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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 November 2016 (24.11.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/186651 A1

(51) International Patent Classification:
G06F 9/48 (2006.01)

(21) International Application Number:
PCT/US2015/031499

(22) International Filing Date:
19 May 2015 (19.05.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventors: **CHEN, Fei**; 1501 Page Mill Road, Palo Alto, California 94304 (US). **JAYARAM, Nandish**; 1501 Page Mill Road, Palo Alto, California 94304 (US). **GONZALEZ DIAZ, Maria Teresa**; 1501 Page Mill Road, Palo Alto, California 94304 (US). **VISWANATHAN, Krishnamurthy**; 1501 Page Mill Road, Palo Alto, California 94304 (US).

(74) Agent: **ESTEBAN, Michelle**; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (US).

(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, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, 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).

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

Published:

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

(54) Title: UPDATING INFERENCE GRAPH ABSENT NODE LOCKING

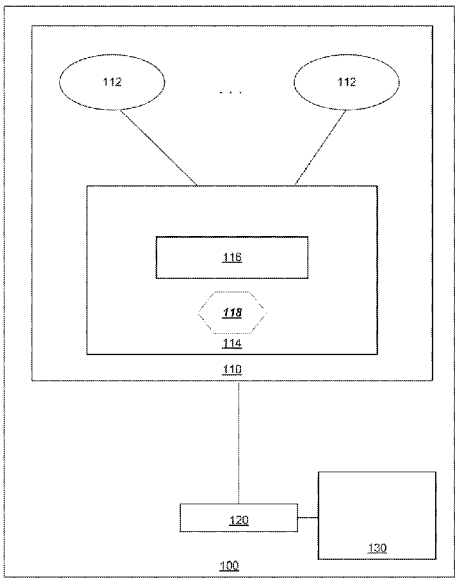

FIG. 1

(57) Abstract: Example implementations relate to updating an inference graph absent node locking. For example, a processor executing a first thread may receive a first task for updating a node of an inference graph stored by a storage device accessible to a second thread, the first task being assigned during a first iteration of a graph update loop. Absent locking the node from access by the second thread, the processor may generate a value for the node and update the node with the value. Based on detecting that each node of the inference graph has been updated, the processor may continue with a second iteration of the graph update loop.

WO 2016/186651 A1

UPDATING INFERENCE GRAPH ABSENT NODE LOCKING

BACKGROUND

[0001] Bioinformatics, computer vision, text analytics, finance and marketing analytics, and other fields may use statistical inferences to generate predictions of expected classifications, labels, or outcomes. Such systems may, by way of example and not limitation, be configured to determine a number of people that are likely to shop at a particular store on a given day, a candidate a voter may likely vote for, and so forth.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Some examples of the present application are described with respect to the following figures:

[0003] FIG. 1 is a schematic diagram of an example computer device for assigning tasks to threads updating an inference graph absent any locks;

[0004] FIG. 2 is a block diagram of an example computing device for processing task lists based on updating values of nodes absent locking the nodes; and

[0005] FIG. 3 is a flowchart of an example method for updating nodes of an inference graph absent locking the nodes.

DETAILED DESCRIPTION

[0006] To generate prediction models, a computer system may use training data that includes a number of samples for a given feature or set of features. Using the training data, the computer system can create, for example, a histogram for a feature that can in turn be used to build or otherwise train the prediction model.

[0007] Graphical models are graphs which compactly represent the joint distributions of random variables. Graph inference algorithms, such as Gibbs sampling, may be used to infer distributions of variables in inference graphs, which are graphical models that compactly represent inferences of joint

distributions of random variables. These algorithms may be used in various applications, such as malware detection, detecting malicious domains, topic modeling, information extraction, image processing analytics, text analytics, and the like. In many of these applications, the graphical models are of large scale, often requiring a large number of random accesses of nodes and/or edges of the graphical model. Parallelization of graph inference algorithms may be used for large scale graph inference, but current techniques may do so inefficiently. As such, running graph inference algorithms, even in parallel, may be time consuming and inefficient, sometimes taking hours to finish on hundreds of machines.

