[037] **FIG. 3** illustrates a schematic diagram of processing units and on-chip memory system, consistent with embodiments of the present disclosure. As mentioned, a computation module 224 of a target device 220 can include a plurality of processing units PU1 to PUn. The plurality of processing units can be connected to a common memory module, such as a second memory module 223. Each of the processing units PU1 to PUn fetches instructions and corresponding operands to process the instructions from a second memory module 223 and sends results back to the second memory module 223. Each of the processing units PU1 to PUn may be connected to its corresponding memory sections or blocks of the second memory module 223. The plurality of processing units PU1 to PUn can perform computation in a pipelined way to accelerate the computation, which will be explained in detail regarding **FIG. 5**.

[038] **FIG. 4** illustrates an example of a computational neural network. Generally, a neural network can be organized in layers. Each layer can perform one or more calculations on its inputs and generate outputs. The output of a layer can be passed onto a next layer for further process. For example, an output of a previous layer can be an input for the next layer. The neural networks that have more than three layers are called deep neural networks (DNN). A typical number of network layers used in deep learning range from five to more than a thousand. Although some embodiments are illustrated with a neural network with 2 or 3 layers, it is appreciated that the present disclosure can also be applied to deep learning networks with more than 2 or 3 layers.

[039] In **FIG. 4**, a neural network 400 with multiple layers, for example, layers 410 and 420 is illustrated. In some embodiments, it can be interpreted that the neural network 400 has more than 3 layers in that the activation function (such as "f(·)" explained below) between the first layer 410 and second layer 420 can be understood as a hidden layer. An example of the computation at each layer can be represented as below:

$$\boldsymbol{A} = f(\boldsymbol{X} * \boldsymbol{W} + \boldsymbol{b}1) \qquad \text{(Equation 1)}$$

$$\boldsymbol{Y} = g(\boldsymbol{A} * \boldsymbol{Z} + \boldsymbol{b}2) \qquad \text{(Equation 2)}$$

Here, the computational operation of the first layer 410 uses an input matrix **X**, a weight matrix **W**, and a bias matrix **b**1 as input operands and outputs an output matrix **A**. The computational operation of the second layer 420 uses an output matrix **A** of the first layer 410, a weight matrix **Z**, and a bias matrix **b**2 as input operands and outputs an output matrix **Y**. The input matrix **X** includes x1 to x4 as its elements. An element wij of the weight matrix **W** represents a weight value corresponding to an arrow from an input node xi to an output node aj. The output matrix **A** of the first layer 410 includes a1 to a4 as its elements and the output matrix **Y** of the second layer 420 includes y1 to y4 as its elements. "f(·)" and "g(·)" can represent activation functions such as a sigmoid function, etc. Bias matrices **b**1 and **b**2 are bias matrices for each layer. The assignment arrows for the bias values b1 and b2 of the bias matrices **b**1 and **b**2 are omitted in **FIG. 4** for simplicity. It is illustrated that the same bias vector b1 is used for each array in the first layer 410 and the same bias vector b2 is used for each array in the second layer 420 in **FIG. 4**, but it is appreciated that different bias values can be used for arrays in each layer 410 and 420.

[040] As illustrated above, in neural networks, operations of the layers generally include matrix computations based on matrix operands. The conventional computation, however, has

been performed on a vector or scalar basis in the accelerator. For example, conventionally four processing units have been used to compute, in parallel, the following operations:

$$a1 = f(x1 * w11 + x2 * w21 + x3 * w31 + x4 * w41 + b1) \quad \text{(Equation 3)}$$

$$a2 = f(x1 * w12 + x2 * w22 + x3 * w32 + x4 * w42 + b1) \quad \text{(Equation 4)}$$

$$a3 = f(x1 * w13 + x2 * w23 + x3 * w33 + x4 * w43 + b1) \quad \text{(Equation 5)}$$

$$a4 = f(x1 * w14 + x2 * w24 + x3 * w34 + x4 * w44 + b1) \quad \text{(Equation 6)}$$

[041] To process such operations in parallel, chips having single instruction, multiple data (SIMD) architecture have been used. SIMD architecture includes one or more processing units configured to perform one or more operations. In the SIMD architecture, each of the processing units can perform the same operation on multiple data points simultaneously.

[042] In conventional systems, instruction sets for processing operations on accelerator chips have been generated on a vector or scalar basis. For example, data-load instructions, data-store instructions, data-preparation instructions, data-operation instructions, etc. are defined on a vector or scalar basis. Such instructions are not efficient in performing matrix operations based on multi-dimensional matrix type operands. To process one matrix multiplication operation, multiple instructions have to be generated and transferred to the accelerator. For example, each element (e.g., a1 to a4) of the output matrix **A** is processed in its own processing unit, after which output elements (results of the operations) are transferred to host device 210. The host device 210 can then reorganize the output elements to find where and when to use the output elements during the execution of the neural network.

[043] The disclosed embodiments overcome these inefficiencies by providing a matrix-based instruction set architecture suitable for executing matrix-based operations on neural

processing accelerators. The disclosed embodiments describe processing a neural network model in a neural network accelerator system based on the matrix-based instruction set

[044] **FIG. 5A** illustrates an example of an intermediate representation associated with a computation graph having a 2-layer neural network, consistent with embodiments of the present disclosure. In this example, a 2-dimensional matrix is used as the multi-dimensional matrix, however, it is appreciated that the present disclosure can be applied to a 3 or more-dimensional matrix.

[045] In **FIG. 5A**, the shaded nodes indicate a first layer 510 of the neural network, which includes a first multiplication operation MUL1, a first addition operation ADD1, and a first activation function MAX1. In the first layer 510, the first multiplication operation MUL1 receives a first input matrix **D1** (having dimensions of 160×384; annotated as **D1**[160,384]) and first weight matrix **W1** [384, 512] as its input operands and outputs its multiplication result matrix **M1** [160, 512] to the first addition operation ADD1. The first broadcast operation Broad1 receives a first bias vector **B1** [1, 512] and transforms the first bias vector **B1** (which can also be considered a bias matrix having one row or one column) into a first bias matrix **B1**, so that **B1** has compatible dimensions with the multiplication result matrix **M1**. In other words, the elements in the first bias vector **B1** having dimension [1, 512] are broadcasted to make the bias vector **B1** have the same dimensions as the multiplication result matrix **M** [160, 512]. Here, the first broadcast operation Broad1 can be performed by replicating the first bias vector **B1** [1, 512] 160 times to make a first bias matrix **B1** [160, 512].

[046] After making the dimensions compatible, the first broadcast operation Broad1 outputs a first bias matrix **B1** for computing the first addition operation ADD1 between the multiplication result matrix **M1** and the bias matrix **B1**. A first addition operation ADD1

receives the result matrices **M1** and **B1** of the first multiplication operation MUL1 and broadcast operation Broad1, and outputs its addition result matrix **A1** [160, 512] to a first activation function MAX1. Here, as an example, a ReLU (rectified linear unit) function is used as the first activation function, which can be expressed as f(x)=max (x, 0). The first activation function MAX1 receives the addition result matrix **A1** and a first constant matrix **C1** [160, 512] as its input operands. ReLU function, which is an element-wise operation, compares each element of the input matrix (here, a result matrix **A1**) with a constant value "0" and outputs the bigger value between the element value and constant value "0" as an output element for an output matrix (here, a result matrix **MX1**). Therefore, the first constant matrix **C1** can have the same dimensions as the addition result matrix **A1** [160, 512] having a value "0" for each element. The maximum result matrix **MX1** of the first activation function MAX1 can be a result matrix of the first layer 510.

[047] In **FIG. 5A**, a second layer 520 of the neural network includes a second multiplication operation MUL2, a second addition operation ADD2, and a second activation function MAX2. The computation of the second layer 520 is similar to that of the first layer 510 except that the result matrix **MX1** of first layer 510 is used as an input operand for the second multiplication operation MUL2, and thus detailed explanation for the second layer 520 is omitted. The second maximum result matrix **MX2** of the second activation function MAX2 can be a result matrix of the second layer 520 as well as the neural network of **FIG. 5A**. The result matrix **MX2** is collected through a tuple operation and is, in turn, fed to a root node.

[048] To process the neural network model of **FIG. 5A** on an accelerator, a compiler can generate an instruction set. In some embodiments, a compiler may be a program or computer software that transforms computer code written in one programming language into accelerator

instructions to create an executable program. The compiler can generate a set of instructions configured to access data from a storage medium, execute a desired operation on the accessed data, generate output data based on the operation, and store the generated output data back into the storage medium for subsequent processing. The instructions can also include assigning a characteristic to the input and the output data. The characteristic of the data can be private, shared, restricted, or the like.

[049] The disclosed embodiments introduce a new set of instructions suitable for executing matrix-based operations on accelerators. The set of instructions will be described with reference to **FIG. 5A**, in accordance with embodiments of the disclosure. **FIG. 5B** to **FIG. 5D** show an example of a set of instructions for executing a neural network of **FIG. 5A**. As further explained in detail below, the instructions in the aforementioned set of instructions generally comprise an operation on the data, characteristics of the data, and a target location within the storage medium. In some embodiments, operations on the data include loading operations (e.g., reading), storing operations (e.g., writing), arithmetic operations (e.g., addition, subtraction, multiplication, division), copying operations, pasting operations, and the like. Characteristics of the data can refer generally to the accessibility of the data within the storage medium. Characteristics of the data can include private, shared, restricted, allowed, global, local, or combinations thereof. Data, in general, is referred to as an operand. Data can be an input operand, for example, a first input matrix **D1**, a first weight matrix **W1**, and a first bias vector **B1** in **FIG. 5A**

[050] **FIG. 5B** and **FIG. 5C** show a first set of instructions for executing the first layer 510 of **FIG. 5A.** In the first set of instructions *i1-i22*, operations and operands are identified on a matrix basis. In some examples, when an input matrix is too big, a compiler can split the input

matrix into sub-matrices and generate instructions to process corresponding operations based on the sub-matrices. A size of a sub-matrix can be determined by a hardware capability of an accelerator. For example, in the first set of instructions *i1-i22*, the first weight matrix **W1** [384, 512] is split into four sub-matrices as shown in instructions *i3*, *i8*, *i13*, and *i18*, and thus same operations are repeated four times for each sub-matrix of the first weight matrix **W1**. The set of instructions will be explained in detail by referring to **FIG. 5A.**

[051] First, an instruction *i1* defines a matrix loading operation MatLoad specifying that a first bias vector **B1**[1, 512] is loaded from a first memory module 222 to a second memory module 223. For example, the first bas vector **B1** is loaded to a corresponding memory block of the second memory module 223. In the instruction *i1*, *sub{1,1;1,512;%B1[1,512]* indicates that a portion of the bias vector **B1** is selected to be loaded. A term "sub" is used to represent a submatrix access to a matrix stored in memory device. For example, a submatrix of the bias vector **B1** stored in a first memory module 222 is accessed and loaded to a second memory module 223. Here, the selected portion to be loaded has the same size with the bias vector **B1**. A range for the portion of the bias vector **B1** is defined by "1,1;1,512." The loaded bias vector **B1** is represented as a submatrix R3 stored in a corresponding memory block of second memory module 223 and has the dimensions of [1, 512] in the instruction *i1*. In order for an accelerator to process data, the data is loaded and stored in the on-chip memory module beforehand.

