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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 June 2020 (11.06.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/113459 A1

(51) International Patent Classification:
G06F 8/41 (2018.01)

(21) International Application Number:
PCT/CN2018/119334

(22) International Filing Date:
05 December 2018 (05.12.2018)

(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). **ZHANG, Dongjie**; 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, 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))*

(54) Title: INTERMEDIATE REPRESENTATION TRANSFORMATION BY SLICE OPERATION HOIST

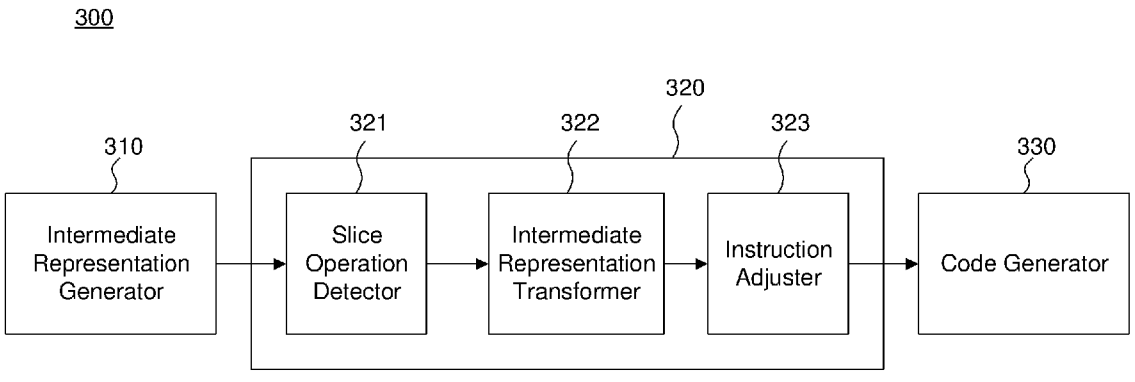

FIG. 3

(57) Abstract: An apparatus (300) for adapting an intermediate representation of a source code. The apparatus (300) comprises a slice operation detector (321) configured to detect a first set of slice operations in the intermediate representation. The intermediate representation includes a plurality of input parameters and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations. The apparatus (300) further comprises an intermediate representation transformer (322) configured to transform the intermediate representation. The transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

WO 2020/113459 A1

INTERMEDIATE REPRESENTATION TRANSFORMATION BY SLICE OPERATION HOIST

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. Operations having discontinuous memory access behaviour often occur in a neural network algorithm, which is inefficient in resource utilization and computation performance. To support these operations and discontinuous memory access behaviour, general-purpose hardware often sacrifices overall performance. On the other hand, dedicated acceleration hardware tries to achieve its maximum performance by eliminating the discontinuous memory access behaviour on chip.

[002] Generally, a machine learning framework can perform optimization such as node clustering and instruction fusing based on underneath hardware in a graph computing phase. Through this optimization, memory operation overhead can be dramatically reduced. Careful instruction fusing improves overall performance of the machine learning process. However, most neural network algorithms are highly likely to include operations having discontinuous memory access behaviour. In these situations, the existing solutions give up the instruction fusing and thus lose optimization opportunity.

SUMMARY

[003] Embodiments of the present disclosure provide an apparatus for adapting an intermediate representation of a source code. The apparatus comprises a slice operation detector configured to detect a first set of slice operations associated with the intermediate representation.

The intermediate representation includes a plurality of input parameters and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations. The apparatus further comprises an intermediate representation transformer configured to transform the intermediate representation. The transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

[004] Embodiments of the present disclosure also provide a method for adapting an intermediate representation of a source code. The method comprises detecting a first set of slice operations associated with the intermediate representation. The intermediate representation includes a plurality of input parameters and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations. The method further comprises transforming the intermediate representation. The transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

[005] Embodiments of the present disclosure also provide a non-transitory computer readable medium that stores a set of instructions that is executable by at least one processor of a computing device to cause the computing device to perform a method for adapting an intermediate representation of a source code. The method comprises detecting a first set of slice operations associated with the intermediate representation. The intermediate representation includes a plurality of input parameters and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations. The method

further comprises transforming the intermediate representation. The transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

[006] The transformation can be performed by determining whether at least one first matrix operation of the first set of one or more matrix operations exists between the first set of slice operations and the plurality of input parameters in the sequence of operations of the intermediate representation, and positioning the second set of slice operations in front of the second set of matrix operations in the sequence of operations in the transformed intermediate representation, wherein the second set of matrix operations correspond to the at least one first matrix operation.

[007] The intermediate representation can be associated with a computation graph, and the transformation can be applied to the computation graph. The computation graph can be a directed acyclic graph. Functionality of the intermediate representation can be maintained after the transformation.

BRIEF DESCRIPTION OF THE DRAWINGS

[008] **FIG. 1** illustrates an exemplary neural network processing unit (NPU) architecture, consistent with embodiments of the present disclosure.

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

[010] **FIG. 3** illustrates a block diagram of exemplary components of a system including an apparatus for adapting an intermediate representation of a source code, consistent with embodiments of the present disclosure.

[011] **FIG. 4** illustrates an example of a LSTM (Long Short-Term Memory) cell architecture.

[012] **FIG. 5** illustrates a first example of a computation graph.

[013] **FIG. 6** illustrates an adapted computation graph of the computation graph of **FIG. 5** after slice operation hoisting, consistent with embodiments of the present disclosure.

[014] **FIG. 7A** illustrates a second example of a computation graph.

[015] **FIG. 7B** illustrates an interim state of the computation graph of **FIG. 7A** after first slice operation hoisting.

[016] **FIG. 8** illustrates an adapted computation graph of the computation graph of **FIG. 7B** after slice operation hoisting, consistent with embodiments of the present disclosure.

[017] **FIG. 9** illustrates an exemplary method for adapting an intermediate representation of a source code, consistent with embodiments of the present disclosure.

[018] **FIG. 10** illustrates an exemplary flow diagram for executing a source code including a method for adapting an intermediate representation of the source code in a neural network accelerator system, consistent with embodiments of the present disclosure.

DETAILED DESCRIPTION

[019] 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.

[020] The disclosed embodiments provide apparatuses and methods for adapting an intermediate representation of a source code. An intermediate representation is a representation of a program between a source code language and a target code language and can be associated with a computation graph (e.g., an Abstract Syntax Tree, Control-Flow Graph, and a Directed Acyclic Graph). An intermediate representation can be the data structure or code used by a compiler or virtual machine to represent a source code without loss of information and may take one of several forms such as an in-memory data structure or a tuple or stack-based code. The disclosed embodiments can resolve aforementioned issues without changing properties of the operations having discontinuous memory access behaviour and thus can be also used for existing neural network accelerator systems.

[021] **FIG. 1** illustrates an exemplary neural network processing unit (NPU) architecture 100. NPU 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, a peripheral component interconnect express (PCIe) interface 160, inter-chip links 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, NPU architecture 100 can include a global memory having on-chip memory blocks (e.g., 4 blocks of 8GB second generation of high bandwidth memory (HBM2)) to serve as main memory.

[022] On-chip communication system 110 can include a global manager 112 and a plurality of cores 116. Global manager 112 can include one or more cluster managers 114 configured to coordinate with one or more cores 116. Each cluster manager 114 can be associated with an array of cores 116 that provide synapse/neuron circuitry for the neural network. For example, the top layer of cores of **FIG. 1** may provide circuitry representing an input layer to neural network, while the second layer of cores may provide circuitry representing a hidden layer of the neural network. As shown in **FIG. 1**, global manager 112 can include two cluster managers 114 configured to coordinate with two arrays of cores 116.

[023] Cores 116 can include one or more processing elements that each includes single instruction, multiple data (SIMD) architecture including 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 122. To perform the operation on the communicated data packets, cores 116 can include one or more processing elements for processing the data packets and at least a buffer or local memory for storing data packets. Each processing element may comprise any number of processing units. In some embodiments, core 116 can be considered a tile or the like.

[024] Off-chip memory 120 can be off-chip memory such as a host CPU's memory. For example, off-chip memory 120 can be a DDR memory (e.g., DDR SDRAM) or the like. Off-chip memory 120 can be configured to store a large amount of data with slower access speed, compared to the on-chip memory integrated within one or more processor.

[025] Memory controller 130 can manage the reading and writing of data to and from a memory block of NPU architecture 100. For example, memory controller 130 can manage read/write data coming from outside chip communication system 110 (e.g., from DMA unit 140

or a DMA unit corresponding with another NPU) or from inside chip communication system 110 (e.g., from a local memory in core 116 via a 2D mesh controlled by a cluster manager of global manager 112). 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. For example, there can be one memory controller for each memory block (e.g., HBM2).

[026] DMA unit 140 can assist with transferring data between off-chip memory 120 and memory corresponding to memory controller 130. DMA unit 140 can 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. The registers can include a memory address register, a byte-count register, and one or more control 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 NPU architecture 100 can include a second DMA unit, which can be used to transfer data between other NPU architectures to allow multiple NPU 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] PCIe interface 160 can support full-duplex communication between any two endpoints, with no inherent limitation on concurrent access across multiple endpoints.

[029] Inter-chip links 170 can connect all the internal components of NPU architecture 100, such as on-chip communication system 110, off-chip memory 120, memory controller 130, DMA unit 140, JTAG/TAP controller 150, and PCIe interface 160 to each other.

[030] While NPU architecture 100 incorporates the embodiments of the present disclosure, it is appreciated that the disclosed embodiments can be applied to chips with SIMD architecture for accelerating some applications such as deep learning. Such chips can be, for example, GPU (Graphics Processing Unit), FPGA (Field Programmable Gate Array), CPU (Central Processing Unit) with vector processing ability, or neural network accelerators for deep learning. SIMD or vector architecture is commonly used to support computing devices with data parallelism, such as graphics processing and deep learning. The SIMD architecture can include multiple processing elements, wherein each of the processing elements can perform the same operation on multiple data points simultaneously.

[031] **FIG. 2** illustrates a block 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 CPU 210 and an accelerator 220 having, for example, the NPU architecture 100 of **FIG. 1**. The accelerator 220 may be connected to the host CPU 201 through a PCIe interface. NPU 221 is the key computing device of the accelerator 220. The host CPU 210 issues workload or commands to the accelerator 220, which performs the computation according to the commands and sends the results back to the host CPU 210. Each of the host CPU 210 and accelerator 220 can be associated with its own memory device. In some embodiments, the

accelerator 220 can be implemented by a heterogenous acceleration chip where processing units do not have equal processing performance with each other.

[032] **FIG. 3** illustrates a block diagram of exemplary components of a system including an apparatus 300 for adapting an intermediate representation of a source code, consistent with embodiments of the present disclosure. The apparatus 300 for adapting an intermediate representation may be implemented within a system. In some embodiments, the system can be a neural network accelerator system 200 of **FIG. 2**. In some embodiments, the apparatus 300 can be implemented in the host CPU 210 side.

[033] As shown in **FIG. 3**, the apparatus 300 can include an intermediate representation generator 310, an optimizer 320, and a code generator 330. The intermediate representation generator 310 can compile a source code to create the intermediate representation. In some embodiments, the intermediate representation can be generated from another high-level code initially compiled from the source code. An intermediate representation may take one of several forms such as an in-memory data structure or a tuple or stack-based code. An intermediate representation can be associated with a computation graph. For example, an Abstract Syntax Tree, Control-Flow Graph, and a Directed Acyclic Graph can be used as an intermediate representation. Here, some embodiments will be explained using a computation graph as an intermediate representation for illustration purposes only and not for restrictive purposes, and thus it is noted that embodiments of the present disclosure can be applied to other forms of intermediate representations than a computation graph.

[034] The optimizer 320 is configured to perform an intermediate representation adaptation, consistent with embodiments of the present disclosure. The optimizer 320 may further be configured to perform an instruction adjustment. The optimizer 320 may perform the

optimization on the intermediate representation by taking runtime environment information of a target machine into account to maximize the performance of the target machine. In other words, the optimizer 320 may perform target specific optimization.