[0008] Various parallelization computing modes may be used to perform graph inference algorithms in multi-core processing environments such that each processor in the multi-core processing environment may concurrently execute a thread associated with the graph inference algorithm, where a thread may be a sequence of programmed instructions that may be managed independently by a scheduler. Each thread may consist of any number of tasks assigned to each thread by a scheduler of the multi-core processing environment. A parallelization computing mode may include a bulk synchronization parallelization (BSP) mode, which may consist of a sequence of super steps performed by threads, where the threads communicate with one another only at the end of each super step at a barrier. A barrier may be a point during processing of a thread at which the processing must stop and may not proceed until all other concurrently processing threads reach the same barrier. After a barrier is reached, each thread obtains a copy of the same consistent global state. Within each super step, each thread may work on their own copy of the global state and may only update their own copy. The global states of each thread may be communicated to other threads at each barrier.

[0009] Another parallelization computing mode may include asynchronous parallelization, where each thread shares the same global data structure (e.g., the topology of the graphical model and the value of each vertex in the model) and each thread updates that global data structure concurrently. Each thread may see updates from other threads as the updates are performed, and each

thread may not need to wait for any other threads in order to process tasks (e.g., tasks may be processed without barriers). At any time, a thread may pick up a task and proceed with processing the task.

[0010] In some examples, graph inference algorithms may be parallelized efficiently using a lock-free in-memory graph inference engine for multi-core environments based on a hybrid computing mode of asynchronous parallelization and BSP. Each thread may share the same global data structures and may update the global state concurrently. Each thread may see updates from other threads as the data structures are updated. A barrier may be enforced periodically such that, when the barrier is reached, a thread may wait for other threads to finish their tasks before that thread may proceed to its next task. When threads update the global state, they may do so without acquiring locks. As such, when a thread updates a node, the thread may perform the update without locking the node.

[0011] For example, a processor core executing a thread may receive a task for updating a node of an inference graph, where the inference graph may be stored by a storage device accessible to other concurrently-running threads. The task may be assigned to that particular thread for a particular iteration of a graph update loop. The graph update loop may be a loop that iterates a graph inference algorithm such that nodes of the inference graph may be updated during each iteration of the loop. When the task assigned to the thread is performed, a value for the node to be updated may be generated, and that node may be updated with that value, without locking the node from access by the other concurrently-running threads. Once each node of the inference graph has been updated during that iteration of the graph update loop, a second iteration of the graph update loop may be performed. A barrier may be used to prevent threads from starting that next iteration until all threads have finished their respective tasks associated with the previous iteration.

[0012] Various sweep strategies may be used to update nodes of an inference graph after each iteration of the graph update loop. A sweep strategy may be an order in which nodes of an inference graph are updated. In a sequential sweep strategy, all of the nodes of the inference graph may be

updated in any order, where the order that is chosen is used for all iterations of the graph update loop. In a shuffling sweep strategy, in each iteration, a random permutation of all of the nodes is generated, and the nodes are updated according to that permutation order for that iteration. In a random sweep strategy, a complete sweep of all nodes is not performed, but instead, at least one node of the inference graph is randomly chosen to be updated for a particular iteration.

[0013] Referring now to the figures, FIG. 1 is a schematic diagram of an example computer device 100 for assigning tasks to threads updating an inference graph absent any locks. The computer device 100 is a physical machine that is constructed from actual machine executable instructions, or "software," and actual hardware. The hardware of the computer device 100 includes a processing node 110, which may be part of, for example, a particular multicore central processing unit (CPU) package or multiple multicore CPU packages. As depicted in FIG. 1, the processing node 110 contains CPU processor cores 112 and a local memory 114. A CPU processor core may be a processor device suitable to read and execute processor executable instructions, such as a CPU, or an integrated circuit configured to perform a configured function.

[0014] The local memory 114 may be a storage device that stores, among other things, CPU executable instructions and data, such as graph update instructions 116 and a plurality of tasks 118. The plurality of tasks 118 may be data that represents any number of tasks to be processed in threads by the CPU processor cores 112. For example, the graph update instructions 116 may be instructions to a first iteration of a graph update loop that, when executed by one of the CPU processor cores, may cause the CPU processor core to select a first subset of tasks and a second subset of tasks from the plurality of tasks 118, where each task in the plurality of tasks 118 corresponds to an update to a respective node of an inference graph. The inference graph may be a shared data structure accessible to both the first thread and the second thread. The graph update instructions 116 may cause the CPU processor core to assign the first subset of tasks to a first thread and assign the second subset of tasks to a second thread, where the first thread

and the second thread update the inference graph absent any locks. The graph update instructions 116 may cause the CPU processor core to execute a second iteration of the graph update loop responsive to detecting that the first thread and the second thread have completed the first subset of tasks and the second subset of tasks.