[052] Similarly, an instruction *i2* specifies that a first input matrix **D1**[160, 384] is loaded from a first memory module 222 to a second memory module 223. Here, the loaded input matrix **D1** is indicated as a submatrix **R5** stored in a corresponding memory block of second memory module 223 and has the dimensions of [160, 384]. An instruction *i3* specifies that a portion of a first weight matrix **W1** [384, 512] is loaded from a first memory module 222 to a

second memory module 223. In the instruction *i3*, the selected portion to be loaded has a range from an element at 1 by 1 position to an element at 384 by 128 position, therefore, the loaded portion has dimensions of [384, 128]. In the instruction *i3*, only a first part (first ¼ part) of the first weight matrix **W1** is loaded to a corresponding memory block of the second memory module 223 for processing. Here, the loaded weight submatrix of the first weight matrix **W1** is indicated as **R6**[384, 128].

[053] In some embodiments of the present disclosure, a data loading operation and a data storing operation can be performed on a matrix basis. Reference will be made to **FIG. 6** to explain a matrix loading operation in detail with respect to the instructions *i2* and *i3*. **FIG. 6** illustrates an example for loading data on a matrix basis according to a matrix loading instruction, consistent with embodiments of the present disclosure.

[054] In **FIG. 6**, a submatrix **R5**[160, 384] is loaded from a first memory module 222 to a second memory module 223. In a first memory module 222, a submatrix **R5** [160, 384] corresponds to an entire first input matrix **D1**[160, 384] stored in a first memory block 611 of the first memory module 222. The selected portion to be loaded in the first input matrix **D1**[160, 384] is indicated by a reference number 621 and a dotted box. In some embodiments of the present disclosure, the first input matrix **D1**[160, 384] is stored in a memory block 611 having a memory cell array arranged in a matrix form and each memory cell can have an address index corresponding to an element of the corresponding matrix. In **FIG. 6**, each memory cell is represented as a square box comprising a corresponding element of the first input matrix **D1**[160, 384]. In some embodiments, elements of a first input matrix **D1**[160, 384] may be physically or logically arranged in a multi-dimensional memory cell array having 160 rows and 384 columns. That is, each element of the first input matrix **D1**[160, 384] is stored in a memory cell of a

corresponding position in the memory cell array arranged in a matrix form having the same dimension with first input matrix **D1**[160, 384]. In the disclosed embodiments, matrices are stored in the first and second memory modules 222 and 223 in a similar manner as illustrated regarding the first input matrix **D1**.

[055] To load the submatrix **R5** out of a first input matrix **D1**[160, 384] to the second memory module 223, the submatrix **R5** is identified by an address index of a top-left element of the submatrix **R5** in the first input matrix **D1** and an address index of a bottom-right element of the submatrix **R5** in the first input matrix **D1**. Here, the top-left element of the submatrix **R5** is $D1_{1,1}$ shaded in **FIG. 6**, and the bottom-right element of the submatrix **R5** is $D1_{160,384}$ also shaded. The submatrix **R5** can include all the elements included in an area bounded by the row numbers 1 and 160 and column numbers 1 and 384. For example, the submatrix **R5** can be defined as *sub{1,1;160,384;%D1[160,384]}* as in an instruction *i2*, consistent with embodiments of the present disclosure. Similarly, when considering a multi-dimensional matrix, the range of the submatrix **R5** is defined by a first address index (e.g., 1,1) for a first element of the submatrix **R5** in the first input matrix **D1** and a second address index (160, 384) for a second element of the submatrix **R5** in the first input matrix **D1**. Here, the first element is positioned at one corner of the submatrix **R5** and the second element is positioned at another corner farthest from the first element of the submatrix **R5**.

[056] By a matrix loading operation, the submatrix **R5** can be loaded to a first memory block 631 of the second memory module 223. As a result, the submatrix **R5** is stored in a first memory block 631 having a memory cell array having the same dimensions as the submatrix **R5**, as explained regarding the first input matrix **D1**. Similarly, a submatrix **R6** is loaded from a memory block 612 of a first memory module 222 to a second memory block 632 of the second

memory module 223. Here, the submatrix **R6** [384, 128] loaded into second memory block 632 is a part of a first weight matrix **D1** [384, 512] and can be defined as *sub{1,1;384, 128;%D1[384, 512]}* as provided in instruction *i3*, consistent with embodiments of the present disclosure. After loading the submatrix **R5** and submatrix **R6** to corresponding memory blocks 631 and 632 of a second memory module 223, an accelerator can process operations based on the loaded submatrices **R5** and **R6**.

[057] Referring back to instructions in **FIG. 5B** and **FIG. 5C**, instructions *i4* to *i7* specify the computations of the first layer 510 of **FIG. 5A** based on the loaded matrices from the instructions *i1* to *i3*. Each of instructions *i4* to *i6* defines a matrix multiplication instruction MatMul, which performs a partial multiplication operation of the first multiplication operation MUL1 of **FIG. 5A.** A term "MatMul" represents a matrix multiplication instruction to multiply listed matrices and produce a multiplication output. This is because the loaded input submatrix **R5** of the first input matrix **D1** and the loaded weight submatrix **R6** of the first weight matrix **W1** by the instructions *i2* and *i3* are too big for a processing unit to process. Thus, each of the loaded input submatrix **R5** and the loaded weight submatrix **R6** is split into 3 sub-matrices and the multiplication operation is performed for each of the 3 submatrices.

[058] In the instruction *i4*, *sub{1,1;160,128;%R5[160,384]}* indicates that a range from an element at 1 by 1 position to an element at 160 by 128 position among the loaded input submatrix **R5** is selected for the first multiplication operation MUL1. In this instruction, *sub{1,1;128,128; %R6[384,128]}* indicates that a range from an element 1 by 1 position to an element at 128 by 128 position among the loaded weight submatrix **R6** is selected for the first multiplication operation MUL1. "*const 0*" indicates that a result of the multiplication instruction MatMul based on the selected submatrices of the loaded input submatrix **R5** and weight

submatrix **R6** is added to a matrix filled with a constant value "0." Here, the result matrix of the matrix multiplication instruction MatMul is indicated as a result submatrix **R7**[160, 128].

[059] Similarly, in the instruction *i5*, *sub{1,129;160,256;%R5[160,384]}* indicates that a range from an element at 1 by 129 position to an element at 160 by 256 position among the loaded input submatrix **R5** is selected for the matrix multiplication instruction MatMul. In this instruction, *sub{129,1;256,128; %R6[384,128]}* indicates that a range from an element 129 by 1 position to an element at 256 by 128 position among the loaded weight submatrix **R6** is selected for the matrix multiplication instruction MatMul. *R7[160,128]* at the end of the instruction *i5* indicates that a result of the matrix multiplication instruction MatMul based on the selected submatrices of the loaded input submatrix **R5** and weight submatrix **R6** is added to the result submatrix **R7**[160,128] of the instruction *i4*. Here, the result submatrix of the matrix multiplication instruction MatMul is indicated as a result submatrix **R7**[160, 128], which incorporates the result of the instruction *i4*.

[060] Similarly, in the instruction *i6*, *sub{1,257;160,384;%R5[160,384]}* indicates that a range from an element at 1 by 257 position to an element at 160 by 384 position among the loaded input submatrix **R5** is selected for the matrix multiplication instruction MatMul. In this instruction, *sub{257,1;384,128; %R6[384,128]}* indicates that a range from an element 257 by 1 position to an element at 384 by 128 position among the loaded weight submatrix **R6** is selected for the matrix multiplication instruction MatMul. *R7[160,128]* at the end of the instruction *i6* indicates that a result of the matrix multiplication instruction MatMul based on the selected submatrices of the loaded input submatrix **R5** and weight submatrix **R6** is added to the result submatrix **R7**[160,128] of the instruction *i5*. Here, the result submatrix of the matrix

multiplication instruction MatMul is indicated as a result submatrix **R7**[160, 128], which incorporates the results of the instructions *i4* and *i5*.

[061] The instruction *i7* defines a matrix miscellaneous instruction MatMisc. A matrix miscellaneous instruction MatMisc instructs a corresponding processing unit to perform an activation function or other miscellaneous functions. In some embodiments, inputs for a matrix miscellaneous instruction MatMisc can come from a matrix multiplication instruction MatMul. Here, the outputs from a matrix multiplication instruction MatMul can be fed to a matrix miscellaneous instruction MatMisc through a FIFO (First In First Out) buffer in the second memory module 223. In some embodiments, the outputs from a matrix multiplication instruction MatMul can be first stored in a corresponding memory block of the second memory module 223 in a matrix form and then fed to a matrix miscellaneous instruction MatMisc. In this example, the matrix miscellaneous instruction MatMisc includes addition operation and maximum operation as an activation function.

[062] In the instruction *i7*, *sub{1,1;1,128; %R3[1,512]}* indicates that a range from an element 1 by 1 position to an element at 1 by 128 position among the loaded bias sub-vector **R3** is selected for an addition operation ADD1. In this instruction, *R7[160,128]* indicates that the result of the matrix multiplication instruction MatMul from the instruction *i6* is added to the selected portion of the loaded bias sub-vector **R3**. To perform an addition operation, the selected portion of the loaded bias sub-vector **R3** can be reshaped to have the same dimensions with the result submatrix **R7**[160, 128] of the instruction *i6*. This reshaping can be performed by copying the selected portion of the loaded bias sub-vector **R3** multiple times. For example, a data preparation instruction MatPrep can be used to instruct reshaping or reformatting of data into different multidimensional matrix shapes. The shape of matrix can be determined based on the

target device capability and corresponding operations. The detailed instruction regarding the broadcast operation is omitted for simplicity. In instruction *i7*, *sub{0;%constDDR}* indicates a constant matrix filled with a constant value "0" and having the same dimensions (here, [160, 128]) with the result submatrix of the first addition operation ADD1 of the result submatrix **R7** and the selected portion of the loaded bias sub-vector **B1**. Then a maximum operation MAX1 is performed between the constant matrix **C1** and the result submatrix of the first addition operation ADD1. The result submatrix of the miscellaneous instruction MatMisc is represented as *sub{1,1;160,128;%R4[160,512]}*, which indicates that a range from an element at 1 by 1 position to an element at 160 by 128 position among the first maximum result matrix **MX1** [160, 512] is generated and the generated result submatrix is stored in a corresponding position in a memory block (here, the memory block is designated for a whole result matrix **R4**) corresponding to the first maximum result matrix **MX1**.

[063] As explained above, the instructions *i3* to *i7* are directed to operations based on a first submatrix part (first ¼ part) of the first weight matrix **W1**. Instructions *i8* to *i12* is directed to operations based on a second submatrix part (second ¼ part) of the first weight matrix **W1**. That is, in instruction *i8*, *sub{1,129;384,256;%W1[384,512]}* indicates that a range from an element at 1 by 129 position to an element at 384 by 256 position among the first weight matrix **W1** is loaded to a memory block of a second memory module 223 from a first memory module 222. Here, the loaded weight submatrix of the first weight matrix **W1** is indicated as submatrix **R8**[384, 128]. The instructions *i4* to *i7* are similarly repeated in the instructions *i9* to *i12* to perform the similar computations regarding the second submatrix part (second ¼ part) of the first weight matrix **W1**. Instructions *i3* to *i7* are similarly repeated in instructions *i13* to *i17* to perform similar operations regarding a third submatrix part (third ¼ part) of the first weight

matrix **W1**. Instructions *i3* to *i7* are similarly repeated in instructions *i18* to *i22* to perform similar operations regarding a fourth submatrix part (fourth ¼ part) of the first weight matrix **W1**. The duplicated explanation regarding the instructions *i8* to *i22* is omitted for simplicity.