[035] The optimizer 320 can include a slice operation detector 321 and an intermediate representation transformer 322, consistent with embodiments of the present disclosure. The slice operation detector 321 is configured to detect a first set of slice operations in the intermediate representation. The intermediate representation transformer 322 is configured to transform the intermediate representation. Here, the intermediate representation can include a plurality of input parameters and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations. The transformed intermediate representation can include a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations. It is noted that functionality of the intermediate representation is maintained after the transformation. The task that the original intermediate representation is targeted to perform when executed on a target machine can also be performed by executing the transformed intermediate representation on the target machine.

[036] In some embodiments, the optimizer 320 can further include an instruction adjuster 323. The instruction adjuster 323 is configured to adjust instructions to implement the intermediate representation on a target machine such as an accelerator 220, consistent with embodiments of the present disclosure. The instruction adjuster 323 may choose appropriate instructions from the intermediate representation based on the runtime environment of the target machine. Here, the intermediate representation can be the transformed intermediate representation by the intermediate representation transformer 322. The instruction adjuster 323

may perform instruction optimization such as instruction fusing or node clustering to maximize the performance of the accelerator 220.

[037] The code generator 330 is configured to generate a target machine language code from the intermediate representation. The target machine language code is for executing the intermediate representation on the target machine. Sequence of instructions of the target machine language code performs the task as the intermediate representation or the source code would do. The code generator 330 may take the optimized instructions or representations of the intermediate representation and translate them to the target machine language code executable on the target machine. In some embodiments, the target machine can be a neural network accelerator, such as an accelerator 220, consistent with embodiments of the present disclosure.

[038] **FIG. 4** illustrates an example of a LSTM (Long Short-Term Memory) cell architecture. Referring to **FIG. 4**, an operation having discontinuous memory access behaviour will be explained. LSTM has an RNN (Recurrent Neural Network) architecture and is designed to address a vanishing gradient problem. As shown in **FIG. 4**, the LSTM cell's state is split in two vectors $\mathbf{h}_t$ and $\mathbf{c}_t$. Vector $\mathbf{h}_t$ represents a short-term state, while vector $\mathbf{c}_t$. represents a long-term state. As the long-term state vector $\mathbf{c}_{t-1}$ traverses the cell from left to right, $\mathbf{c}_{t-1}$ first goes through a forget gate, dropping some memories, and then adds some new memories (further explained below) via an addition operation adding the memories selected by an input gate. The result $\mathbf{c}_t$ is sent straight out without further transformation. Thereby, at each time step, some memories are dropped and some memories are added. After the addition operation, the long-term state vector $\mathbf{c}_t$ is also copied and passed through a tanh function, and then the result is filtered by an output gate. This produces the short-term state vector $\mathbf{h}_t$, which is the cell's output for this time step.

[039] The creation of new memories involves several steps. First, a previous short-term state vector $\mathbf{h}_{t-1}$ and a current input vector $\mathbf{x}_t$ are fed to four different layers, each of which serves a different purpose. The candidate layer is the one that outputs $\mathbf{g}_t$ and has the role of analysing a weighted current input vector $\mathbf{x}_t$ and a weighted previous short-term state vector $\mathbf{h}_{t-1}$. In an LSTM cell, this layer's output does not go straight out, but instead it is partially stored in the long-term state $\mathbf{c}_t$.

[040] The three other layers are gate controllers (forget gate, input gate, and output gate). They use logistic activation functions (e.g., sigmoid function), and thus their outputs range from 0 to 1. As shown in **FIG. 4**, the three layers' outputs are fed to element-wise multiplication operations, so if they output 0s, they close the gates, and if they output 1s, they open the gates. Specifically, the forget gate, which is controlled by $\mathbf{f}_t$, controls which parts of the long-term state should be erased. The input gate, which is controlled by $\mathbf{i}_t$, controls which parts of $\mathbf{g}_t$ should be added to the long-term state $\mathbf{c}_t$. The output gate, which is controlled by $\mathbf{o}_t$, controls which parts of the long-term state $\mathbf{c}_t$ should be read and output at this time step as $\mathbf{h}_t$ and $\mathbf{y}_t$.

[041] To achieve maximum training performance, weight matrices $\mathbf{W}_h$ and $\mathbf{W}_x$ are multiplied to the inputs $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$. Here, the weight matrices $\mathbf{W}_h$ and $\mathbf{W}_x$ can be different for each of the different gates. For example, weight matrix $\mathbf{W}_{h\text{-}f}$ corresponding to the short-term state vector of the forget gate can be different from weight matrices $\mathbf{W}_{h\text{-}i}$ and $\mathbf{W}_{h\text{-}o}$ corresponding to the short-term state vector of input and output gates. Moreover, weight matrix $\mathbf{W}_{x\text{-}f}$ corresponding to the input vector of the forget gate can be different from weight matrices $\mathbf{W}_{x\text{-}i}$ and $\mathbf{W}_{x\text{-}o}$ corresponding to the input vector of input and output gates.

[042] After the multiplication of the inputs $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ and their corresponding weight matrices $\mathbf{W}_h$ and $\mathbf{W}_x$ at each gate, the result should be split into four which are fed into the

sigmoid functions and hyperbolic tangent function (represented as four activation functions AF1, AF2, AF3, AF4, respectively) to perform forget gate computing, output computing, input gate computing, and output gate computing. This split causes four discontinuous memory accesses and is reflected in an intermediate representation as a set of slice operations (e.g., as shown in **FIG. 7**). When a neural network accelerator system performs the slice operation, pipelined computations for neural network processing will be discontinued. In addition, the slice operations will also introduce additional memory operations such as data transfer, data load, data store, etc. It is important to reduce the impact of the slice operation on performance of a neural network accelerator system without changing the functionality.

[043] Embodiments of the present disclosure adapt an intermediate representation by slice operation hoisting to address the issue caused by the slice operation. To show how a slice operation can be hoisted consistent with embodiments of the present disclosure, reference is now made to **FIG. 5** and **FIG. 6**. **FIG. 5** illustrates a first example of a computation graph. **FIG. 5** shows a computation graph including a multiplication operation MUL, two input matrices **H** and **W** to the multiplication operation MUL, and a first set of slice operations Slice 1 and Slice 2 to an output **R** of the multiplication operation MUL. The multiplication operation MUL receives the two input matrices **H** and **W** and outputs **R** to two slice operations Slice 1 and Slice 2. The first slice operation Slice 1 receives the output **R** as its input and splits the output **R** to produce its output **R1**. The second slice operation Slice 2 receives the output **R** as its input and splits the output **R** to produce its output **R2**.

[044] **FIG. 6** illustrates an adapted computation graph of the computation graph of **FIG. 5** after slice operation hoisting, consistent with embodiments of the present disclosure. As shown in **FIG. 6**, a second set of slice operations Slices 1 to 4 are placed in front of multiplication

operations MUL1 and MUL2 in the adapted computation graph while in **FIG. 5** the first set of slice operations Slices 1 and 2 are placed right after the multiplication operation MUL in the original computation graph. Here, it is noted that the multiplication operations MUL1 and MUL2 in **FIG. 6** correspond to the multiplication operation MUL in **FIG. 5**.

[045] In **FIG. 6**, each of the second set of slice operations Slices 1 to 4 receives one of input matrices **H** and **W** as its input and splits the one of the input matrices **H** and **W** to produce its output sliced matrices **H1**, **H2**, **W1**, or **W2**. For example, the first slice operation Slice 1 among the second set of slice operations receives the input matrix **H** and splits the input matrix **H** to produce sliced matrix **H1** as its output. The third slice operation Slice 3 among the second set of slice operations receives the input matrix **W** and splits matrix **W** to produce sliced matrix **W1** as its output. The first multiplication operation MUL1 receives the sliced matrices **H1** and **W1** from the first slice operation S1 and third slice operation Slice 3, and outputs a result **R1** of multiplication of the sliced matrices **H1** and **W1**. Similarly, the second multiplication operation MUL2 receives the sliced matrices **H2** and **W2** of the second slice operation Slice 2 and fourth slice operation Slice 4 and outputs a result **R2** of multiplication of the sliced matrices **H2** and **W2**.

[046] The adapted computation graph of **FIG. 6** includes the second set of slice operations placed in front of the multiplication operations MUL1 and MUL2 in a sequence of operations while the multiplication operation MUL of **FIG. 5** is placed in front of the first set of slice operations in the computation graph**.** According to embodiments of the present disclosure, the outputs **R1** and **R2** of the computation graph in **FIG. 5** are functionally equal to the outputs **R1** and **R2** of the adapted computation graph in **FIG. 6**. According to embodiments of the present disclosure, a slice operation can be hoisted in a computation graph without changing the

functionality of the original computation graph. In order to maintain the functionality of the computation graph before and after the slice operation hoist, input operands should be properly segmented by a second set of slice operations. Particularly, when the input operands **H** and **W** are matrices, it is important to properly segment the input operand matrices to maintain the functionality of an intermediate representation before and after the slice operation hoist.

[047] Segmenting the input matrices in an adapted intermediate representation to maintain the functionality of the original intermediate representation can include several steps. One exemplary equation for a matrix multiplication operation can be represented as below:

$$R = H \times W \qquad \text{(Equation 1)}$$

Here, a multiplication operation receives matrices **H** and **W** as its input operands and outputs **R**, which corresponds to an example shown in **FIG. 5**. Here, when the dimensions of matrices **H** and **W** are [M,p] and [p,N] respectively, the dimensions of the result matrix **R** will be [M, N], which means the result matrix **R** has M number of rows and N number of columns. Here, M, N, and p are natural numbers.

[048] When the result matrix **R** is sliced into sliced result matrix **R1** ([s:s+m-1, t:t+n-1), where s+m-1<M, t+n-1<N, the sliced result matrix **R1** can be expressed as below:

$$\mathrm{R1}[\mathrm{s}\colon \mathrm{s}+\mathrm{m}-1, \mathrm{t}\colon \mathrm{t}+\mathrm{n}-1] = \begin{bmatrix} r_{st} & \cdots & r_{s(t+n-1)} \\ \vdots & \ddots & \vdots \\ r_{(s+m-1)t} & \cdots & r_{(s+m-1)(t+n-1)} \end{bmatrix} \qquad \text{(Equation 2)}$$

Here, the sliced result matrix **R1** has a portion of rows of original result matrix **R** from a row number "s" to a row number "s+m-1." Similarly, the sliced result matrix **R1** has a portion of columns of the original result matrix **R** from a column number "t" to a column number "t+n-1." Each element of the sliced result matrix **R1** can be represented as r_{ij}, where $s \leq i \leq s+m-1$, $t \leq j \leq$

t+n-1, and s, m, t, and n are non-negative integers. Each element r_{ij} of the sliced result matrix **R1** can be expressed as a multiplication of an element from input matrix **H** and an element from weight matrix **W** as below:

$$r_{ij} = \sum_{k=0}^{p-1} h_{ik} * w_{kj} \qquad \text{(Equation 3)}$$

Here, an element r_{ij} of the sliced result matrix **R1** is a multiplication of an element h_{ik} of the input matrix **H** and an element w_{kj} of the weight matrix **W**. The element r_{ij} is an element located at an intersection of an i'th row and j'th column of the original result matrix **R**. The element h_{ik} is an element located at an intersection of an i'th row and k'th column of the input matrix **H**, and the element w_{kj} is an element located at an intersection of a k'th row and j'th column of the weight matrix **W**.

[049] As known from Equations 2 and 3, the sliced result matrix **R1** can be obtained by matrix multiplication of a portion of input matrix **H** (**H1** [s:s+m-1,0:p-1]) and a portion of weight matrix **W** (**W1** [0:p-1,t:t+n-1]). Here, the matrix **H1** has a portion of rows of original input matrix **H** from a row number "s" to a row number "s+m-1." Similarly, the matrix **H1** has a portion of columns of the original input matrix **H** from a column number "0" to a column number "p-1." The matrix **W1** has a portion of rows of original weight matrix **W** from a row number "0" to a row number "p-1." Similarly, the matrix **W1** has a portion of columns of the original weight matrix **W** from a column number "t" to a column number "t+n-1."