[0015] The graph update instructions 116 may also include instructions to the second iteration of the graph update loop that, when executed, cause the CPU processor core to select a third subset of tasks and a fourth subset of tasks from the plurality of tasks 118, assign the third subset of tasks to the first thread, and assign the fourth subset of tasks to the second thread. In some examples, the selection of the first subset of tasks, the second subset of tasks, the third subset of tasks, and the fourth subset of tasks is based on an ordering of the nodes of the inference graph. In some examples, the ordering of the nodes of the inference graph may be updated according to a random permutation after detecting that the first thread and the second thread have completed the first subset of tasks and the second subset of tasks.

[0016] FIG. 1 also depicts that the processing node 110 may be coupled to a persistent memory 130 (a non-volatile memory, such as flash memory, for example) that may be accessed by the CPU processor cores 112 via the memory hub 120. The persistent memory 130 may store persistent copies of the graph update instructions 116 and the plurality of tasks 118.

[0017] FIG. 2 is a block diagram of an example computing device 200 for processing task lists based on updating values of nodes absent locking the nodes. In some examples, computing device 200 may be similar to the computer device 100 of FIG. 1.

[0018] Computing device 200 may be, for example, a web-based server, a local area network server, a cloud-based server, a notebook computer, a desktop computer, an all-in-one system, a tablet computing device, a mobile phone, an electronic book reader, a printing device, or any other electronic device suitable for processing task lists based on updating values of nodes absent locking the nodes. Computing device 200 may include a processor 202 and a machine-readable storage medium 204. Computing device 200 may

receive a first task list with tasks corresponding to a subset of nodes in an inference graph and process the first task list based on updating values of the subset of nodes absent locking the subset of nodes.

[0019] Processor 202 is a tangible hardware component that may be a CPU, a semiconductor-based microprocessor, and/or other hardware device suitable for retrieval and execution of instructions stored in machine-readable storage medium 204. Processor 202 may fetch, decode, and execute instructions 206, 208, and 210 to control a process of processing task lists based on updating values of nodes absent locking the nodes. As an alternative or in addition to retrieving and executing instructions, processor 202 may include at least one electronic circuit that includes electronic components for performing the functionality of instructions 206, 208, 210, or a combination thereof.

[0020] Machine-readable storage medium 204 may be any electronic, magnetic, optical, or other physical storage device that contains or stores executable instructions. Thus, machine-readable storage medium 204 may be, for example, Random Access Memory (RAM), an EPROM, an Electrically Erasable Programmable Read-Only Memory (EEPROM), a storage device, an optical disc, and the like. In some examples, machine-readable storage medium 204 may be a non-transitory storage medium, where the term "non-transitory" does not encompass transitory propagating signals. As described in detail below, machine-readable storage medium 204 may be encoded with a series of processor executable instructions 206, 208, and 210 for receiving a first task list, where tasks in the first task list correspond to a subset of nodes in an inference graph, the values of the subset of nodes being stored in a shared data structure accessible to another processor executing a thread; processing the first task list based on updating the values of the subset of nodes absent locking the subset of nodes; and after processing the first task list, waiting for the thread executing on the other processor to finish execution of a second task list.

[0021] Task receipt instructions 206 may manage and control the receipt of task lists with tasks to be executed by the processor 202. For example, task receipt instructions 206 may receive a first task last, where tasks in the

first task list correspond to a subset of nodes in an inference graph and where the values of the subset of nodes are stored in a shared data structure accessible to another processor executing a thread. The other processor may be another processor that may concurrently update the values of the inference graph with the processor 202.

[0022] Task processing instructions 208 may manage and control the processing of task lists. For example, task processing instructions 208 may process the first task list based on updating the values of the subset of nodes absent locking the subset of nodes. In some examples, the subset of nodes may include a first node, and, during an iteration of a graph update loop, the first node may be updated based on a value of a second node, where the second node may be updated by the thread executing on the other processor.