[064] As shown in the set of instructions, the instructions are defined on a matrix basis. The instructions are generated to perform operations based on matrices of which sizes are supported by accelerator hardware. For example, a processing unit supports operations on a matrix size not more than [128, 128], and thus the instructions are generated to perform operations based on a smaller size matrix after splitting a bigger matrix into submatrices. For example, in the set of instructions above, the weight matrix **W1** is split into four sub-matrices and the similar operations are repeated four times to compute the whole result for the first layer 510. In addition, for each iteration, the matrix multiplication instruction MatMul is repeated three times to compute the whole result for the first matrix multiplication MUL1 for the corresponding submatrix of the weight matrix **W1**. In some embodiments, the matrix size can also be determined based on hardware resource usage such as available memory capacity of a first memory module 222 and second memory module 223.

[065] In some embodiments of the present disclosure, one processing unit (e.g., PU1 in **FIG. 3**) can process one or more matrix operations among the set of instructions *i1* to *i22*. For example, a first processing unit PU1 can sequentially execute instructions *i4* to *i7*. As indicated in the instructions, the intermediate result matrix of the instruction or operation can be stored in a proper block of a second memory module 223 and the intermediate result matrix can be reused for a subsequent instruction or operation. Here, the proper block for storing the intermediate result matrix can be predetermined by a compiler when generating the set of instructions, and a corresponding instruction can instruct a processing unit to store the intermediate result matrix

into the predetermined block. Compiler can also predetermine an address for a memory cell in which each element of the intermediate result matrix is designated to be stored. Therefore, a processing unit can know where to store the intermediate result matrix and where to find the intermediate result matrix for reuse. For example, the intermediate result submatrix **R7**[160, 128] of the instruction *i4* is stored in a corresponding memory block of a second memory module 223 and is used for an addition operation according to the subsequent instruction *i5*. Similarly, the result matrix of the instruction *i7* is a submatrix of the first maximum result matrix **MX1** (indicated as result matrix **R4**[160, 512] in the instruction *i7*) having a dimension [160, 128] and stored in a memory block of a second memory module 223. In this way, one processing unit PU1 can compute matrix-based operations in a pipelined way. Each element of the result submatrix of the instruction *i7* is stored in a memory cell having an address index corresponding to a position of an element in the result matrix **R4**[160, 512]. In some embodiments, the first processing unit PU1 can sequentially execute the set of instruction *i4* to *i7*, *i9* to *i12*, *i14* to *i17*, and *i19* to *i22* in a pipelined way. It is also possible that a first processing unit PU1 can execute the instructions *i4* to *i7*, a second processing unit PU2 can execute the instructions *i9* to *i12*, a third processing unit PU3 can execute the instructions *i14* to *i17*, and a fourth processing unit PU4 can execute the instructions *i19* to *i22*. In this case, each result submatrix of the instructions *i7*, *i12*, *i17*, and *i22* can be stored in a corresponding portion of a memory block for the result matrix **R4**[160, 512].

[066] By the instruction *i22*, a result submatrix, which is a submatrix of the result matrix **R4** [160, 512] and includes elements from an element at 1 by 385 position to an element at 160 by 512 position among the result matrix **R4** [160, 512] as represented by *sub{1,385;160,512;%R4[160,512]}*, is generated and stored in a corresponding position in a memory block corresponding to the result matrix **R4**[160, 512] (corresponding to a first

maximum result matrix **MX1**). By storing each resultant submatrix from the instructions *i7*, *i12*, *i17*, and *i22* to a memory cell array having corresponding memory indexes in a memory block of the result matrix **R4**[160, 512], the whole first maximum result matrix **MX1** of the first layer 510 in **FIG.5A** can be obtained. The first maximum result matrix **MX1** can be used again in a computation module 224 for processing a second layer 520 of the neural network of **FIG. 5A** Except the fact that the first maximum result matrix **MX1** is used as an input operand to a second multiplication operation MUL2 of the second layer 520, a second set of instructions for the second layer 520 is similar to the first set of instructions *i1* to *i22* for the first layer 510.

[067] **FIG. 5D**shows a second set of instructions for executing the second layer 520 of **FIG. 5A**As shown in a second set of instructions, the second layer 520 is processed in a similar way as in the first layer 510, and the difference will be mainly explained hereinafter. In the second set of instructions, a second weight matrix **W2** is split into three submatrices and loaded three times in instructions *i24*, *i30*, and *i36*, and similar computations are repeated for each submatrix of the second weight matrix **W2**. For each iteration, a multiplication instruction MatMul is processed four times for a second multiplication operation MUL2 due to the limited capability of a processing unit PU. Here, as an example, the processing unit PU can support a matrix computation for a 128 by 128 size of matrices. In addition, the result matrix **R4**[160, 512] of a first layer 510 is used as an input matrix to a second multiplication operation MUL2 in instructions *i25* to *i28*, *i31* to *i34*, and *i37* to *i40* without loading the input matrix since the result matrix **R4**[160, 512] is already stored in a second memory module 223 where the processing unit PU can access. By storing each resultant submatrix from the instructions *i29*, *i35*, and *i41* to a memory cell array having corresponding address indexes in a memory block of a result matrix **R14**[160, 384]), the second maximum result matrix **MX2** of the second layer 520 in **FIG. 5**can

be obtained. According to an instruction *i42*, the result matrix **R14**[160, 384] of the second layer 520 is stored to a corresponding memory block of a first memory module 222. "MatStore" represents an instruction to move data from a second memory module 223 to a first memory module 222. A matrix storing operation can also be identified on a matrix basis and the storing can be performed in a similar way as illustrated regarding the matrix loading operation. In the example of **FIG.5A**, the first memory module 222 can be implemented as a DDR memory as indicated in the instruction *i42*. Transferring the final result (here, the result matrix **R14**[160, 384] of the second layer 520) of the neural network of **FIG. 5A**to a host device 210 is omitted for simplicity. The final result can be stored in a second memory segment 222B of the first memory module 222 so that the final result of the second layer 520 can be transferred to a host device 210.

[068] In some embodiments, the first processing unit PU1 can sequentially execute instructions *i4* to *i7*, *i9* to *i12*, *i14* to *i17*, and *i19* to *i22*, and the same first processing unit PU1 can sequentially execute the instructions *i25* to *i29*, *i31* to *i35*, and *i37* to *i41* in a pipelined way by using an intermediate result matrix from a previous instruction. When four processing units PU1 to PU4 are used to execute four groups of instructions *i4* to *i7*, *i9* to *i12*, *i14* to *i17*, and *i19* to *i22* for a first layer 510, it is also possible that three processing units can be used to execute three groups of instructions *i25* to *i29*, *i31* to *i35*, and *i37* to *i41* for a second layer 520.

[069] As shown from the first and second sets of instruction, data interaction between the host device 210 and the target device 220 can be reduced according to embodiments of the present disclosure. **FIG. 7A**illustrates a conventional data flow for processing a 3-layer neural network model. **FIG. 7B**illustrates an exemplary data flow for processing a 3-layer neural network model, consistent with embodiments of the present disclosure.

[070] When using a set of instructions on a scalar or vector basis, a result of each layer of the neural network model needs to be transferred to a host device 210. In a conventional process as shown in **FIG. 7A**, before executing an operation, raw data and operation instructions are transferred to a target device 220. After completing the execution of the operation, the target device 220 returns the operation result to a host device 210. For example, to process a 3-layer neural network model, input parameters are loaded from a host device 210 to a target device 220 for each layer via transmissions 701, 704, and 707. Referring back to **FIG. 5A** as an example, the input parameters sent via transmission 701 can be a first input matrix **D1**, a first weight matrix **W1**, and a first bias vector **B1**, and the input parameters sent via transmission 704 can be a second weight matrix **W2** and a second bias matrix **B2**. A first maximum result matrix **MX1** of the first layer 510 is transferred to a host device 210 via transmission 702 and host device 210 stores matrix **MX1** in some manner. Afterwards, because the first maximum result matrix **MX1** of the first layer 510 is used as input operand for the second layer 520, the first maximum result matrix **MX1** needs to be accessed from storage and transferred back to the target device 220 via transmission 703. This data transfer between the devices is performed for each layer process according to conventional instructions as shown in **FIG. 7A**. In a heterogeneous accelerator system, data transfer between the host device 210 and the target device 220 is heavy in view of time and energy, and thus frequent data interaction causes a huge overhead and delay.

[071] According to embodiments of the present disclosure, unnecessary data interaction between the host device 210 and target device 220 can be reduced. **FIG. 7B** illustrates an exemplary data flow for processing a 3-layer neural network model, consistent with embodiments of the present disclosure. In **FIG. 7B**, input parameters are transferred from a host device 210 to a target device 220 for each layer via transmissions 721, 722, and 723. Referring

back to **FIG. 5A** as an example, the input parameters for transmission 721 can be a first input matrix **D1**, a first weight matrix **W1**, and a first bias vector **B1**, and the input parameters for transmission 722 can be a second weight matrix **W2** and a second bias matrix **B2**. Here, first maximum result matrix **MX1** of the first layer 510 is stored in a memory module of a target device 220 such as a second memory module 223 and is used again for executing a second layer 520. Therefore, the first maximum result matrix **MX1** of the first layer 510 is not transferred (nor stored) to host device 210 after executing the first layer 510. Also the first maximum result matrix **MX1** of the first layer 510 is not transferred back to a target device 210 for executing the second layer 520. In **FIG. 7B**, only a final result of the 3-layer neural network model can be transferred to a host device 210 via transmission 724.

[072] As illustrated referring to **FIG. 7B**, embodiments of the present disclosure can reduce data interaction between the host device 210 and target device 220 by generating instructions to integrate operations of multiple layers and to control data flow. For example, a set of instructions can specify that one processing unit processes operations of the three layers in a pipelined way. By preserving intermediate results for a first layer and second layer in a target device 220, the same intermediate results can be reused in a subsequent layer or operation. Particularly, the intermediate results are stored in a matrix form in a memory module and operations are processed on a matrix basis, a computation module 224 can access the data stored in a predetermined address (e.g., determined by a compiler) in a memory module without receiving help from a host device 210.

[073] Further, to solve a performance bottleneck issue in transmitting data between devices, three-step optimization flow in a target device can be performed when generating a set of instructions. First, if an on-chip memory such as a second memory module 223 is large

enough, unnecessary matrix load and store instruction MatLoad and MatStore can be removed by preserving data in the on-chip memory, which enables executing corresponding operations or instructions based on on-chip data interactions within a target device 220. Second, if an off-chip memory such as a first memory module 222 is large enough, the off-chip memory can be used to store data from the on-chip memory using a matrix storing instruction MatStore. Third, if the on-chip memory and off-chip memory are not large enough, data interaction between a host device 210 and a target device 220 is used to achieve functionality.