[050] Now the sliced result matrix **R1** can be expressed with multiplication of a sliced input matrix **H1** and a sliced weight matrix **W1** as below:

$$H1[s: s+m-1, 0: p-1] \times W1[0: p-1, t: t+n-1]$$

$$= \begin{bmatrix} \sum_{k=0}^{p-1} h_{sk} * w_{kt} & \cdots & \sum_{k=0}^{p-1} h_{sk} * w_{k(t+n-1)} \\ \vdots & \ddots & \vdots \\ \sum_{k=0}^{p-1} h_{(s+m-1)k} * w_{kt} & \cdots & \sum_{k=0}^{p-1} h_{(s+m-1)k} * w_{k(t+n-1)} \end{bmatrix}$$

$$= \begin{bmatrix} r_{st} & \cdots & r_{s(t+n-1)} \\ \vdots & \ddots & \vdots \\ r_{(s+m-1)t} & \cdots & r_{(s+m-1)(t+n-1)} \end{bmatrix}$$

$$= R1[\mathrm{s}: \mathrm{s}+\mathrm{m}-1, \mathrm{t}: \mathrm{t}+\mathrm{n}-1] \qquad \text{(Equation 4)}$$

Equation 4 can be simplified as Equation 5:

$$H1[s: s+m-1, 0: p-1] \times W1[0: p-1, t: t+n-1] = R1[\mathrm{s}: \mathrm{s}+\mathrm{m}-1, \mathrm{t}: \mathrm{t}+\mathrm{n}-1]$$

(Equation 5)

From Equation 5, it is noted that a first dimension (a row number) of the sliced input matrix **H1** becomes a first dimension of the sliced result matrix **R1** and a second dimension (a column number) of the sliced weight matrix **W1** becomes a second dimension of the sliced result matrix **R1**. These equations can also be used to determine the result of sliced result matrix **R2**.

[051] In order to make the adapted computation graph in **FIG. 6** maintain the functionality of the original computation graph in **FIG. 5**, it is important that the second set of slice operations Slices 1 to 4 in **FIG. 6** properly segment the input and weight matrices **H** and **W**. In other words, the second set of slice operations Slices 1 to 4 in **FIG. 6** segment the input and weight matrices **H** and **W** such that the output matrices **R1** and **R2** of the multiplication operations MUL1 and MUL2 in **FIG. 6** are equal to the sliced result matrices **R1** and **R2** in **FIG. 5**. For example, the first slice operation Slice 1 among the second set of slice operations segments the input matrix **H** such that the sliced input matrix **H1** in **FIG. 6** has at least one row of the input matrix **H** corresponding to at least one row number that the sliced result matrix **R1** occupies in the original result matrix **R** in **FIG. 5** as expressed in Equation 5. Also, the third slice operation Slice 3 among the second set of slice operations segments the weight matrix **W** such

that the sliced weight matrix **W1** in **FIG. 6** has at least one column of the weight matrix **W** corresponding to at least one column number that the sliced result matrix **R1** occupies in the original result matrix **R** in **FIG. 5** as expressed in Equation 5. The second and fourth slice operations Slices 2 and 4 may segment the input and weight matrices **H** and **W** such that the sliced input and weight matrices **H2** and **W2** in **FIG. 6** can have a similar relationship with the sliced result matrix **R2** in **FIG. 5** as expressed in Equation 5. As illustrated above, when there is a first set of slice operations after a multiplication operation in the computation graph, embodiments of the present disclosure can transform a computation graph by positioning a second set of slice operations in front of a corresponding set of multiplication operations in the transformed computation graph without changing the functionality of the original computation graph.

[052] It will be understood that the slice operation hoisting explained regarding the matrix multiplication operation can also be performed in relation to an element-wise matrix operation or broadcast operation. An element-wise matrix operation may refer to a matrix operation performed on an element-by-element basis between matrices having the same dimensions. The matrix multiplication operation illustrated referring to **FIG. 5** and **FIG. 6** is different from an element-wise matrix operation in that, if **H** is a [M, p] matrix and **W** is a [p, N] matrix, their matrix multiplication **H**×**W** is a [M, N] matrix, in which the p entries across a row of **H** are multiplied with p entries down a column of **W** and summed to produce an entry of **H**×**W**. The element-wise matrix operation may include addition, subtraction, multiplication, division, exponentiation, etc. Here, the element-wise matrix multiplication, generally called as the Hadamard product, is a binary operation that takes two matrices (e.g., input matrix **H** [M, N] and weight matrix **W** [M, N]) of the same dimensions, and produces another matrix (e.g., result

matrix **R** [M, N]) where each element r_{ij} is the product of elements h_{ij} and w_{ij} of the two input and weight matrices **H** and **W**. In an element-wise matrix operation, each of the second set of slice operations can segment the input matrix such that the segmented input matrix has the same dimensions with the sliced output matrix. A broadcast operation is to make arrays with different shapes have compatible shapes for arithmetic operations. Similar to an element-wise matrix operation, slice operation hoisting to a broadcast operation can be performed in a back-direction manner, consistent with embodiments of the present disclosure.

[053] Another example of adapting a computation graph will be explained with reference to **FIG. 7A, FIG. 7B** and **FIG. 8**. **FIG. 7A** illustrates a second example of a computation graph. In **FIG. 7A,** a multiplication operation MUL receives an input matrix **I** and weight matrix **W** as its input operands and outputs its multiplication result matrix **M** to an addition operation ADD. A broadcast operation receives a bias vector **B** and makes the bias vector **B** (which can also be considered a bias matrix having one row or one column) have compatible dimensions with the multiplication result matrix **M**, thereby transforming bias vector **B** into a bias matrix **B**. In other words, the elements in the bias vector **B** having dimension [1, 1152] are broadcasted to make the bias vector **B** have the same dimensions as the multiplication result matrix **M** of [288, 1152]. Here, the broadcast operation can be performed by replicating the bias vector **B** [1, 1152] 287 times to make a broadcast bias matrix **B** [288, 1152].

[054] After making the data compatible, the broadcast operation outputs a bias matrix **B** for computing the addition operation ADD between the multiplication result matrix **M** and the bias matrix **B**. An addition operation ADD receives the output result matrices **M** and **B** of the multiplication operation MUL and broadcast operation, and outputs its result matrix **A** to each of the first set of slice operations Slice 1 to Slice 4. Each of the first set of slice operations Slice 1 to

Slice 4 segments the result matrix **A** of the addition operation ADD. Sliced result matrices **A1** to **A4** of the first set of slice operations are fed to corresponding activation function operations sigmoid 1, sigmoid 2, tanh, and sigmoid 3, respectively. Results matrices **AF1** to **AF4** of the activation function operations are collected as a sequence through a tuple operation. The result of the tuple operation is fed to a root node.

[055] In **FIG. 7A**, the first set of slice operations Slice 1 to 4 are located in the middle in a sequence of operations in the computation graph. These slice operations may break pipelined computing and cause discontinuous memory access. According to embodiments of the present disclosure, it is possible to expand a degree of freedom for optimizing the computation graph by hoisting the slice operations to a higher order in a sequence of operations in a computation graph. Moreover, by moving the slice operations to a higher order before the computing can assist with reducing memory overhead and with increasing the instruction fuse optimization opportunities, thereby improving the overall system performance for a matrix-based hardware accelerator.

[056] **FIG. 8** illustrates an adapted computation graph of the computation graph of **FIG. 7A** after slice operation hoisting, consistent with embodiments of the present disclosure. The slice operation hoisting can be performed as explained referring to **FIG. 5** and **FIG. 6**. In **FIG. 8**, a second set of slice operations Slice 1 to Slice 8 are located right after the weight and bias input parameters, unlike the scenario in **FIG. 7A** where the first set of slice operations Slice 1 to Slice 4 are located after the addition operation ADD. In the example of **FIG. 8**, the input paraments include the input matrix **I**, weight matrix **W**, and bias vector **B**, with weigh matrix **W** and bias vector **B** being segmented by slice operations. The adapted computation graph of **FIG. 8** can be obtained by transforming the computation graph of **FIG. 7A**. For example, the adapted

computation graph of **FIG. 8** can be obtained by twice performing slice operation hoisting on the computation graph of **FIG. 7A**.

[057] **FIG. 7B** illustrates an interim state of the computation graph of **FIG. 7A** after first slice operation hoisting. Using **FIG. 7B** as a reference, in a first slice operation hoisting phase, an interim set of slice operations can be positioned in front of the addition operation ADD. In the first slice operation hoisting phase, an interim set of slicing operations (e.g., interim slicing operations Slice 1_i to Slice 8_i) are placed in front of four addition operations ADD1 to ADD4. Each of the four addition operations ADD1 to ADD4 can provide a result having dimensions of [288, 288], thereby matching the resulting dimensions [288, 1152] of the single ADD operation of **FIG. 7A**.

[058] In the first slice operation hoisting phase, interim slicing operations Slices 1_i -4_i would receive the resulting matrix **M** of multiplication operation MUL as their input. Then, Slice 1_i would segment the resulting matrix **M** of [288, 1152] to obtain a sliced resulting matrix **M1** including row numbers 0 to 287 and column numbers 0 to 287 of the resulting matrix **M** of the multiplication operation MUL. Slice 2_i would segment the resulting matrix **M** of [288, 1152] to obtain a sliced resulting matrix **M2** including row numbers 0 to 287 and column numbers 288 to 575 of the resulting matrix **M**. Slice 3_i would segment the resulting matrix **M** of [288, 1152] to obtain a sliced resulting matrix **M3** including row numbers 0 to 287 and column numbers 576 to 863 of the resulting matrix **M**. Slice 4_i would segment the resulting matrix **M** of [288, 1152] to obtain a sliced resulting matrix **M4** including row numbers 0 to 287 and column numbers 864 to 1151 of the resulting matrix **M**. Moreover, interim slicing operations Slices 5_i -8_i would receive the resulting matrix **B** of broadcast operation Broadcast. Similarly, slice 5_i would segment the resulting matrix **B** of [288, 1152] to obtain a sliced resulting matrix **B1** including row numbers 0

to 287 and column numbers 0 to 287 of the resulting matrix **B** of the broadcast operation Broadcast. Slice 6_i would segment the resulting matrix **B** of [288, 1152] to obtain a sliced resulting matrix **B2** including row numbers 0 to 287 and column numbers 288 to 575 of the resulting matrix **B**. Slice 7_i would segment the resulting matrix **B** of [288, 1152] to obtain a sliced resulting matrix **B3** including row numbers 0 to 287 and column numbers 576 to 863 of the resulting matrix **B**. Slice 8_i would segment the resulting matrix **B** of [288, 1152] to obtain a sliced resulting matrix **B4** including row numbers 0 to 287 and column numbers 864 to 1151 of the resulting matrix **B**.

[059] The addition operations ADD1 to ADD4 receive sliced resulting matrices **M1** to **M4** and **B1** to **B4** from corresponding interim slice operations Slices 1_i to 8_i. For example, the first addition operation ADD1 receives the sliced resulting matrices **M1** and **B1** from the first and fifth interim slice operations Slices 1_i and 5_i and outputs the result matrix **A1**. Here, the result matrix **A1** of the first addition operation ADD1 in **FIG. 7B** is equal to the sliced result matrix **A1** of the first slice operation Slice 1 in **FIG. 7A**. Since an addition operation is an element-wise matrix operation, the interim slice operation Slice 1_i segments the resulting matrix **M** of the multiplication operation MUL such that the sliced resulting matrix **M1** has rows and columns of the resulting matrix **M** corresponding to the row and column numbers that the sliced resulting matrix **A1** occupies in the result matrix **A** in **FIG. 7A**, here [0:287, 0:287]. Also, the interim slice operation Slice 5_i segments the resulting matrix **B** of the broadcast operation Broadcast such that the sliced resulting matrix **B1** has rows and columns of the resulting matrix **B** corresponding to the same row and column numbers that the sliced resulting matrix **A1** occupies in the result matrix **A** in **FIG. 7A**, here [0:287, 0:287]. Similarly, the interim slice operations 2_i to 4_i and 6_i to 8_i can segment the input matrices **M** or **B** so that the result matrices

A2 to **A4** of addition operations ADD2 to ADD4 in **FIG. 7B** are equal to the sliced result matrices **A2** to **A4** of slice operations Slices 2 to 4 in **FIG. 7A**.