[0023] Wait instructions 210 may wait for threads executing on other processors to finish execution of other task lists. For example, after processing the first task list, the wait instructions 210 may wait for the thread executing on the other processor to finish execution of a second task list. In some examples, a barrier may be used to wait for the thread executing on the other processor to finish the execution of the second task list. In some examples, the first task list and the second task list may correspond to different nodes in the inference graph.

[0024] FIG. 3 is a flowchart of an example method 300 for updating nodes of an inference graph absent locking the nodes. In some examples, method 300 may be implemented using computer device 100 of FIG. 1 and/or computing device 200 of FIG. 2.

[0025] Method 300 includes, at 302, receiving a first task for updating a node of an inference graph stored by a storage device accessible to a second thread, where the first task is assigned during a first iteration of a graph update loop. The first task may be part of a subset of a plurality of tasks, where each task from the plurality of tasks is assigned to update a respective node of the inference graph.

[0026] Method 300 also includes, at 304, absent locking the node from access by the second thread, generating a value for the node. The value of

the node may depend on values of nodes neighboring the node. For example, the value of the node may be a value that is based on and/or derived from values of those neighboring nodes.

[0027] Method 300 also includes, at 306, absent locking the node from access by the second thread, updating the node with the value.

[0028] Method 300 also includes, at 308, based on detecting that each node of the inference graph has been updated, continuing with a second iteration of the graph update loop. In some examples, detecting that each node of the inference graph has been updated may include coordinating the first thread and the second thread with a barrier. For example, the first thread and the second thread may execute in coordination based on a barrier.

[0029] In some examples, as part of the second iteration, a second task for updating a different node of the inference graph may be received. Absent locking the different node from access by the second thread, a value for the different node may be generated, and the different node may be updated with the value. Based on detecting that each node of the inference graph has been updated, a third iteration of the graph update loop may be continued.

[0030] Examples provided herein (e.g., methods) may be implemented in hardware, software, or a combination of both. Example systems may include a controller/processor and memory resources for executing instructions stored in a tangible non-transitory medium (e.g., volatile memory, non-volatile memory, and/or machine-readable media). Non-transitory machine-readable media can be tangible and have machine-readable instructions stored thereon that are executable by a processor to implement examples according to the present disclosure.

[0031] An example system can include and/or receive a tangible non-transitory machine-readable medium storing a set of machine-readable instructions (e.g., software). As used herein, the controller/processor can include one or a plurality of processors such as in a parallel processing system. The memory can include memory addressable by the processor for execution of machine-readable instructions. The machine-readable medium can include volatile and/or non-volatile memory such as a random access

memory ("RAM"), magnetic memory such as a hard disk, floppy disk, and/or tape memory, a solid state drive ("SSD"), flash memory, phase change memory, and the like.

Claims

What is claimed is:

1. A method comprising:

receiving, by a processor executing a first thread, a first task for updating a node of an inference graph stored by a storage device accessible to a second thread, the first task being assigned during a first iteration of a graph update loop;

absent locking the node from access by the second thread:

generating, by the processor, a value for the node, and

updating, by the processor, the node with the value; and

based on detecting that each node of the inference graph has been updated, continuing with a second iteration of the graph update loop.

2. The method of claim 1, wherein the first task is part of a subset of a plurality of tasks, each task from the plurality of tasks being assigned to update a respective node of the inference graph.

3. The method of claim 1, further comprising, as part of the second iteration:

receiving, by the processor, a second task for updating a different node of the inference graph;

absent locking the different node from access by the second thread:

generating, by the processor, a value for the different node, and

updating, by the processor, the different node with the value;

and

based on detecting that each node of the inference graph has been updated, continuing with a third iteration of the graph update loop.

4. The method of claim 1, wherein the value for the node depends on values of nodes neighboring the node.

5. The method of claim 1, wherein detecting that each node of the inference graph has been updated comprises coordinating the first thread and the second thread with a barrier.