[074] **FIG. 8**illustrates an exemplary flow diagram for executing a neural network model in a neural network accelerator system, consistent with embodiments of the present disclosure. Here, it is illustrated that a neural network model of **FIG. 5A** is executed in a neural network accelerator system 200 in **FIG. 2**as an example. A host device 210 receives runtime environment information of a target device on which the neural network model is executed at step S801. The runtime environment can include available spaces of an on-chip memory module 223 and off-chip memory module 222, power consumption and cost for data transfer between memory modules, power consumption and cost for data transfer between a host device 210 and a target device 220, and available processing units and their capability.

[075] At step S802, a matrix-based instruction set for executing the neural network model on a target device 220 is generated. The instruction set can be generated from a high-level language code for the neural network model. At step S802, optimization can be performed to generate the matrix-based instruction set. In some embodiments, the optimization is performed by considering the runtime environment, performance requirement, etc. In some embodiments, a compiler of the host device 210 generates the instruction set and performs the optimization. When considering the runtime environments of a target device 220, the design is based on a

static memory layout, which is determined optimal. The compiler can obtain the assumption by using Just-In-Time (JIT) compilation. And matrix sizes or dimensions for all operands for being processing in a computation module 222 are also determinable at this step.

[076] In some embodiments, an off-chip memory module (e.g., a first memory module 222) is configured to include separate segments for global data and internal data such as a second and third memory segments 222B and 222C. To meet the requirements, a linear growth distribution technique can be used as long as the off-chip memory's capacity is sufficient. The compiler also optimizes usage of an on-chip memory (e.g., a second memory module 223) through a lifetime calculation technology based on the matrix-based instruction set. Linear growth allocation is also usable for the on-chip memory.

[077] In some embodiments, the compiler can also use the following techniques to maximize hardware resources such as an on-chip memory module and off-chip memory module and to achieve optimal performance of target device 220. First, the compiler statically determines a memory layout for target device 220 and a cost of data transfer such as a time and energy, and uses a proper distribution program to avoid dynamic memory application and releases and to reduce a resulting memory fragmentation. Second, the compiler can use Live Range Analysis for data memory space reuse, thereby non-overlapping active cycles can be assigned to a same memory space. Third, the compiler can apply a register allocation algorithm such as graph coloring for an on-chip memory allocation so that frequently used data is preserved in an on-chip memory to reduce the data transfer cost.

[078] The compiler determines a matrix size or dimension to be processed on a processing unit based on the runtime environment and the optimization results. The compiler also determines where and when to load/store data from/to a first memory module 222 to/from a

second memory module 223. The compiler also determines where and when to store results of each operations. For example, the compiler determines where and when to store a first maximum result matrix **MX1** and where and when to use the same. The compiler determines all the data flow, data store place (or address), and timing and processing unit for reuse of intermediate results. The compiler can also determine which processing unit executes which operation. The compiler can determine that one processing unit can execute multiple operations for multiple layers of a neural network model in a pipelined manner. The compiler generates a set of instructions reflecting the optimization and determined results. Therefore, target device 220 can follow the instructions throughout an execution of the neural network model.

[079] At step S803, the generated instruction set and input parameters for executing the neural network model are sent to a target device 220.

[080] At step S804, the instruction set and input parameters are received and stored in a first memory module 222. In some embodiments, the instruction set is stored in a first memory segment 222A of the first memory module 222 and the input parameters are stored in a second memory segment 222B of the first memory module 222. In some embodiments, the size of instruction set for a neural network model can be huge, and thus the data transfer overhead will be considerable if instructions are transferred from a host device 210 to a target device 220 on demand. Therefore, the instruction set can be transferred to a target device 220 all at once, consistent with embodiments of the present disclosure. It is also possible that the instruction set can be transferred to a target device 220 at a segment basis for a certain blocks of a neural network model. According to the present disclosure, the volume of instructions can be considerably reduced since the instructions are generated on a matrix basis while conventional instruction sets are based on a scalar or vector, which require multiple repetitions of one

instruction on various data. Thereby, the disclosed embodiments can reduce huge instruction communication overhead by avoiding repeated instruction delivery between devices. The disclosed embodiments can also reduce the instruction communication overhead occurred within the target device 220 compared to a conventional instruction set based on a scalar or vector. The disclosed embodiments can also minimize hardware configuration operation.

[081] At step S805, input matrices are loaded from a first memory module 222 to a second memory module 223 on a matrix basis according to instructions of the instruction set. Here, the instruction set defines when and where to load the input matrices. The input matrices can be a part of larger matrices stored in a memory cell arranged in a matrix form in the first memory module 222. The input matrices are stored in a memory cell arranged in a matrix form in the second memory module 223.

[082] At step S806, a computation module 224 can execute matrix operations of a first layer 510. In some embodiments, one processing unit PU1 can execute matrix operations of the first layer 510 including a matrix multiplication. The matrix operations can also include matrix addition operations, activation function operations, and matrix preparation operations. During execution of the first layer 510, intermediate results can be stored in the second memory module 223. Here, the instructions define when and where to store the intermediate result. At step S807, the intermediate result can be stored. Here the intermediate result refers to a result matrix of a first layer 510 of a neural network model. In some embodiments, the intermediate result can be transferred from a second memory module 223 to a first memory module 222 in case that space of the second memory module 223 is limited and the intermediate result is not to be used soon enough. Here, if the intermediate result is to be reused for executing a second layer 520 or a

following instruction, the intermediate result is stored in a third memory segments 223B as local data and is not transferred to a host device 210.

[083] At step S808, the computation module 224 can execute matrix operations of a second layer 520. In some embodiments, one processing unit PU1 that executes matrix operations of the first layer 510 can execute matrix operations of the second layer 520. Here, the result matrix of the first layer 510 can be reused for executing the second layer 520. The target device 220 knows when and where to find the result matrix according instructions for the second layer 520. At this time, the result matrix of the first layer 510 is stored in a second memory module 223. If the result matrix is preserved in a first memory module 222, the result matrix is loaded to a second memory module on a matrix basis.

[084] After a final result is generated from the execution of the neural network model, the result is sent to the host device 210 at step S809. It is illustrated that only two layers for a neural network model are executed on a target device as an example, however, it will be noted that three or more layers can be executed in a pipelined manner in one processing unit. The final result may refer to a result of the last layer of the network model. It is also appreciated that the neural network model for execution here can be a part of a larger neural network model. The host device 210 receives the final result at step S810 and stores the result in its associated memory device.

[085] Embodiments herein include database systems, methods, and tangible non-transitory computer-readable media. The methods may be executed, for example, by at least one processor that receives instructions from a tangible non-transitory computer-readable storage medium. Similarly, systems consistent with the present disclosure may include at least one processor and memory, and the memory may be a tangible non-transitory computer-readable

storage medium. As used herein, a tangible non-transitory computer-readable storage medium refers to any type of physical memory on which information or data readable by at least one processor may be stored. Examples include random access memory (RAM), read-only memory (ROM), volatile memory, non-volatile memory, hard drives, CD ROMs, DVDs, flash drives, disks, registers, caches, and any other known physical storage medium. Singular terms, such as "memory" and "computer-readable storage medium," may additionally refer to multiple structures, such a plurality of memories and/or computer-readable storage media. As referred to herein, a "memory" may comprise any type of computer-readable storage medium unless otherwise specified. A computer-readable storage medium may store instructions for execution by at least one processor, including instructions for causing the processor to perform steps or stages consistent with embodiments herein. Additionally, one or more computer-readable storage media may be utilized in implementing a computer-implemented method. The term "computer-readable storage medium" should be understood to include tangible items and exclude carrier waves and transient signals.

[086] In the foregoing specification, embodiments have been described with reference to numerous specific details that can vary from implementation to implementation. Certain adaptations and modifications of the described embodiments can be made. Other embodiments can be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims. It is also intended that the sequence of steps shown in figures are only for illustrative purposes and are not intended to be limited to any particular sequence of steps. As such, those

skilled in the art can appreciate that these steps can be performed in a different order while implementing the same method.

CLAIMS

1. A computing device for executing a set of instructions for a neural network model, comprising:

a computation module including a plurality of processing units performing at least one operation according to the set of instructions;

a first memory module storing the set of instructions and input parameters for executing the set of instructions; and

a second memory module including a plurality of memory blocks,

wherein the second memory module is configured to store a first matrix of the input parameters loaded from the first memory module in a first memory block of the plurality of memory blocks according to a first instruction of the set of instructions,

wherein the first memory block of the second memory module has a first memory cell array meeting dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

2. The computing device of claim 1, wherein:

the first instruction defines a range of the first matrix out of a second matrix stored in a first memory block of the first memory module to be loaded to the first memory module, and

the first memory block of the first memory module has a second memory cell array meeting dimension of the second matrix, and each element of the second matrix is stored in a memory cell of a corresponding address index in the second memory cell array.

3. The computing device of claim 2, wherein:

the range of the first matrix is defined by a first address index for a first element of the first matrix in the second matrix and a second address index for a second element of the first matrix in the second matrix, and

the first element is positioned at one corner of the first matrix and the second element is positioned at another corner farthest from the first element of the first matrix.

4. The computing device of any one of claims 1-3, wherein:

the second memory module is further configured to store a third matrix of the input parameters loaded from the first memory module in a second memory block of the plurality of memory blocks according to a second instruction of the set of instructions, and

a first processing unit of the plurality of processing units is configured to:

receive the first matrix and the third matrix from the second memory module,

perform a matrix operation based on the first matrix and the third matrix according to a third instruction of the set of instructions, and

send an output matrix of the matrix operation to the second memory module.

5. The computing device of claim 4, wherein the matrix operation includes at least one of a matrix multiplication operation, addition operation or activation function operation.

6. The computing device of any one of claims 1-3, wherein:

the second memory module is further configured to store a third matrix and a fourth matrix of the input parameters loaded from the first memory module according to a second instruction and third instruction of the set of instructions, and

a first processing unit of the plurality of processing units is configured to:

receive the first matrix, the third matrix, and the fourth matrix from the second memory module,

perform a matrix multiplication operation between the first matrix and the third matrix according to a fourth instruction of the set of instructions,

perform an addition operation between a multiplication result matrix from the matrix multiplication operation and the fourth matrix according to a fifth instruction of the set of instructions,

perform an activation function operation on an addition result matrix, and

send an output matrix of the activation function operation to the second memory module.

7. The computing device of claim 1 or claim 2, wherein:

the neural network model comprises two or more layers,

the second memory module is configured to store on a matrix basis a part of the input parameters for executing the neural network model,

a first processing unit of the plurality of processing units is configured to:

receive the part of the input parameters,

execute instructions among the set of instructions corresponding to a first layer of the two or more layers,

execute instructions among the set of instructions corresponding to a second layer of the two or more layers by using the first output result, and

send an output matrix of the neural network model to the second memory module.

8. The computing device of claim 7, wherein:

the first processing unit is further configured to:

send a first intermediate result of the first layer after executing the instructions corresponding to the first layer to the second memory module,

fetches the first intermediate result from the second memory module for executing the instructions corresponding to the second layer.