[060] After a first slice hoisting phase is performed, a second slice hoisting phase can occur where the interim-slicing operations Slices 1_i -8_i are moved before the multiplication operation MUL and the broadcast operation Broadcast of **FIG. 7B**. To do this hoisting, however, the multiplication and broadcast operations are segmented in a manner similar to the adding operation during the first slice hoisting phase. That is, both the multiplication and broadcast operations are separated into four operations (each of which can generate a resulting matrix having dimensions of [288, 288] as shown in **FIG. 8**), thereby matching the resulting dimensions [288, 1152] of both the MUL and Broadcast operations of **FIG. 7B**.

[061] In this particular example, the previous interim slice operations Slices 1_i -4_i can be repositioned as part of a second set of slicing operations Slices 1-4 in front of multiplication operations MUL1-MUL4, respectively 4, while the previous interim slice operations Slices 5_i -8_i can be repositioned as part of the second set of slicing operations Slices 5-8 in front of broadcast operations Broadcast1-Broadcast4 as shown in **FIG. 8**. Depending on the input, the slice operation dimensions may change. For example, because the dimensions of weight matrix is larger, the slicing operations Slices 1-4 have larger dimensions of [576, 288] than its previous interim slicing operations Slice 1_i-4_i of the first hoisting phase. Moreover, because the dimensions of bias vector is smaller, the slicing operations Slices 5-8 have smaller dimensions of [1, 288] than its previous interim slicing operations Slice 5_i -8_i of the first hoisting phase. Here, since the row dimension ([0:287]) of the resulting matrix **M** of the multiplication operation MUL in **FIG. 7B** is not segmented by the interim slice operations Slices 1_i to 4_i, the input matrix **I** is not sliced in **FIG. 8**. That is, all the sliced result matrices **M1** to **M4** of the interim slice

operations Slices 1_i to 4_i have the same row numbers ([0:287]) with the resulting matrix **M** of the multiplication operation MUL in **FIG. 7B**. Thus, **FIG. 8** does not include additional slice operations for the input matrix **I**. The slice operation hoisting can be repeated until a set of slice operations are placed right after the input parameters.

[062] In this way, slice operations can be positioned higher in a sequence of operations in the adapted computation graph. Thereby, it is possible to expand a degree of freedom for optimizing the rest of the computation graph without experiencing discontinuous memory access or pipelined computation discontinuity due to the slice operations. The adapted computation graph after the slice operation hoisting can allow more opportunities to optimize the computation graph through instruction fusing, node clustering, etc. A typical machine learning or deep learning model may have thousands or even millions of nodes and hundreds of Mbytes of data. It means that a computation graph representing the typical machine learning or deep learning model may be thousands or millions of times larger than the computation graph illustrated in **FIG. 5** or **FIG. 7A**. To accelerate the execution of the machine learning or deep learning model, enormous amount of resources such as processing units and storage spaces are necessary. Otherwise, the execution of the machine learning or deep learning model will take too much time. Since the resources of an accelerator is limited, optimization on the computation graph is important to improve performance of the accelerator. In some embodiments, the optimization can be performed by an instruction adjuster 323 of **FIG. 3**.

[063] According to embodiments of the present disclosure, the functionality of the computation graph of **FIG. 7A** is maintained after the slice operation hoisting. Therefore, the adapted computation graph of **FIG. 8** can perform the one or more tasks of the original computation graph of **FIG. 7A**.

[064] In some embodiments, a mapping data associating a raw data (e.g., input parameters) with a corresponding slice operation can be generated. For example, referring to **FIG. 8**, a raw weight matrix **W** is fed to the four slice operations Slices 1 to 4, and a raw bias vector **B** is fed to the other four slice operations Slices 5 to 8. The mapping data can include the relationship between the raw data and the corresponding slice operations. In some embodiments, the mapping data can be generated by an apparatus 300 in **FIG. 3**. The apparatus 300 may store the mapping data in an associated memory device. When a host CPU 210 sends commands to a target accelerator 220 to execute the adapted intermediate representation, the mapping data can be sent together. Then, the raw data can be copied to proper memory places associated with the target accelerator 220 so that the raw data can be fed to proper processing units when computing the corresponding slice operations.

[065] **FIG. 9** illustrates an exemplary method 900 for adapting an intermediate representation of a source code, consistent with embodiments of the present disclosure. In some embodiments of the present disclosure, the method 900 may be performed by an apparatus 300 in **FIG. 3**. After initial start, at step S910, a first set of slice operations are detected in the intermediate representation. Next, at step S920, whether at least one first operation (e.g., ADD in **FIG. 7A**) exists between the first set of slice operations (e.g., Slices 1 to 4 in **FIG. 7**) and the input parameters (e.g., Input, Weight, Bias in **FIG. 7**) in the sequence of operations in the intermediate representation is determined. If it is determined that there is no additional operation between the detected first set of slice operations and the input parameters at step S920, the method ends. In this case, the first set of slice operations are already placed right after the input parameters, the method 900 does not perform slice operation hoisting. If it is determined that

there is at least one operation between the detected first set of slice operations and the input parameters at step S920, the method proceeds to step S930.

[066] At step S930, slice operation hoisting is performed. By this slice operation hoisting, the second set of slice operations are positioned in front of a second operation in the sequence of operations in the transformed intermediate representation. Here, a set of second operations (e.g., ADD1 to ADD4 in **FIG. 7B**) in the transformed intermediate representation correspond to a first operation (e.g., ADD in **FIG. 7A**) among the at least one first operation placed in front of the first set of slice operations in the sequence of operations in the intermediate representation before the transformation.

[067] According to embodiments of the present disclosure, the steps S920 and S930 can be repeated until the input parameters are positioned in front of the slice operations. In other words, the steps S920 and S930 are repeated until no operation exists between the slice operations and the input parameters. After the first slice operation hoisting is performed, at step S920, whether at least one operation exists between the second set of slice operations and input parameters to the intermediate representation in a sequence of operations in the transformed intermediate representation by the first slice operation hoisting is determined. In the second slice operation hoisting phase, the second set of slice operations are treated as a first set of slice operations in the first slice operation hoisting phase. If it is determined that there is no additional operation between the second set of slice operations and the input parameters at step S920, the method ends. If it is determined that there is at least one operation between the second set of slice operations and the input parameters at step S920, the method proceeds to step S930. At step S930, second slice operation hoisting is performed. Similarly, a third slice operation hoisting and more can be performed, consistent with embodiments of the present disclosure.

[068] In some embodiments, at step S930, the intermediate representation is transformed such that the transformed intermediate representation includes a second set of slice operations positioned higher in a sequence of operations in the transformed intermediate representation than the first set of slice operations in a sequence of operations in the intermediate representation before the transformation. Here, functionality of the intermediate representation is maintained after the transformation.

[069] In some embodiments, at step S930, each slice operation of the second set of slice operations (e.g., Slices 1 to 4 in **FIG. 6**) is instructed to receive the same input operand (e.g., input matrix **H** or weight matrix **W** in **FIG. 5**) to the first operation (e.g., MUL in **FIG. 5**) before the transformation, and to segment the input operand based on an output result (e.g., sliced result matrix **R1** or **R2** in **FIG. 5**) of a corresponding slice operation (e.g., Slices 1 to 2 in **FIG. 5**) among the first set of slice operations before the transformation. In some embodiments, at step S930, each slice operation of the second set of slice operations is instructed to receive an input matrix to the first operation before the transformation, and to segment the input matrix based on an output matrix of a corresponding slice operation among the first set of slice operations before the transformation.

[070] In some embodiments, at step S920, when the first operation is a multiplication operation (e.g., MUL in **FIG. 5**), and the input operand to the first operation before the transformation comprises a first matrix (e.g., input matrix **H** in **FIG. 5**) and a second matrix (e.g., weight matrix **W** in **FIG. 5**), one slice operation (e.g., Slice 1 in **FIG. 6**) of the second set of slice operations is instructed to receive the first matrix (e.g., input matrix **H** in **FIG. 5**), and to segment the first matrix based on an output matrix (e.g., sliced result matrix **R1** in **FIG. 5**) of a corresponding slice operation (e.g., Slice 1 in **FIG. 5**) among the first set of slice operations

before the transformation. Here the segmentation is performed such that the segmented first matrix (e.g., sliced input matrix **H1** in **FIG. 6**) includes rows (e.g., s:s+m-1 in Equation 5) of the first matrix corresponding to row numbers of a portion (e.g., corresponding to sliced result matrix **R1**) in an output matrix (e.g., result matrix **R** in **FIG. 5**) of the first operation (e.g., MUL in **FIG. 5**) before the transformation. Here, the portion in the output matrix of the first operation corresponds to the output matrix (e.g., sliced result matrix **R1** in **FIG. 5**) of the corresponding slice operation (e.g., Slice 1 in **FIG. 5**) before the transformation.

[071] In some embodiments, at step S920, when the first operation is a multiplication operation (e.g., MUL in **FIG. 5**), and the input operand to the first operation before the transformation comprises a first matrix (e.g., input matrix **H** in **FIG. 5**) and a second matrix (e.g., weight matrix **W** in **FIG. 5**), one slice operation (e.g., Slice 3 in **FIG. 6**) of the second set of slice operations is instructed to receive the second matrix, and to segment the second matrix based on an output matrix (e.g., sliced result matrix **R1** in **FIG. 5**) of a corresponding slice operation (e.g., Slice 1 in **FIG. 5**) among the first set of slice operations before the transformation. Here, the segmentation is performed such that the segmented second matrix (e.g., sliced weight matrix **W1** in **FIG. 6**) includes columns (e.g., t:t+n-1 in Equation 5) of the second matrix corresponding to column numbers of a portion (e.g., corresponding to sliced result matrix **R1**) in an output matrix (e.g., result matrix **R** in **FIG. 5**) of the first operation before the transformation. Here, the portion in the output matrix (e.g., sliced result matrix **R1** in **FIG. 5**) of the first operation corresponds to the output matrix of the corresponding slice operation (e.g., Slice 1 in **FIG. 5**) before the transformation.

[072] In some embodiments of the present disclosure, the intermediate representation is associated with a computation graph, and the transformation is applied to the computation graph.

[073] **FIG. 10** illustrates an exemplary flow diagram for executing a source code including a method for adapting an intermediate representation of the source code in a neural network accelerator system, consistent with embodiments of the present disclosure. Here, it is illustrated that the source code is executed in a neural network accelerator system 200 in **FIG. 2** as an example. In a host CPU 210, an intermediate representation for the source code can be generated at step S1010. In some embodiments, the intermediate representation can be generated from another high-level code initially compiled from the source code.

[074] At step S1020, optimization on the intermediate representation is performed. As explained referring to **FIG. 3**, the optimization can include transforming the intermediate representation by performing slice operation hoisting. Further, the optimization may include an instruction adjustment. Here, the optimization can be performed by considering the runtime environment of the target machine. Then, based on the optimized intermediate representation and instructions, a target machine language code is generated at step S1030. In addition to the target machine language code, a mapping data associating a raw data with a corresponding slice operation can be generated. The mapping data can be stored in a memory device associated with the host CPU 210.

[075] At step S1040, the generated code and map are sent to a target machine for running the target machine language code on the target machine. Here, runtime parameters including the slice information can also be sent to the target machine. In some embodiments, the target machine can be a neural network accelerator, such as an accelerator 220 in **FIG. 2**.