6. A device comprising:

a processor; and

a machine-readable storage device comprising instructions to a first iteration of a graph update loop that, when executed, cause the processor to:

select a first subset of tasks and a second subset of tasks from a plurality of tasks, each task from the plurality of tasks corresponds to an update to a respective node of an inference graph;

assign the first subset of tasks to a first thread;

assign the second subset of tasks to a second thread, wherein the first thread and the second thread update the inference graph absent any locks; and

execute a second iteration of the graph update loop responsive to detecting that the first thread and the second thread have completed the first subset of tasks and the second subset of tasks.

7. The device of claim 6, wherein the inference graph is a shared data structure accessible to both the first thread and the second thread.

8. The device of claim 6, wherein the machine-readable storage device comprises instructions to the second iteration of the graph update loop that, when executed, cause the processor to:

select a third subset of tasks and a fourth subset of tasks from the plurality of tasks;

assign the third subset of tasks to the first thread; and

assign the fourth subset of tasks to the second thread.

9. The device of claim 8, wherein the selection of the first subset of tasks, the second subset of tasks, the third subset of tasks, and the fourth subset of tasks is based on an ordering of the nodes of the inference graph.

10. The device of claim 8, wherein the machine-readable storage device comprises instructions that, when executed, cause the processor to update the ordering of the nodes of the inference graph according to a random permutation after detecting that the first thread and the second thread have completed the first subset of tasks and the second subset of tasks.

11. A machine-readable storage device comprising instructions that, when executed, cause a processor to:

receive a first task list, tasks in the first task list corresponding to a subset of nodes in an inference graph, the values of the subset of nodes being stored in a shared data structure accessible to another processor executing a thread;

process the first task list based on updating the values of the subset of nodes absent locking the subset of nodes; and

after processing the first task list, wait for the thread executing on the another processor to finish execution of a second task list.

12. The machine-readable storage device of claim 11, wherein the processor and the another processor concurrently update the values of the inference graph.

13. The machine-readable storage device of claim 11, wherein the subset of nodes includes a first node, and wherein the instructions, when executed, further cause the processor to, during an iteration of a graph update loop, update the first node based on a value of a second node, the second node being updated by the thread executing on the another processor.

14. The machine-readable storage device of claim 11, wherein the instructions, when executed, further cause the processor to use a barrier to wait for the thread executing on the another processor to finish the execution of the second task list.

15. The machine-readable storage device of claim 11, wherein the first task list and the second task list correspond to different nodes in the inference graph.