9. The computing device of any one of claims 1-8, wherein:

the second memory module comprises a first segment and a second segment,

the first segment is configured to store data that is determined to interact with an off-chip device, and

the second segment is configured to store data that is determined to be reused by the computation module within a preset time period.

10. The computing device of any one of claims 1-9, wherein the set of instructions includes at least one of a matrix-based loading instruction, a matrix-based storing instruction, or a matrix-based operation instruction.

11. A method for executing a set of instructions for a neural network in a computing device comprising a computation module including a plurality of processing units performing at least one operation according to the set of instructions, a first memory module storing the set of instructions and input parameters for executing the set of instructions; and a second memory module including a plurality of memory blocks, the method comprising:

loading a first matrix of the input parameters from the first memory module to the second memory module according to a first instruction of the set of instructions,

wherein the first matrix is stored in a first memory block of the plurality of memory blocks, the first memory block of the second memory module has a first memory cell array meeting dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

12. The method of claim 11, wherein:

the first instruction defines a range of the first matrix out of a second matrix stored in a first memory block of the first memory module to be loaded to the first memory module, and

the first memory block of the first memory module has a second memory cell array meeting dimensions of the second matrix, and each element of the second matrix is

stored in a memory cell of a corresponding address index in the second memory cell array.

13. The method of claim 12, wherein:

the range of the first matrix is defined by a first address index for a first element of the first matrix in the second matrix and a second address index for a second element of the first matrix in the second matrix, and

the first element is positioned at one corner of the first matrix and the second element is positioned at another corner farthest from the first element of the first matrix.

14. The method of any one of claims 11-13, further comprising:

loading a third matrix of the input parameters from the first memory module to the second memory module according to a second instruction of the set of instructions, the third matrix is stored in a second memory block of the plurality of memory blocks,

performing, by a first processing unit of the plurality of processing units, a matrix operation based on the first matrix and the third matrix received from the second memory module according to a third instruction of the set of instructions, and

storing an output matrix of the matrix operation to the second memory module.

15. The method of claim 14, wherein the matrix operation includes at least one of a matrix multiplication operation, addition operation or activation function operation.

16. The method of any one of claims 11-13, further comprising:

loading a third matrix and a fourth matrix of the input parameters from the first memory module to the second memory module according to a second instruction and third instruction of the set of instructions, and

performing by a first processing unit of the plurality of processing units:

a matrix multiplication operation between the first matrix and the third matrix according to a fourth instruction of the set of instructions,

an addition operation between a multiplication result matrix from the matrix multiplication operation and the fourth matrix according to a fifth instruction of the set of instructions, and

an activation function operation on an addition result matrix; and

storing an output matrix of the matrix operation to the second memory module.

17. The method of claim 11 or claim 12, wherein:

the neural network model comprises two or more layers,

the second memory module is configured to store on a matrix basis a part of the input parameters for executing the neural network model, and

further comprising performing by a first processing unit of the plurality of processing units:

executing instructions among the set of instructions corresponding a first layer of the two or more layers,

executing instructions among the set of instructions corresponding to a second layer of the two or more layers by using the first output result, and

sending an output matrix of the neural network model to the second memory module.

18. The method of claim 17, further comprising by the first processing unit:

sending a first intermediate result of the first layer after executing the instructions corresponding to the first layer to the second memory module,

fetching the first intermediate result from the second memory module for executing the instructions corresponding to the second layer.

19. The method of any one of claims 14, 16, and 17, wherein:

the second memory module comprises a first segment and a second segment, further comprising:

transferring the output result from the second memory module to the first segment when the output result is determined to interact with an off-chip device, or

transferring the output result from the second memory module to the second segment when the output result is determined to be reused by the computation module within a preset time period.

20. The method of any one of claims 11-19, wherein the set of instructions includes at least one of a matrix-based loading instruction, a matrix-based storing instruction, or a matrix-based operation instruction.

21. A non-transitory computer readable medium that stores a set of instructions that is executable by a computing device to cause the computing device to perform a method for executing a neural network, the computing device comprising a computation module including a plurality of processing units performing at least one operation according to the set of instructions, a first memory module storing input parameters; and a second memory module including a plurality of memory blocks, the method comprising:

loading a first matrix of the input parameters from the first memory module to the second memory module according to a first instruction of the set of instructions,

wherein the first matrix is stored in a first memory block of the plurality of memory blocks, the first memory block of the second memory module has a first memory cell array meeting dimensions of the first matrix, and each element of the first matrix is stored in a memory cell having a corresponding address index in the first memory cell array.

22. The computer readable medium of claim 21, wherein:

the first instruction defines a range of the first matrix out of a second matrix stored in a first memory block of the first memory module to be loaded to the first memory module, and

the first memory block of the first memory module has a second memory cell array meeting dimensions of the second matrix, and each element of the second matrix is stored in a memory cell of a corresponding address index in the second memory cell array.

23. The computer readable medium of claim 22, wherein:

the range of the first matrix is defined by a first address index for a first element of the first matrix in the second matrix and a second address index for a second element of the first matrix in the second matrix, and

the first element is positioned at one corner of the first matrix and the second element is positioned at another corner farthest from the first element of the first matrix.

24. The computer readable medium of any one of claims 21-23, wherein the set of instructions that is executable by the computing device to cause the computing device to further perform:

loading a third matrix of the input parameters from the first memory module to the second memory module according to a second instruction of the set of instructions, the third matrix is stored in a second memory block of the plurality of memory blocks,

performing, by a first processing unit of the plurality of processing units, a matrix operation based on the first matrix and the third matrix received from the second memory module according to a third instruction of the set of instructions, and

storing an output matrix of the matrix operation to the second memory module.

25. The computer readable medium of claim 24, wherein the matrix operation includes at least one of a matrix multiplication operation, addition operation or activation function operation.

26. The computer readable medium of any one of claims 21-23, wherein the set of instructions that is executable by the computing device to cause the computing device to further perform:

loading a third matrix and a fourth matrix of the input parameters from the first memory module to the second memory module according to a second instruction and third instruction of the set of instructions, and

performing by a first processing unit of the plurality of processing units:

a matrix multiplication operation between the first matrix and the third matrix according to a fourth instruction of the set of instructions,

an addition operation between a multiplication result matrix from the matrix multiplication operation and the fourth matrix according to a fifth instruction of the set of instructions, and

an activation function operation on an addition result matrix; and

storing an output matrix of the matrix operation to the second memory module.

27. The computer readable medium of claim 21 or claim 22, wherein:

the neural network model comprises two or more layers,

the second memory module is configured to store on a matrix basis a part of the input parameters for executing the neural network model, and

the set of instructions that is executable by the computing device to cause the computing device to further perform by a first processing unit of the plurality of processing units:

executing instructions among the set of instructions corresponding a first layer of the two or more layers,

executing instructions among the set of instructions corresponding to a second layer of the two or more layers by using the first output result, and

sending an output matrix of the neural network model to the second memory module.

28. The computer readable medium of claim 27, wherein the set of instructions that is executable by the computing device to cause the computing device to further perform by the first processing unit:

sending a first intermediate result of the first layer after executing the instructions corresponding to the first layer to the second memory module,

fetching the first intermediate result from the second memory module for executing the instructions corresponding to the second layer.

29. The computer readable medium of any one of claims 24, 26, and 27, wherein:

the second memory module comprises a first segment and a second segment,

the set of instructions that is executable by the computing device to cause the computing device to further perform:

transferring the output result from the second memory module to the first segment when the output result is determined to interact with an off-chip device, or

transferring the output result from the second memory module to the second segment when the output result is determined to be reused by the computation module within a preset time period.

30. The computer readable medium of any one of claims 21-29, wherein the set of instructions includes at least one of a matrix-based loading instruction, a matrix-based storing instruction, or a matrix-based operation instruction.

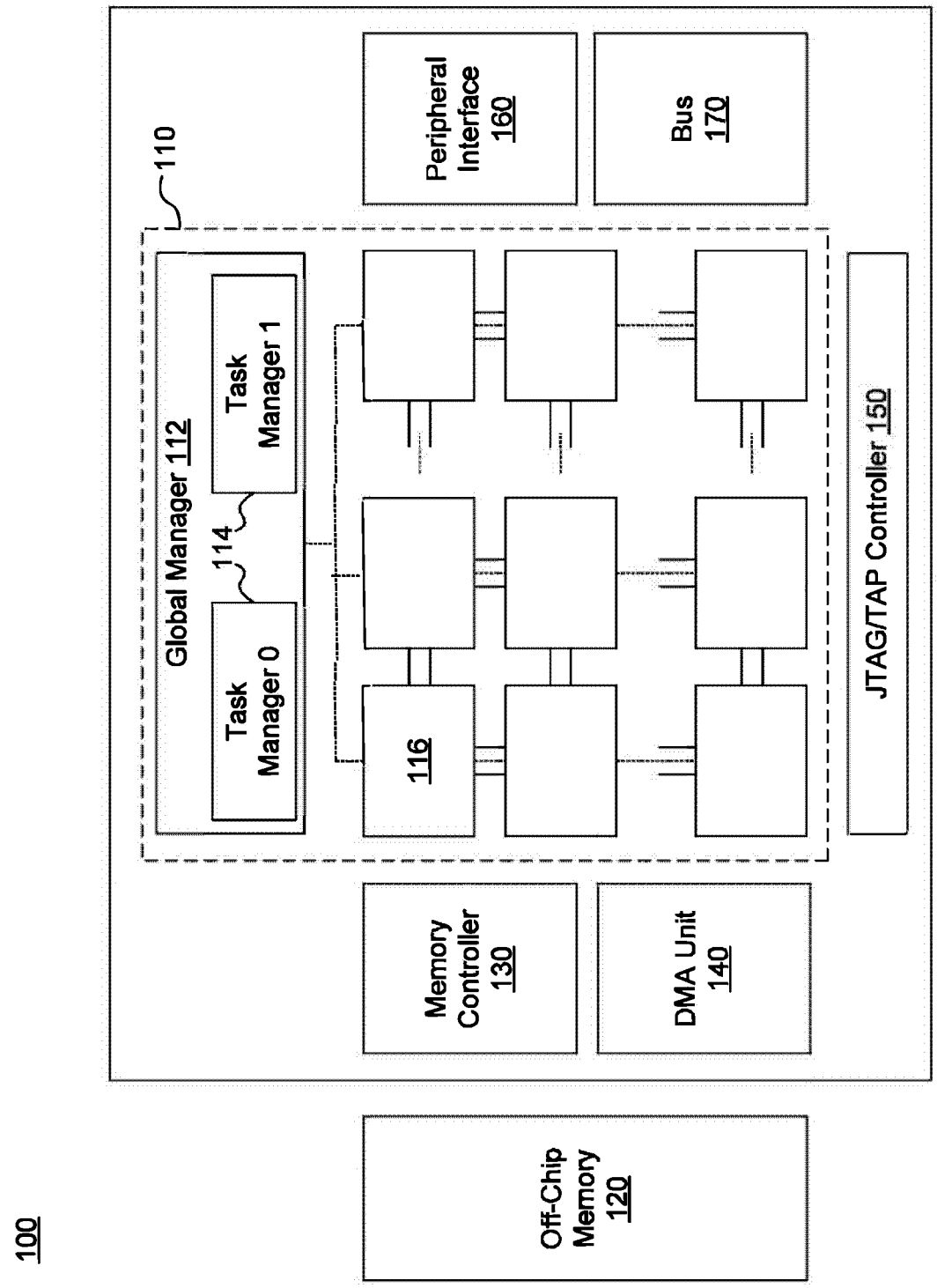
100
Off-Chip Memory 120
Memory Controller 130
DMA Unit 140
Peripheral Interface 160
Bus 170
JTAG/TAP Controller 150
110
Global Manager 112
Task Manager 0
Task Manager 1
114
116