[076] At step S1050, the accelerator 220 copies the raw data to proper memory places associated with the accelerator 220 based on the mapping data so that the raw data can be fed to proper processing units when computing the corresponding slice operations. In some

embodiments, the raw data can be copied to multiple memory places since multiple slice operations may need to segment the same raw data after slice operation hoisting. At step S1060, the target machine language code is executed on the accelerator 220. After a result is generated from the execution of the code, the result is sent to the host CPU 210 at step S1070. Here, in some embodiments, the accelerator 220 may store the result in its associated memory device for further use. The host CPU 220 receives the result at step S1080 and stores the result in its associated memory device.

[077] Based on the foregoing, slice operation hoisting is performed to an intermediate representation of a source code until the slice operation is positioned right after input parameters to the intermediate representation, consistent with embodiments of the present disclosure. According to embodiments of the present disclosure, by positioning slice operations at an input end of an intermediate representation, memory operation overhead due to discontinuous memory access behaviour can be reduced. According to embodiments of the present disclosure, it is possible to increase a scope for neural network instruction fuse and optimization. According to embodiments of the present disclosure, on-chip pipelined computation discontinuity and DRAM memory operation overhead problems can be resolved. According to embodiments of the present disclosure, overall performance of a neural network accelerator system can be improved. According to embodiments of the present disclosure, overall performance of a matrix-based hardware accelerator can be improved.

[078] 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, 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.

[079] 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. An apparatus for adapting an intermediate representation of a source code, comprising:

a slice operation detector configured to detect a first set of slice operations associated with the intermediate representation, wherein the intermediate representation includes a plurality of input parameters and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations; and

an intermediate representation transformer configured to transform the intermediate representation to generate a transformed intermediate representation, wherein the transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

2. The apparatus of claim 1, wherein the first set of one or more matrix operations includes a first and second matrix operations that are positioned in front of the first set of slice operations in the sequence of operations,

wherein the intermediate representation transformer is configured to position a first portion of the second set of slice operations in front of a third set of matrix operations in the sequence of operations of the transformed intermediate representation and a second portion of the second set of slice operations in front of a fourth set of matrix operations, and

wherein the third set of matrix operations in the transformed intermediate representation corresponds to the first matrix operation of the intermediate representation,

wherein the fourth set of matrix operations in the transformed intermediate representation corresponds to the second matrix operation of the intermediate representation.

3. The apparatus of claim 2, wherein:

the first matrix operation is a multiplication operation,

the second matrix operation is a broadcast operation,

the third set of matrix operations are a set of multiplication operations that correspond to the multiplication operation of the intermediate representation, and

the fourth set of matrix operations are a set of broadcast operations that correspond to the broadcast operation of the intermediate representation.

4. The apparatus of claim 3, wherein:

the first portion of the second set of slice operations includes a slicing operation that is configured to acquire a weight input matrix, to segment the weight input matrix, and to provide the segmented weight input matrix to a multiplication operation of the set of multiplication operations, and

the second portion of the second set of slice operations includes a slicing operation that is configured to acquire a bias input matrix, to segment the bias input matrix, and to provide the segmented bias input matrix to a corresponding broadcast operation of the set of broadcast operations.

5. The apparatus of claim 4, wherein the multiplication operations and the broadcast operations output results to addition operations.

6. The apparatus of any one of claims 1-3, wherein each slice operation of the second set of slice operations is instructed to receive a corresponding input matrix and to segment the corresponding input matrix.

7. The apparatus of claim 6, wherein the received corresponding input matrix is from a corresponding input parameter of the plurality of input parameters.

8. The apparatus of claim 1, wherein the first set of one or more matrix operations includes at least one of a matrix multiplication operation, an element-wise matrix operation, or a broadcast operation.

9. The apparatus of claim 1, wherein the intermediate representation transformer is configured to perform the transformation by performing:

determining whether at least one first matrix operation of the first set of one or more matrix operations exists between the first set of slice operations and the plurality of input parameters in the sequence of operations of the intermediate representation; and

positioning the second set of slice operations in front of the second set of matrix operations in the sequence of operations in the transformed intermediate representation, wherein the second set of matrix operations correspond to the at least one first matrix operation.

10. The apparatus of any one of claims 1-9, wherein the intermediate representation is associated with a computation graph, and the transformation is applied to the computation graph.

11. The apparatus of claim 10, wherein the computation graph is a directed acyclic graph.

12. The apparatus of any one of claims 1-11, wherein functionality of the intermediate representation is maintained after the transformation.

13. The apparatus of any one of claims 1-12, wherein at least some of the second set of slice operations are configured to output results having dimensions that are different from dimensions of outputs from the first set of slice operations.

14. The apparatus of any one of claims 1-13, wherein the transformed intermediate representation enables to reduce memory operation overhead due to discontinuous memory access behavior when the transformed intermediate representation is implemented.

15. A method for adapting an intermediate representation of a source code, comprising:

detecting a first set of slice operations associated with the intermediate representation, wherein the intermediate representation includes a plurality of input parameters

and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations; and

transforming the intermediate representation, wherein the transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

16. The method of claim 15, wherein the first set of one or more matrix operations includes a first matrix operation that is positioned in front of the first set of slice operations in the sequence of operations,

wherein the transformation is performed by positioning the second set of slice operations in front of a third set of matrix operations in the sequence of operations in the transformed intermediate representation, and

wherein the third set of matrix operations corresponds to the first matrix operation of the intermediate representation.

17. The method of claim 16, wherein each slice operation of the second set of slice operations is instructed to receive an input matrix corresponding to an input matrix of the first matrix operation, and to segment the received input matrix.

18. The method of claim 16, wherein the first matrix operation is a matrix multiplication operation, and the first matrix operation receives a first matrix and a second matrix as inputs,

wherein a slice operation of the second set of slice operations is instructed to receive the first matrix, to segment the first matrix, and to provide the segmented first matrix to a corresponding matrix multiplication operation of the third set of matrix operations, and

wherein the segmented first matrix includes at least one row of the first matrix corresponding to at least one row number that an output of a corresponding slice operation of the first set of slice operations occupies in an output matrix of the first matrix operation.

19. The method of claim 16, wherein the first matrix operation is a matrix multiplication operation, and the first matrix operation receives a first matrix and a second matrix as inputs,

wherein a slice operation of the second set of slice operations is instructed to receive the second matrix, to segment the second matrix, and to provide the segmented second matrix to a corresponding matrix multiplication operation of the third set of matrix operations, and

wherein the segmented second matrix includes at least one column of the second matrix corresponding to at least one column number that an output of a corresponding slice operation of the first set of slice operations occupies in an output matrix of the first matrix operation.

20. The method of claim 15, wherein transforming the intermediate representation further comprises:

determining whether at least one first matrix operation of the first set of one or more matrix operations exists between the first set of slice operations and the plurality of input parameters in the sequence of operations of the intermediate representation; and

positioning the second set of slice operations in front of the second set of matrix operations in the sequence of operations in the transformed intermediate representation, wherein the second set of matrix operations correspond to the at least one first matrix operation.

21. The method of claim 20, wherein transforming the intermediate representation further comprises:

repeating the determining and positioning until at least some of the plurality of input parameters are positioned immediately in front of the second set of slice operations.

22. The method of any one of claims 15-21, wherein functionality of the intermediate representation is maintained after the transformation.

23. A non-transitory computer readable medium that stores a set of instructions that is executable by at least one processor of a computing device to cause the computing device to perform a method for adapting an intermediate representation of a source code, the method comprising:

detecting a first set of slice operations associated with the intermediate representation, wherein the intermediate representation includes a plurality of input parameters

and a first set of one or more matrix operations positioned earlier than the first set of slice operations in a sequence of operations; and

transforming the intermediate representation, wherein the transformed intermediate representation includes a second set of slice operations positioned in a sequence of operations earlier than a second set of matrix operations that correspond to the first set of one or more matrix operations.

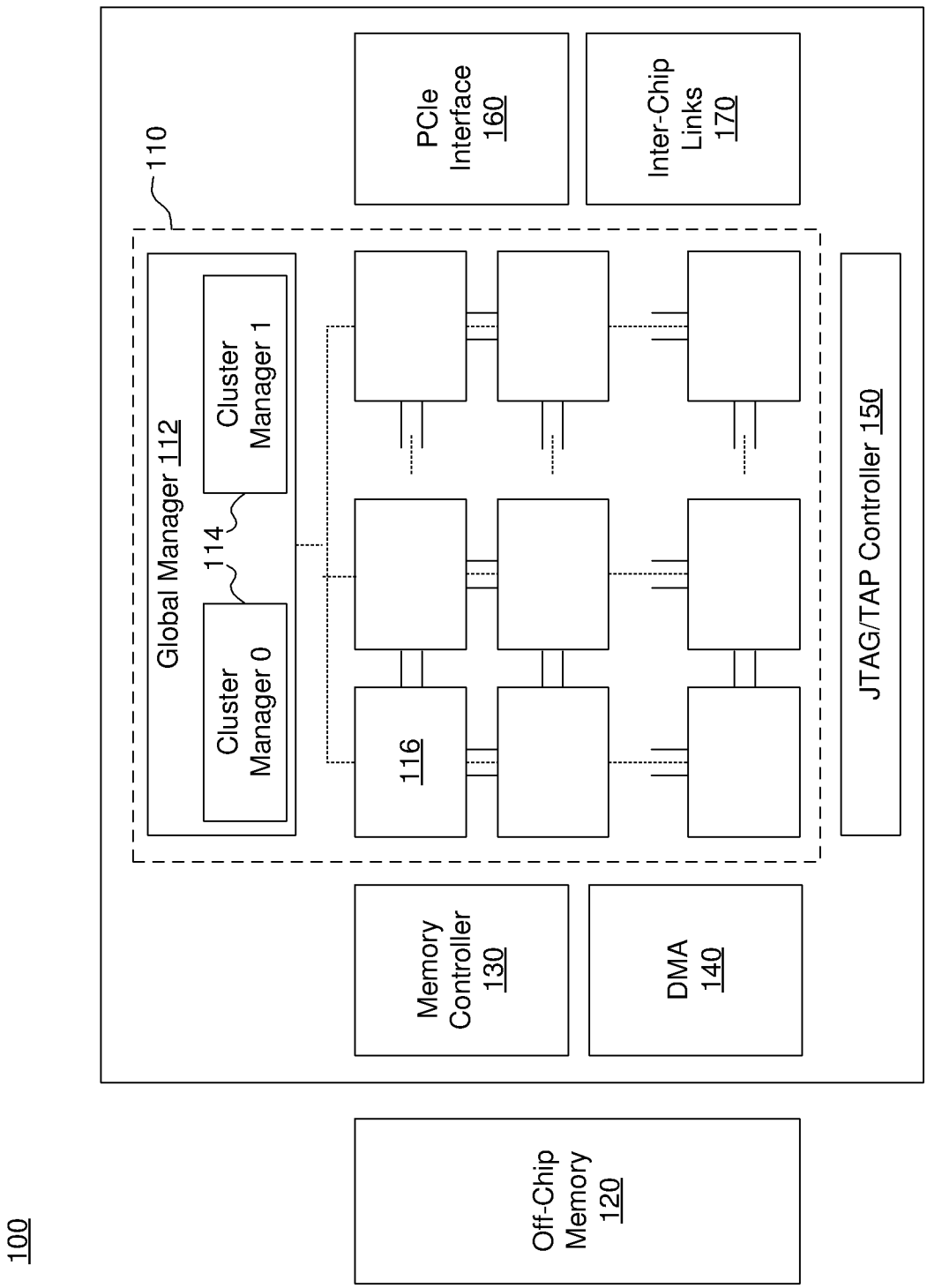
100
Off-Chip Memory 120
Memory Controller 130
DMA 140
PCIe Interface 160
Inter-Chip Links 170
110
Global Manager 112
Cluster Manager 0
114
Cluster Manager 1
116
JTAG/TAP Controller 150