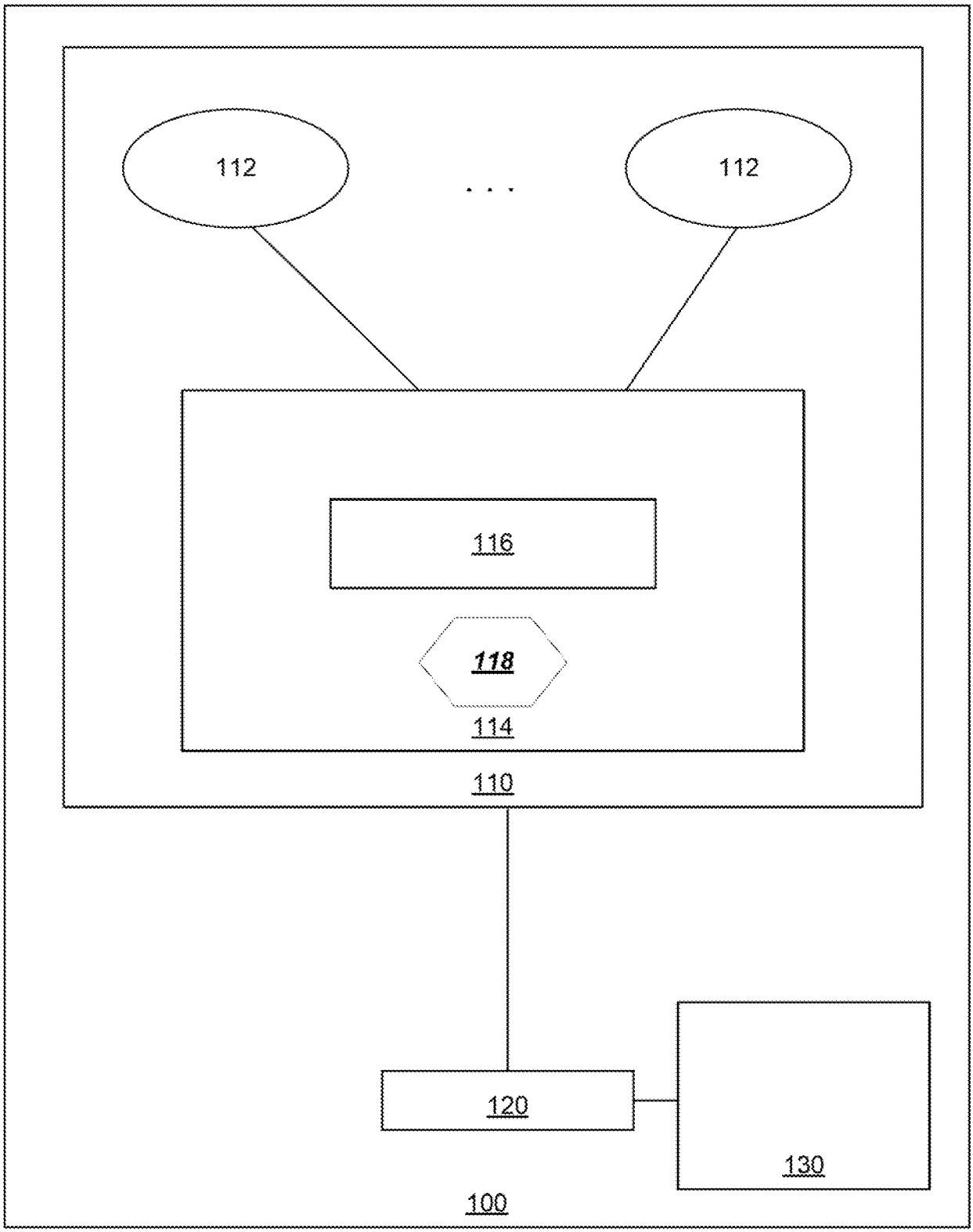

FIG. 1

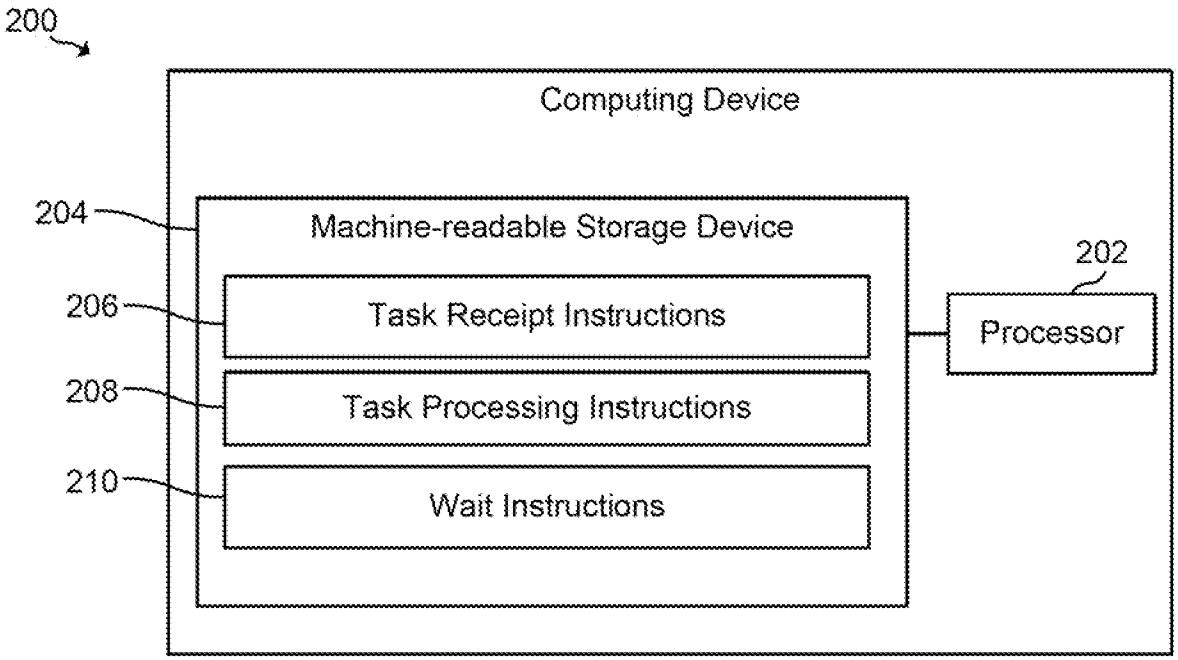
200
Computing Device
204
Machine-readable Storage Device
206
Task Receipt Instructions
208
Task Processing Instructions
210
Wait Instructions
202
Processor