FIG. 1

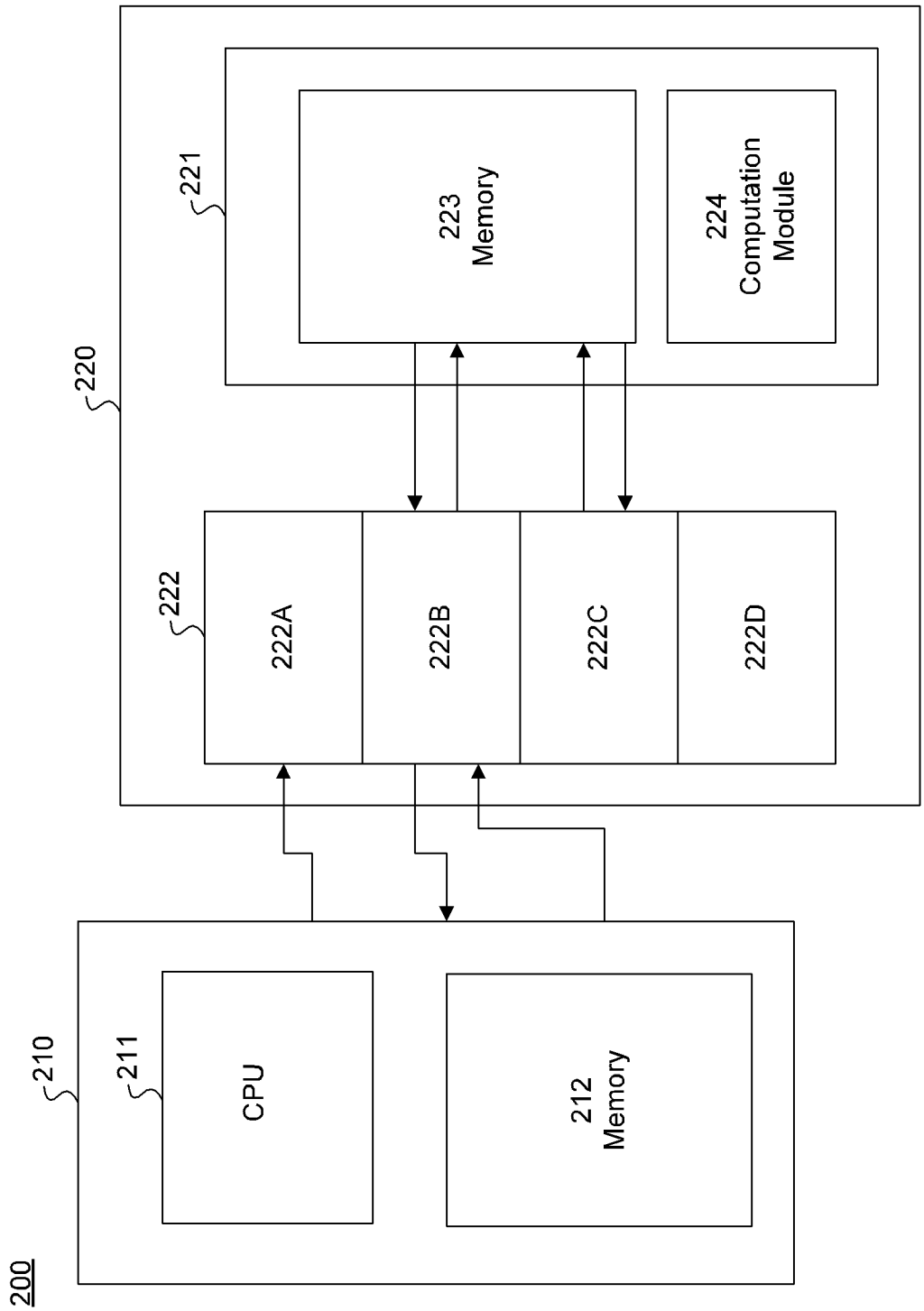
200
210
211
CPU
212
Memory
220
221
223
Memory
224
Computation
Module
222
222A
222B
222C
222D

FIG. 2

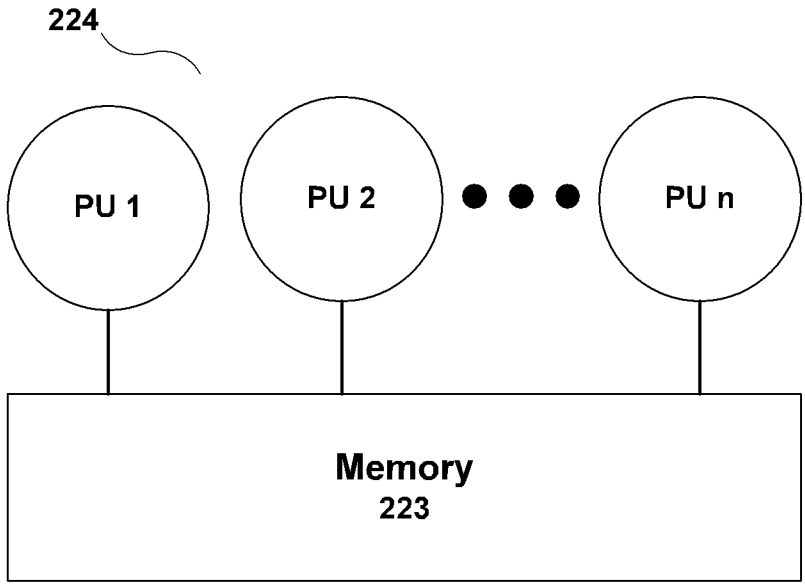

FIG. 3

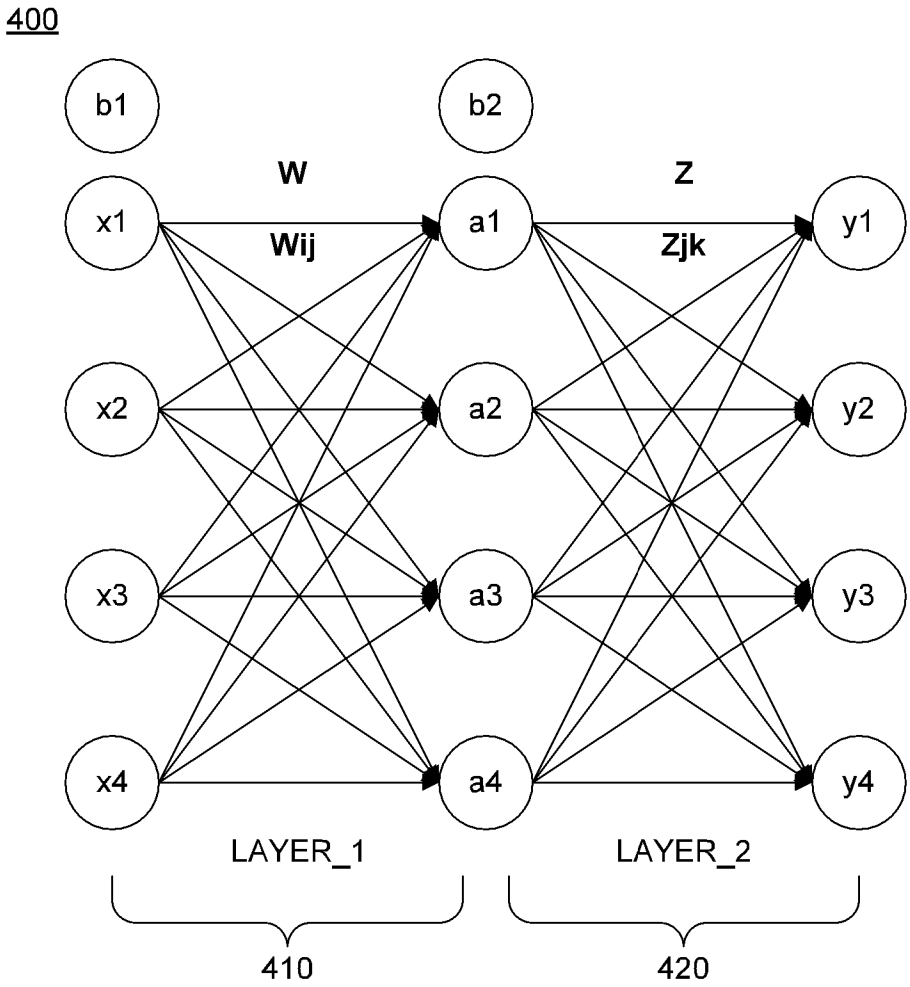
400
b1
b2
W
Z
x1
a1
Wij
Zjk
y1
x2
a2
y2
x3
a3
y3
x4
a4
y4
LAYER_1
LAYER_2
410
420