FIG. 1

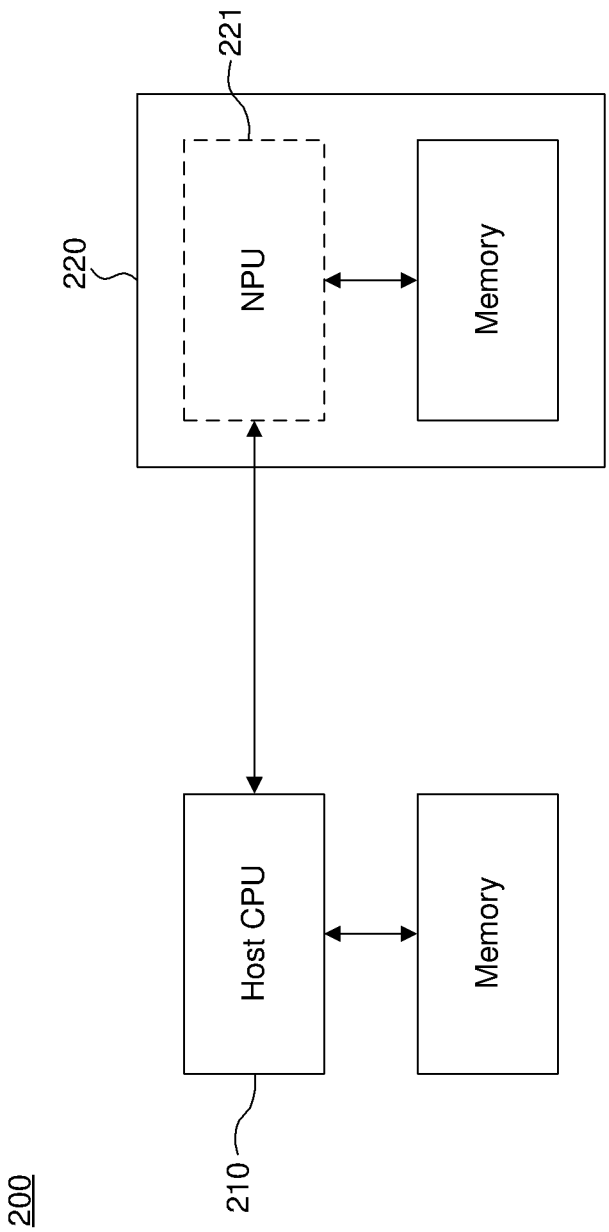

FIG. 2

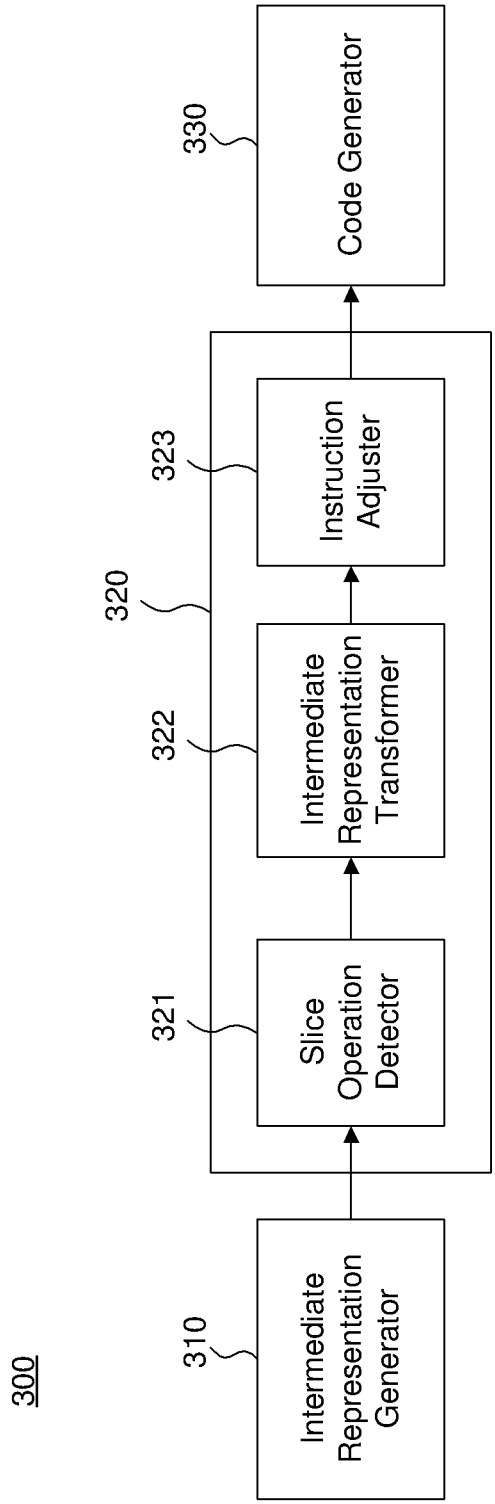
300
310
Intermediate Representation Generator
320
321
Slice Operation Detector
322
Intermediate Representation Transformer
323
Instruction Adjuster
330
Code Generator

FIG. 3

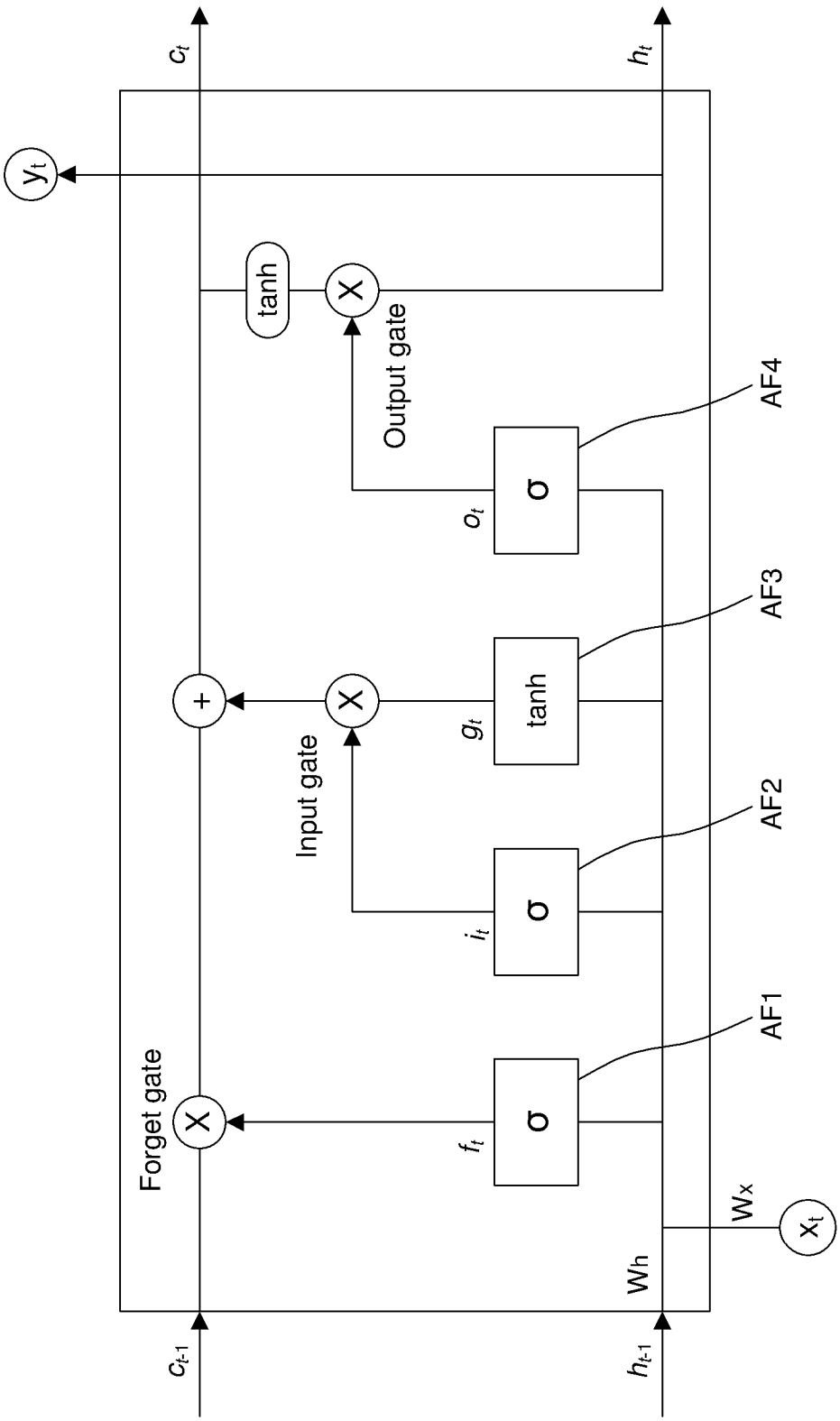
c_t
h_t
y_t
tanh
Output gate
AF4
o_t
σ
AF3
Input gate
g_t
tanh
AF2
i_t
σ
AF1
Forget gate
f_t
σ
Wx
x_t
Wh
c_{t-1}
h_{t-1}

FIG. 4

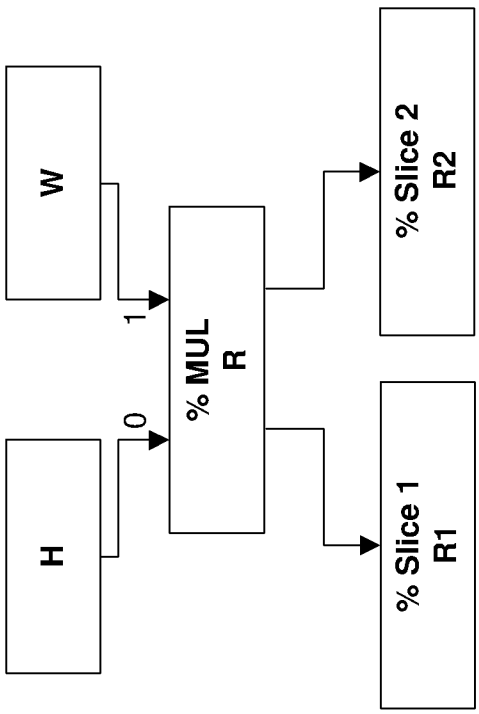

FIG. 5

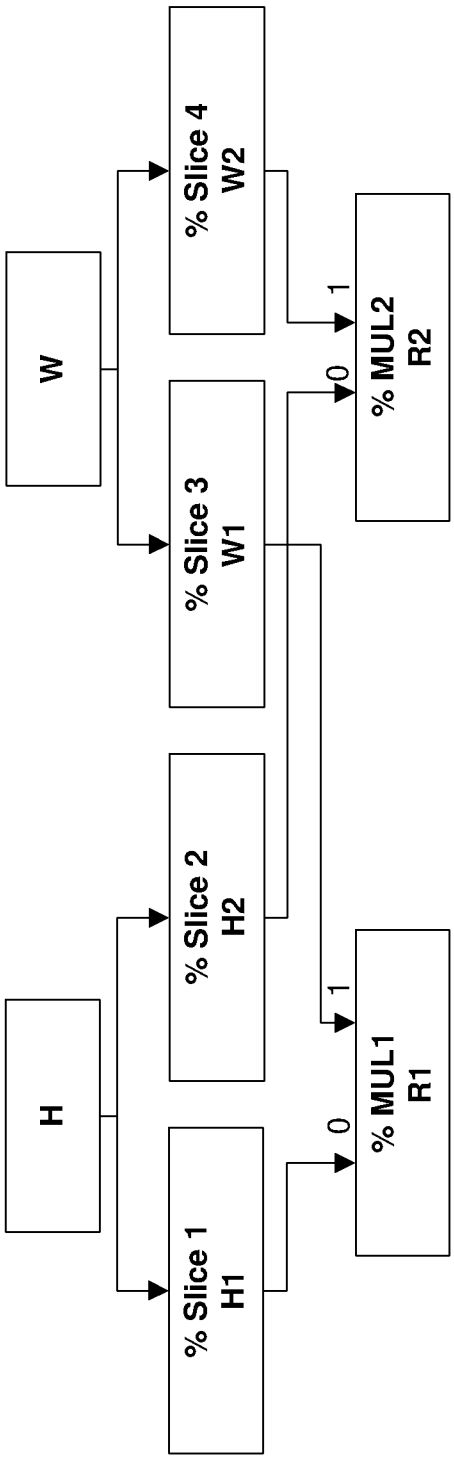
H
W
% Slice 1
H1
% Slice 2
H2
% Slice 3
W1
% Slice 4
W2
0
1
% MUL1
R1
0
1
% MUL2
R2