FIG. 4

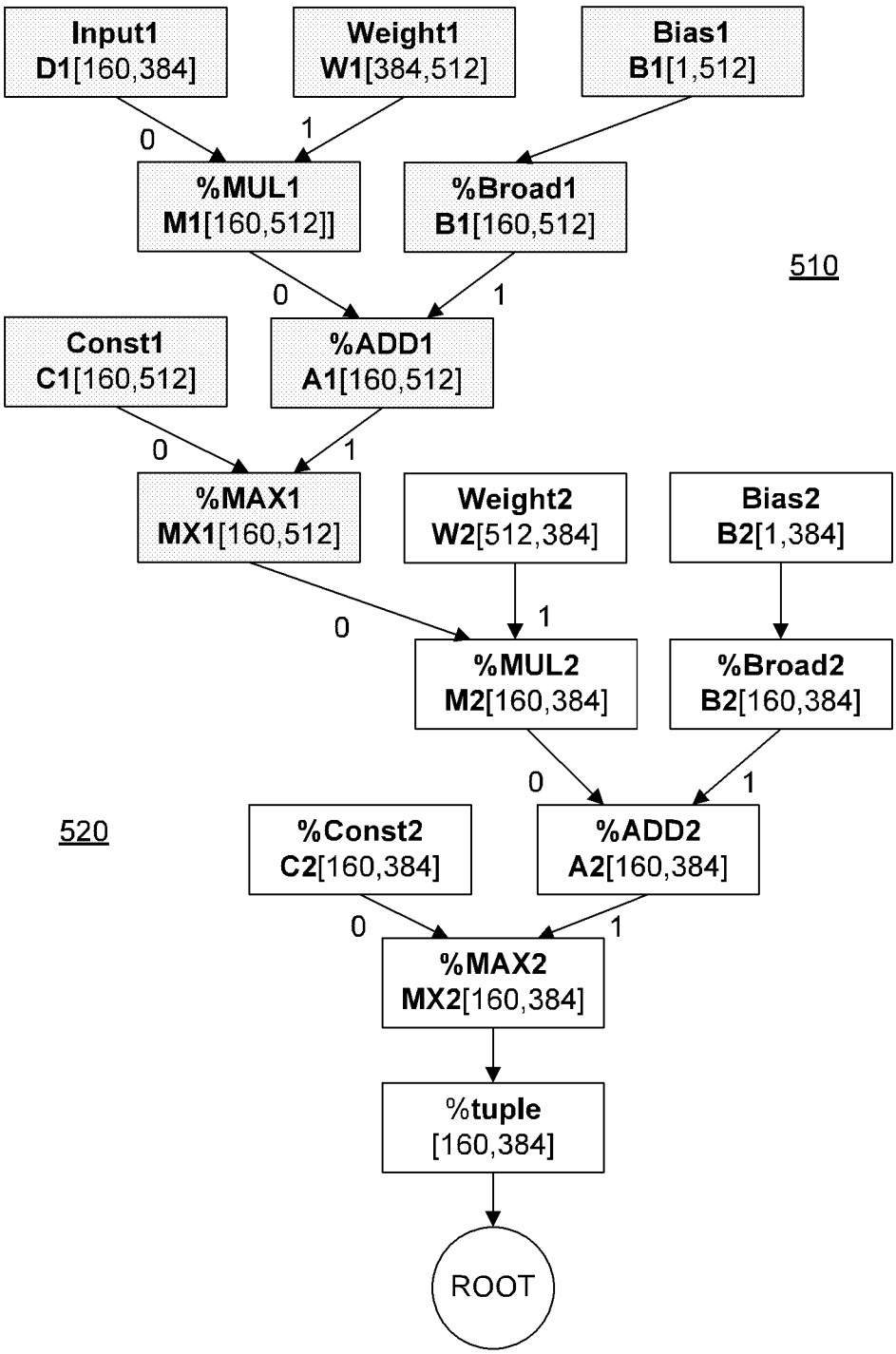
Input1
D1[160,384]
Weight1
W1[384,512]
Bias1
B1[1,512]
0
1
%MUL1
M1[160,512]]
%Broad1
B1[160,512]
510
0
1
Const1
C1[160,512]
%ADD1
A1[160,512]
0
1
%MAX1
MX1[160,512]
Weight2
W2[512,384]
Bias2
B2[1,384]
0
1
%MUL2
M2[160,384]
%Broad2
B2[160,384]
0
1
520
%Const2
C2[160,384]
%ADD2
A2[160,384]
0
1
%MAX2
MX2[160,384]
%tuple
[160,384]
ROOT

FIG. 5A

i1: MatLoad %R3[1, 512] sub{1,1;1,512;%B1[1,512]}

i2: MatLoad %R5[160, 384] sub{1,1;160,384;%D1[160,384]}

i3: MatLoad %R6[384, 128] sub{1,1;384,128;%W1[384,512]}

i4: MatMul %R7[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1;128,128;%R6[384,128]} %const 0

i5: MatMul %R7[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128;%R6[384,128]} %R7[160,128]

i6: MatMul %R7[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128;%R6[384,128]} %R7[160,128]

i7: MatMisc sub{1,1;160,128;%R4[160,512]} %R7[160,128] sub{1,1;1,128;%R3[1,512]} sub{0;%constDDR}

i8: MatLoad %R8[384, 128] sub{1,129;384,256;%W1[384,512]}

i9: MatMul%R9[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1;128,128;%R8[384,128]} %const 0

i10: MatMul %R9[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128;%R8[384,128]} %R9[160,128]

i11:MatMul %R9[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128;%R8[384,128]} %R9[160,128]

i12:MatMisc sub{1,129;160,256;%R4[160,512]} %R9[160,128] sub{1,129;1,256;%R3[1,512]} sub{0;%constDDR}

FIG. 5B

i13: MatLoad %R10[384, 128] sub{1,257;384,384;%W1[384,512]}

i14: MatMul %R11[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1;128,128; %R10[384,128]} %const 0

i15: MatMul %R11[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128; %R10[384,128]} %R11[160,128]

i16: MatMul %R11[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128; %R10[384,128]} %R11[160,128]

i17: MatMisc sub{1,257;160,384;%R4[160,512]} %R11[160,128] sub{1,257;1,384; %R3[1,512]} sub{0;%constDDR}

i18: MatLoad %R12[384, 128] sub{1,257;384,384;%W1[384,512]}

i19: MatMul %R13[160, 128] sub{1,1;160,128;%R5[160,384]} sub{1,1;128,128; %R12[384,128]} %const 0

i20: MatMul %R13[160, 128] sub{1,129;160,256;%R5[160,384]} sub{129,1;256,128; %R12[384,128]} %R13[160,128]

i21: MatMul %R13[160, 128] sub{1,257;160,384;%R5[160,384]} sub{257,1;384,128; %R12[384,128]} %R13[160,128]

i22: MatMisc sub{1,385;160,512;%R4[160,512]} %R13[160,128] sub{1,3854;1,512; %R3[1,512]} sub{0;%constDDR}

FIG. 5C

FIG. 5D

i23: MatLoad %R3[1, 384] sub{1,1;1,384;%B2[1,384]}

i24: MatLoad %R6[512, 128] sub{1,1;512,128;%W2[512,384]}

i25: MatMul %R7[160, 128] sub{1,1;160,128;%R4[160,512]} sub{1,1;128,128;%R6[512,128]} %const 0

i26: MatMul %R7[160, 128] sub{1,129;160,256;%R4[160,512]} sub{129,1;256,128;%R6[512,128]} %R7[160,128]

i27: MatMul %R7[160, 128] sub{1,257;160,384;%R4[160,512]} sub{257,1;384,128;%R6[512,128]} %R7[160,128]

i28: MatMul %R7[160, 128] sub{1,385;160,512;%R4[160,512]} sub{385,1;512,128;%R6[512,128]} %R7[160,128]

i29: MatMisc sub{1,1;160,129;%R14[160,384]} %R7[160,128] sub{1,1;1,128;%R3[1,384]} sub{0;%constDDR}

i30: MatLoad %R8[512, 128] sub{1,129;512,256;%W2[512,384]}

i31: MatMul %R9[160, 128] sub{1,1;160,128;%R4[160,512]} sub{1,1;128,128;%R8[512,128]} %const 0

i32: MatMul %R9[160, 128] sub{1,129;160,256;%R4[160,512]} sub{129,1;256,128;%R8[512,128]} %R9[160,128]

i33: MatMul %R9[160, 128] sub{1,257;160,384;%R4[160,512]} sub{257,1;384,128;%R8[512,128]} %R9[160,128]

i34: MatMul %R9[160, 128] sub{1,385;160,512;%R4[160,512]} sub{385,1;512,128;%R8[512,128]} %R9[160,128]

i35: MatMisc sub{1,129;160,256;%R14[160,384]} %R9[160,128] sub{1,129;1,256;%R3[1,384]} sub{0;%constDDR}

i36: MatLoad %R10[512, 128] sub{1,257;512,384;%W2[512,384]}

i37: MatMul %R11[160, 128] sub{1,1;160,128;%R4[160,512]} sub{1,1;128,128;%R10[512,128]} %const 0

i38: MatMul %R11[160, 128] sub{1,129;160,256;%R4[160,512]} sub{129,1;256,128;%R10[512,128]} %R11[160,128]

i39: MatMul %R11[160, 128] sub{1,257;160,384;%R4[160,512]} sub{257,1;384,128;%R10[512,128]} %R11[160,128]

i40: MatMul %R11[160, 128] sub{1,385;160,512;%R4[160,512]} sub{385,1;512,128;%R10[512,128]} %R11[160,128]

i41: MatMisc sub{1,257;160,384;%R14[160,384]} %R11[160,128] sub{1,257;1,384;%R3[1,384]} sub{0;%constDDR}

i42: MatStore sub{1,1;160,384; %DDR} %R14[160,384]

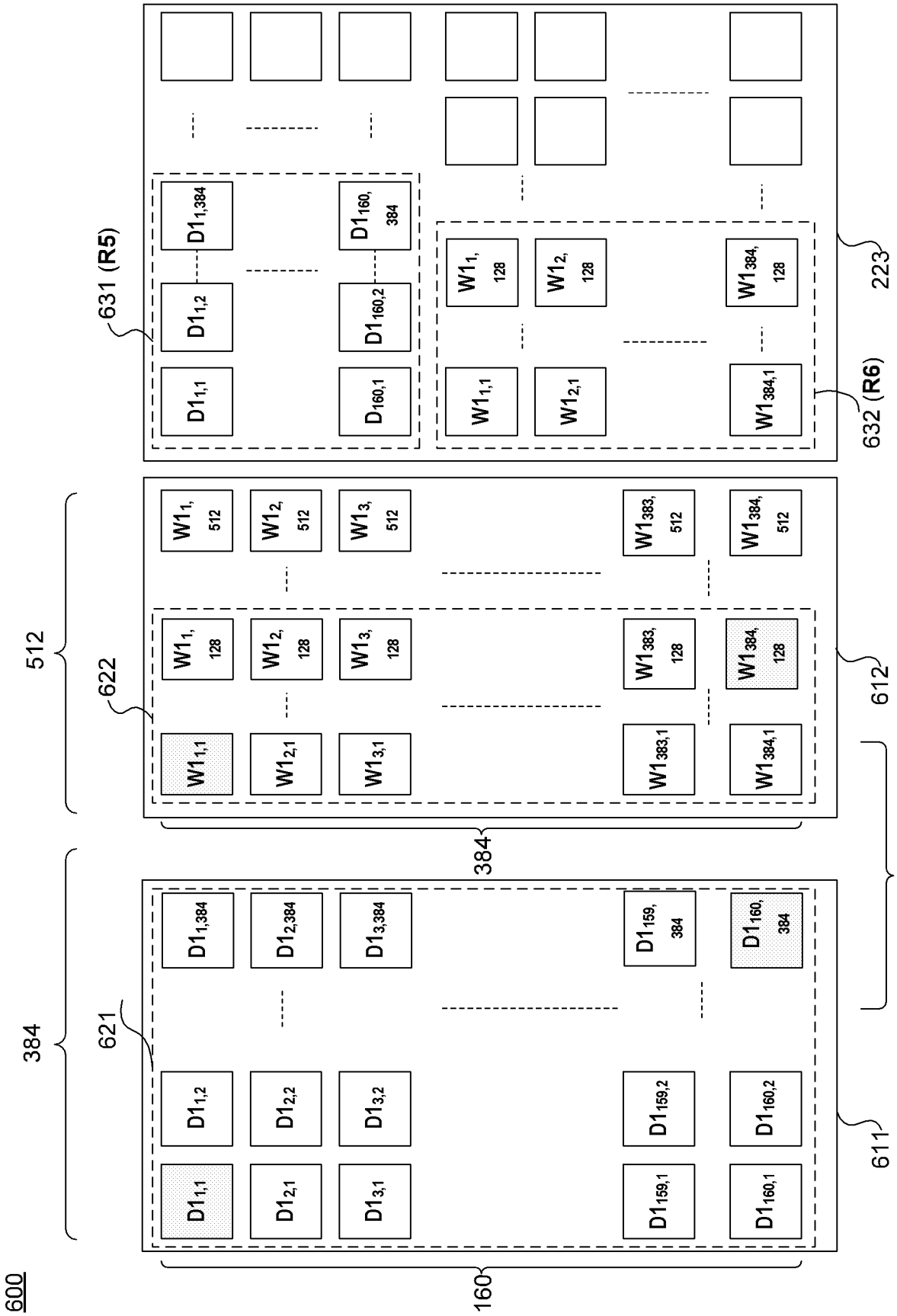
600
160
384
621
D1 1,1
D1 2,1
D1 3,1
D1 159,1
D1 160,1
D1 1,2
D1 2,2
D1 3,2
D1 159,2
D1 160,2
D1 1,384
D1 2,384
D1 3,384
D1 159, 384
D1 160, 384
611
222
384
512
622
W1 1,1
W1 2,1
W1 3,1
W1 383,1
W1 384,1
W1 1, 128
W1 2, 128
W1 3, 128
W1 383, 128
W1 384, 128
W1 1, 512
W1 2, 512
W1 3, 512
W1 383, 512
W1 384, 512
612
631 (R5)
D1 1,1
D1 2,1
D1 160,1
D1 160,2
D1 1,384
D1 160, 384
632 (R6)
W1 1,1
W1 2,1
W1 384,1
W1 1, 128
W1 2, 128
W1 384, 128
223