FIG. 6

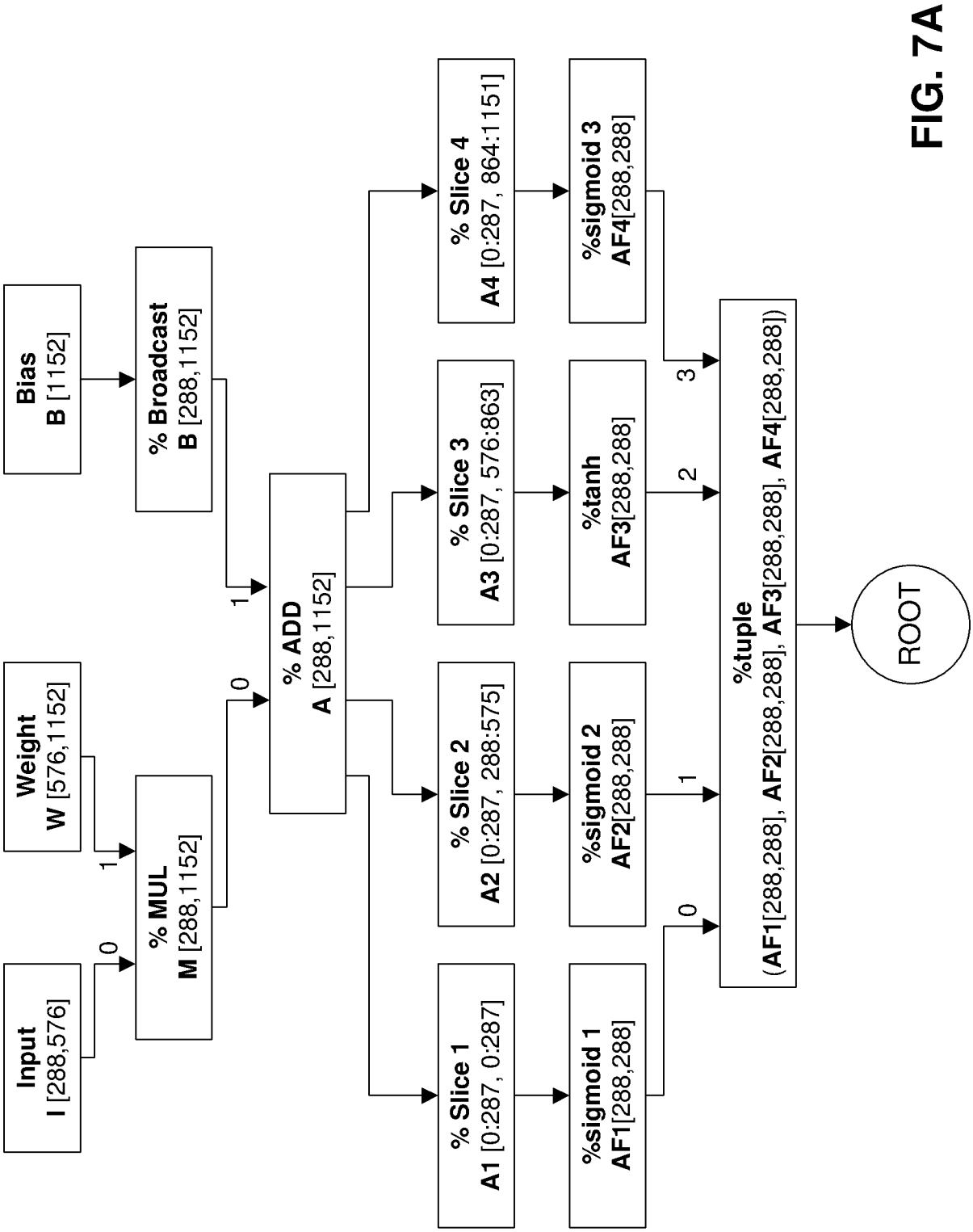
Input
I [288,576]
Weight
W [576,1152]
Bias
B [1152]
% MUL
M [288,1152]
% Broadcast
B [288,1152]
% ADD
A [288,1152]
% Slice 1
A1 [0:287, 0:287]
% Slice 2
A2 [0:287, 288:575]
% Slice 3
A3 [0:287, 576:863]
% Slice 4
A4 [0:287, 864:1151]
%sigmoid 1
AF1[288,288]
%sigmoid 2
AF2[288,288]
%tanh
AF3[288,288]
%sigmoid 3
AF4[288,288]
%tuple
(AF1[288,288], AF2[288,288], AF3[288,288], AF4[288,288])
ROOT
0
1
0
1
0
1
2
3

FIG. 7A

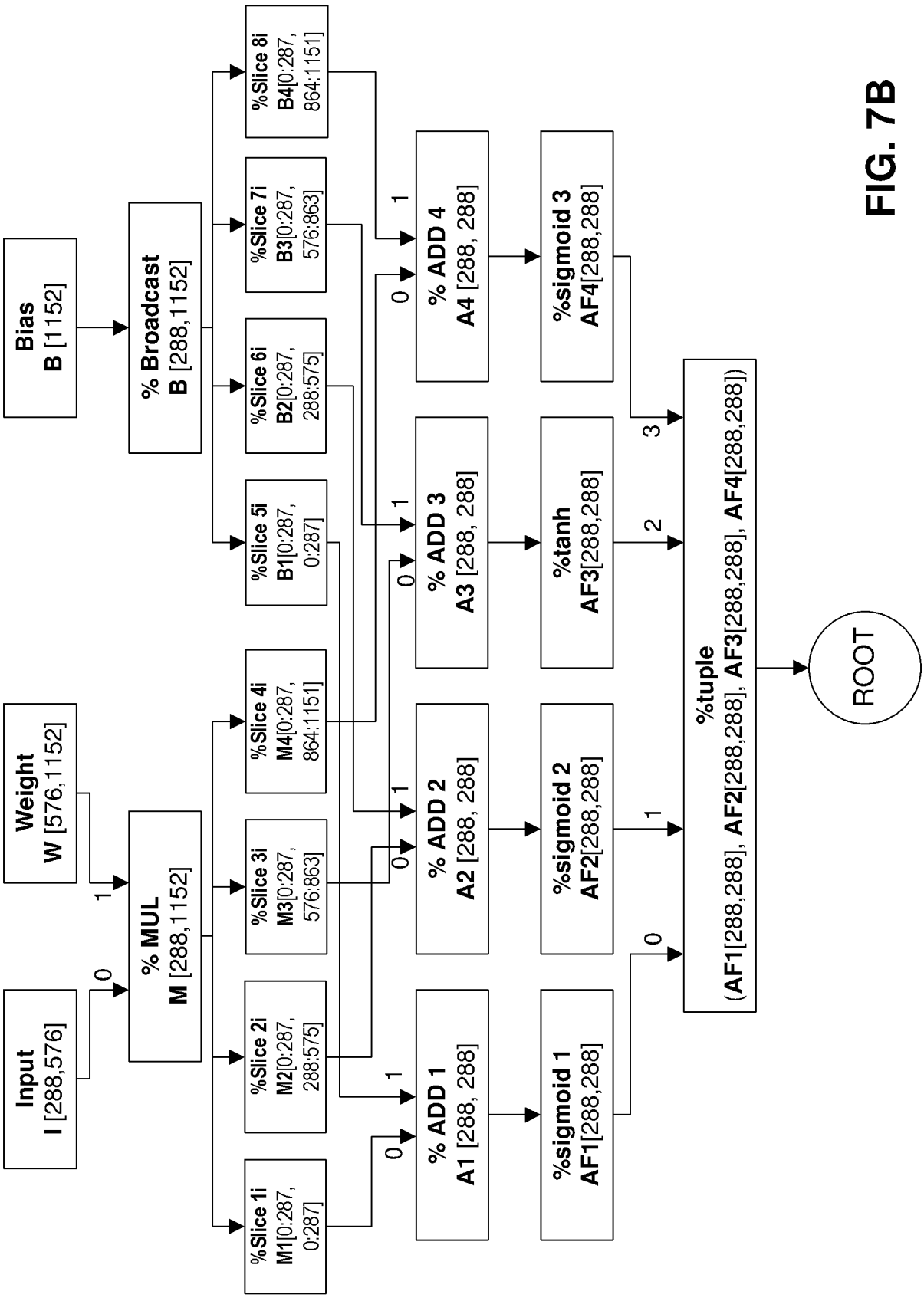
Input
I [288,576]
Weight
W [576,1152]
Bias
B [1152]
% MUL
M [288,1152]
% Broadcast
B [288,1152]
%Slice 1i
M1[0:287, 0:287]
%Slice 2i
M2[0:287, 288:575]
%Slice 3i
M3[0:287, 576:863]
%Slice 4i
M4[0:287, 864:1151]
%Slice 5i
B1[0:287, 0:287]
%Slice 6i
B2[0:287, 288:575]
%Slice 7i
B3[0:287, 576:863]
%Slice 8i
B4[0:287, 864:1151]
% ADD 1
A1 [288, 288]
% ADD 2
A2 [288, 288]
% ADD 3
A3 [288, 288]
% ADD 4
A4 [288, 288]
%sigmoid 1
AF1[288,288]
%sigmoid 2
AF2[288,288]
%tanh
AF3[288,288]
%sigmoid 3
AF4[288,288]
%tuple
(AF1[288,288], AF2[288,288], AF3[288,288], AF4[288,288])
ROOT
0
1
2
3

FIG. 7B

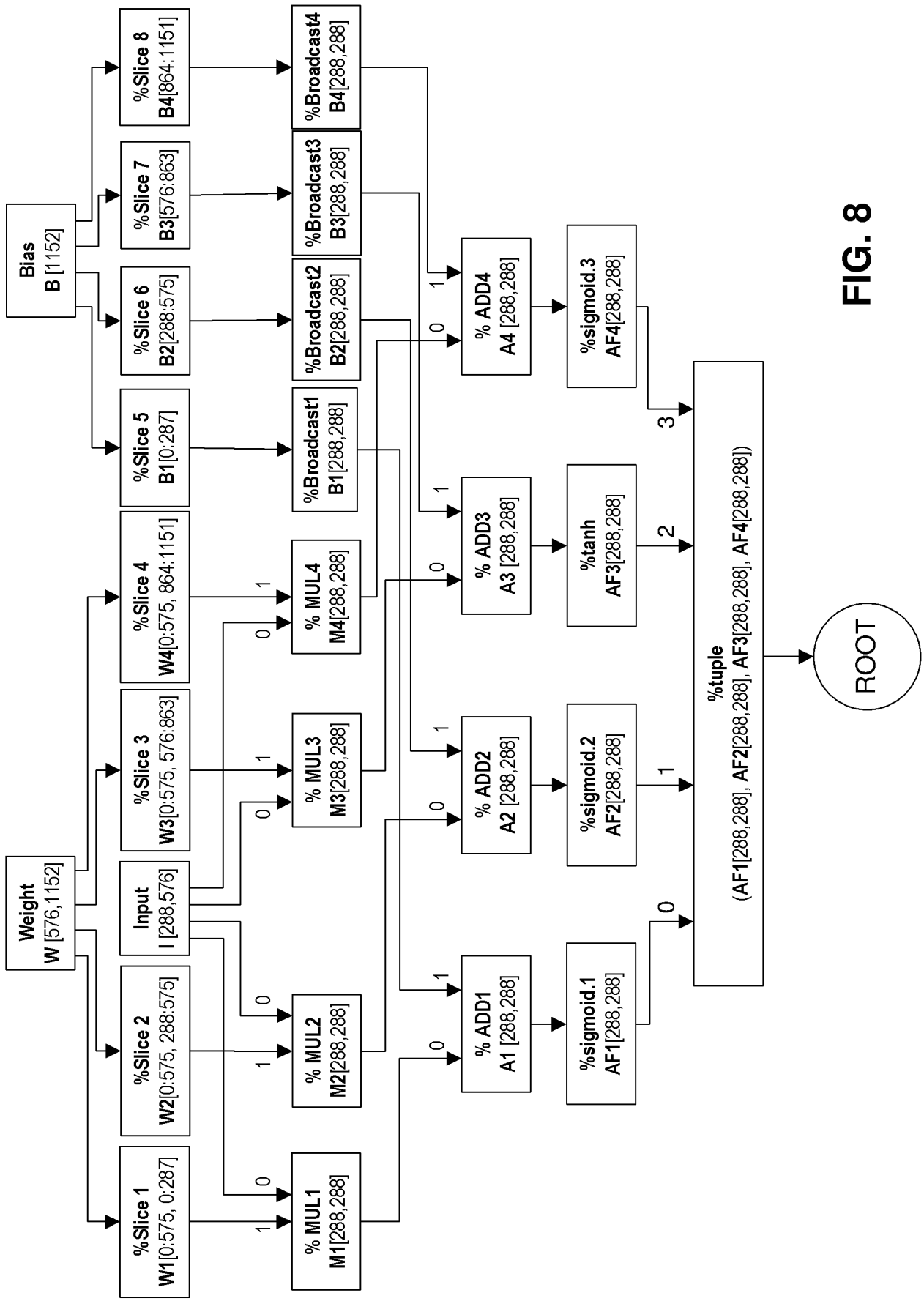
Weight
W [576,1152]
Input
I [288,576]
Bias
B [1152]
%Slice 1
W1[0:575, 0:287]
%Slice 2
W2[0:575, 288:575]
%Slice 3
W3[0:575, 576:863]
%Slice 4
W4[0:575, 864:1151]
%Slice 5
B1[0:287]
%Slice 6
B2[288:575]
%Slice 7
B3[576:863]
%Slice 8
B4[864:1151]
% MUL1
M1[288,288]
% MUL2
M2[288,288]
% MUL3
M3[288,288]
% MUL4
M4[288,288]
%Broadcast1
B1[288,288]
%Broadcast2
B2[288,288]
%Broadcast3
B3[288,288]
%Broadcast4
B4[288,288]
% ADD1
A1 [288,288]
% ADD2
A2 [288,288]
% ADD3
A3 [288,288]
% ADD4
A4 [288,288]
%sigmoid.1
AF1[288,288]
%sigmoid.2
AF2[288,288]
%tanh
AF3[288,288]
%sigmoid.3
AF4[288,288]
%tuple
(AF1[288,288], AF2[288,288], AF3[288,288], AF4[288,288])
ROOT

FIG. 8

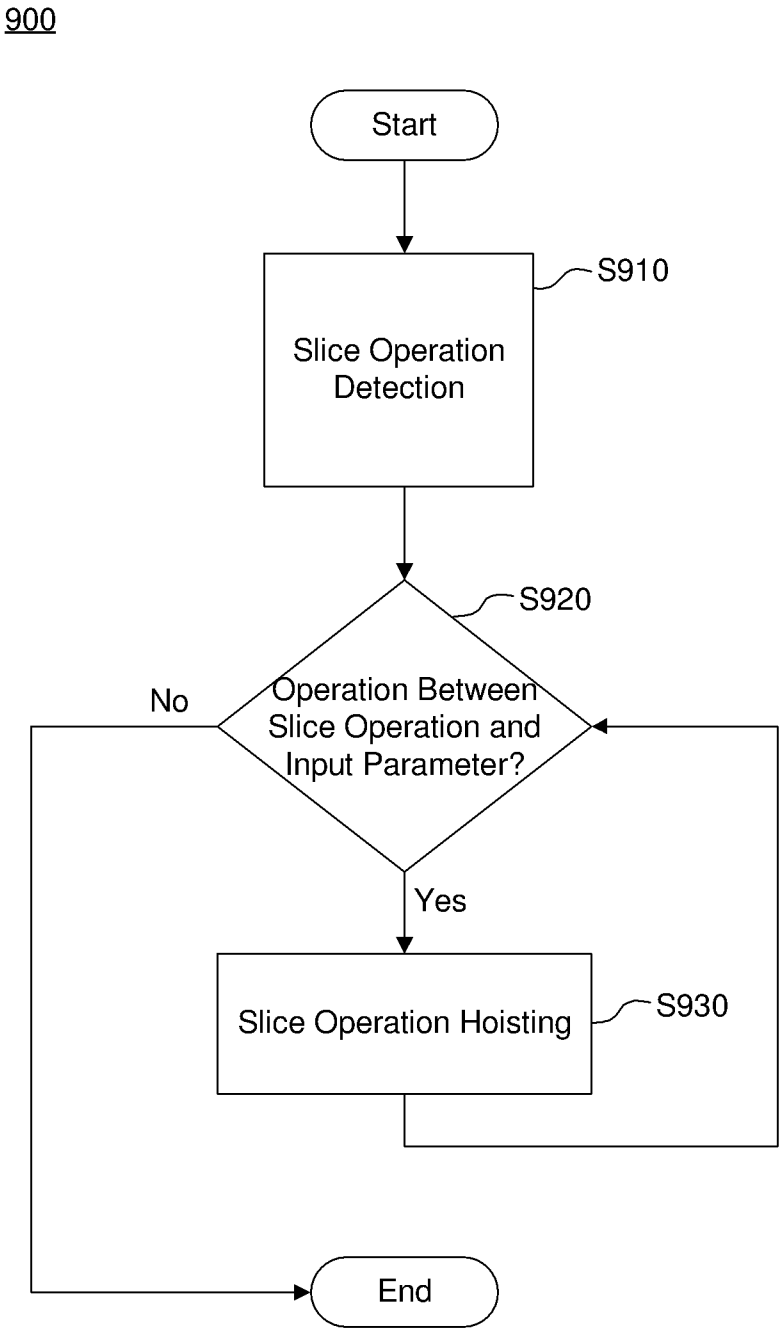
900
Start
S910
Slice Operation Detection
S920
Operation Between Slice Operation and Input Parameter?
No
Yes
Slice Operation Hoisting
S930
End

FIG. 9

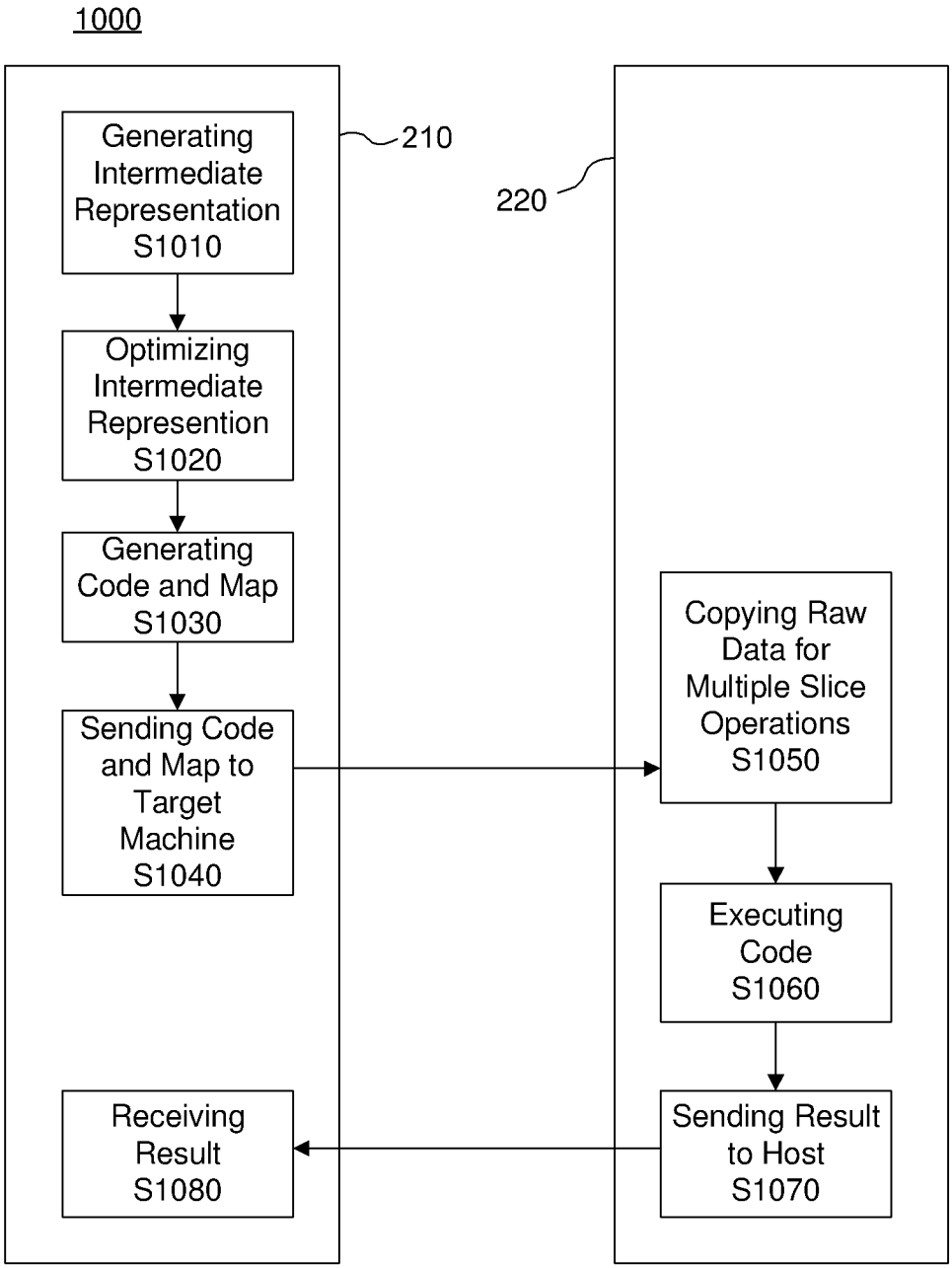
1000
210
220
Generating Intermediate Representation S1010
Optimizing Intermediate Represention S1020
Generating Code and Map S1030
Sending Code and Map to Target Machine S1040
Receiving Result S1080
Copying Raw Data for Multiple Slice Operations S1050
Executing Code S1060
Sending Result to Host S1070

FIG. 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2018/119334

A. CLASSIFICATION OF SUBJECT MATTER

G06F 8/41(2018.01)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)

G06F

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)

WPI;EPODOC;CNPAT;CNKI;IEEE: intermediate, representation, transform, slice, matrix, operation, compile, memory, access

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	Richard Wei etc. "DLVM: A MODERN COMPILER INFRASTRUCTURE FOR DEEP LEARNING SYSTEMS" *arxiv.org/pdf/1711.03016.pdf*, 02 February 2018 (2018-02-02), pages 3-4	1-23
A	US 2018157471 A1 (THE MATHWORKS, INC.) 07 June 2018 (2018-06-07) the whole document	1-23
A	CN 107111503 A (HUAWEI TECHNOLOGY CO., LTD.) 29 August 2017 (2017-08-29) the whole document	1-23
A	US 2009217252 A1 (MICROSOFT CORPORATION) 27 August 2009 (2009-08-27) the whole document	1-23

☐ 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
21 August 2019	**04 September 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**	**XIE,Xin**
Facsimile No. **(86-10)62019451**	Telephone No. **86-(10)-53961366**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2018/119334

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2018157471 A1	07 June 2018	WO 2018094087 A1	24 May 2018
CN 107111503 A	29 August 2017	WO 2016105225 A1	30 June 2016
US 2009217252 A1	27 August 2009	US 2004237074 A1	25 November 2004