FIG. 6

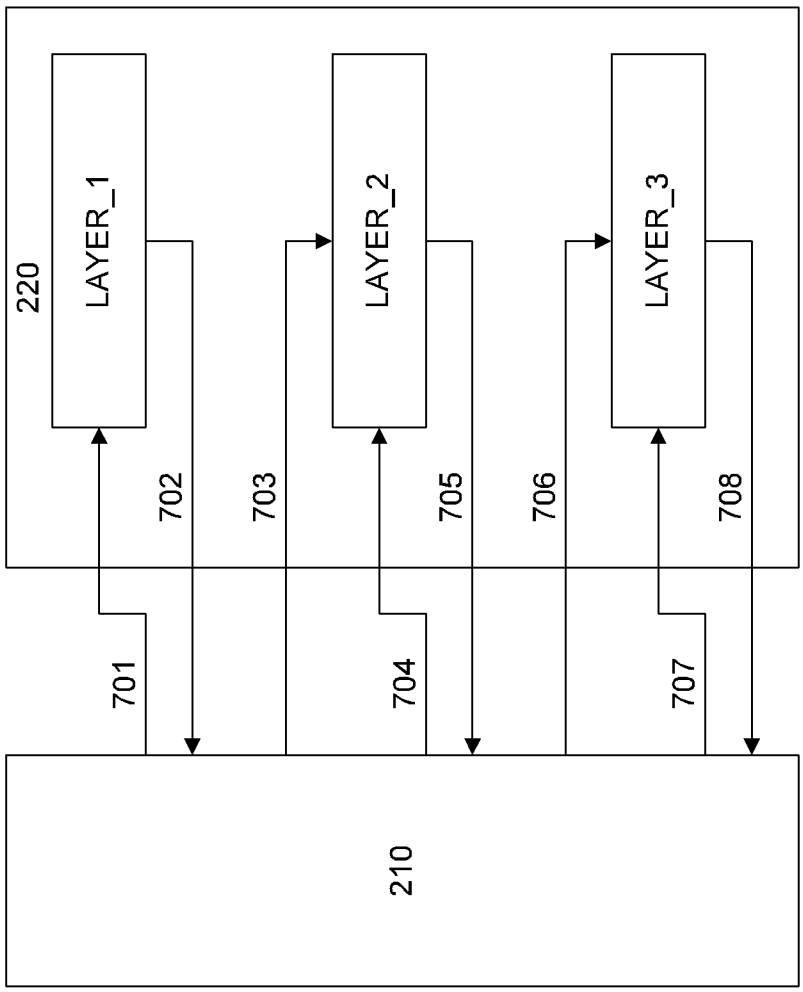

FIG. 7A

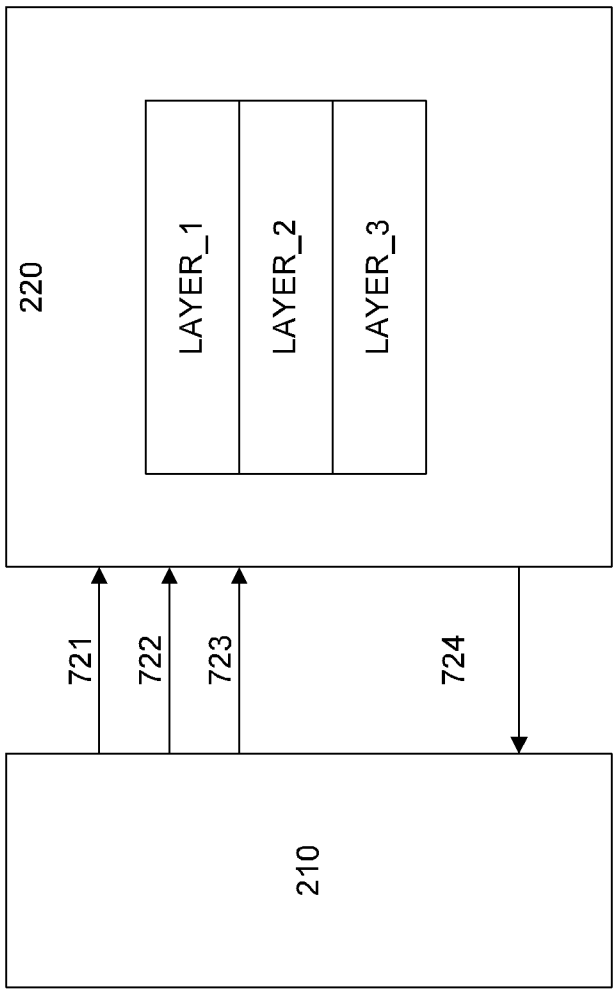

FIG. 7B

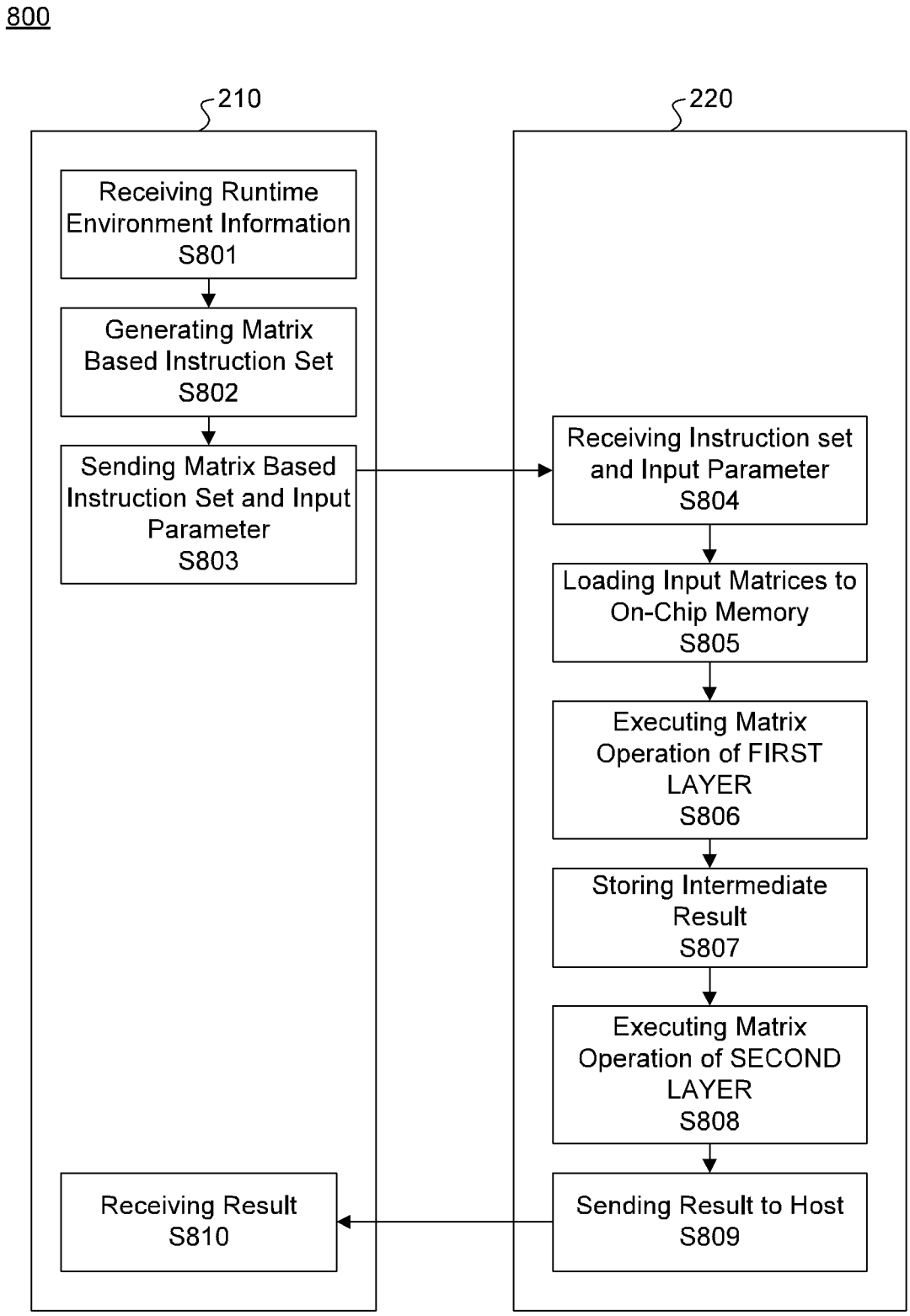
800
210
220
Receiving Runtime Environment Information S801
Generating Matrix Based Instruction Set S802
Sending Matrix Based Instruction Set and Input Parameter S803
Receiving Instruction set and Input Parameter S804
Loading Input Matrices to On-Chip Memory S805
Executing Matrix Operation of FIRST LAYER S806
Storing Intermediate Result S807
Executing Matrix Operation of SECOND LAYER S808
Sending Result to Host S809
Receiving Result S810

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2019/071090

A. CLASSIFICATION OF SUBJECT MATTER

G06N 3/02(2006.01)i

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)

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)

DWPI, CNKI, CNABS, SIPOABS: address, matrix, neural, network, instruction, memory, parameter, block, index

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CN 109034383 A (SHANGHAI CAMBRICON INFORMATION TECHNOLOGY) 18 December 2018 (2018-12-18) the whole document	1-30
A	WO 2018098230 A1 (MASSACHUSETTS INSTITUTE OF TECHNOLOGY) 31 May 2018 (2018-05-31) the whole document	1-30

☐ Further documents are listed in the continuation of Box C.

☑ 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
13 March 2019	**31 May 2019**

Name and mailing address of the ISA/CN	Authorized officer
National Intellectual Property Administration, PRC **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China**	**ZHANG,Tan**
Facsimile No. **(86-10)62019451**	Telephone No. **(86-10)-62411649**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2019/071090

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
CN 109034383 A	18 December 2018	None	
WO 2018098230 A1	31 May 2018	US 2018260703 A1	13 September 2018