FIG. 2

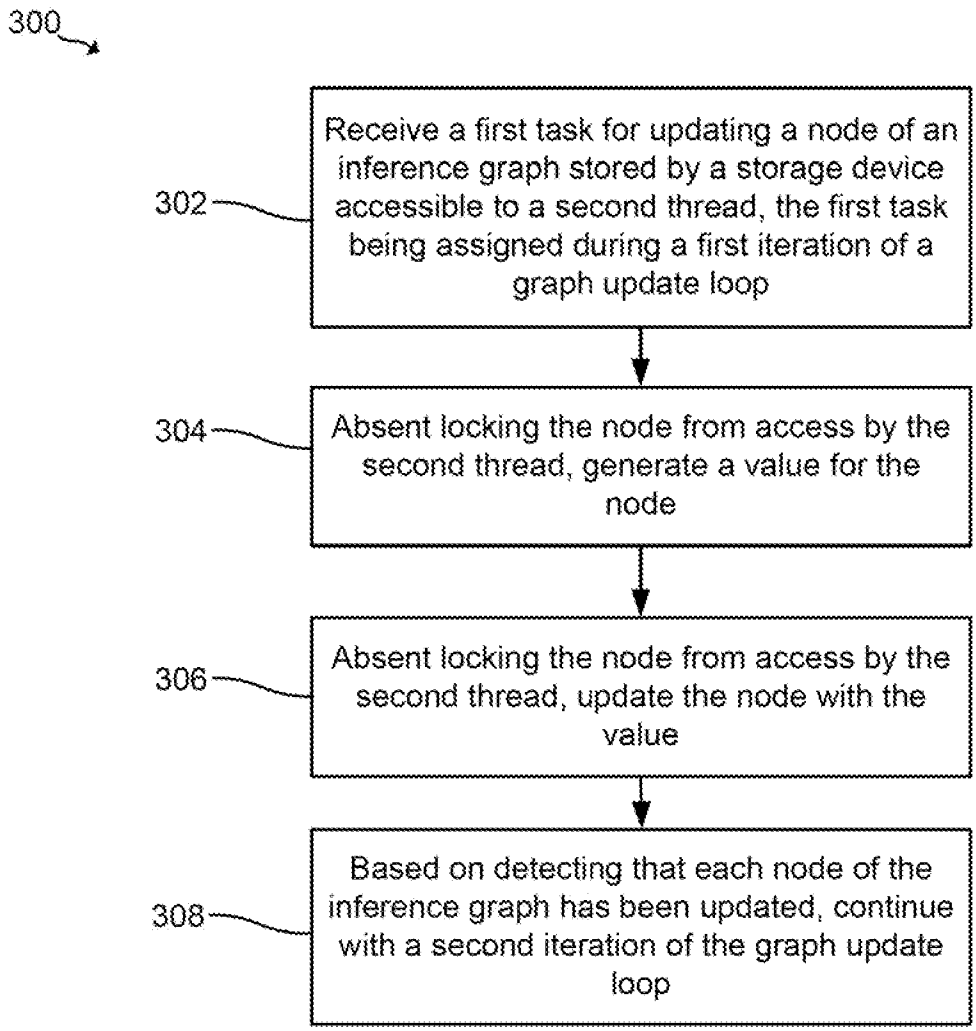
300
302
Receive a first task for updating a node of an inference graph stored by a storage device accessible to a second thread, the first task being assigned during a first iteration of a graph update loop
304
Absent locking the node from access by the second thread, generate a value for the node
306
Absent locking the node from access by the second thread, update the node with the value
308
Based on detecting that each node of the inference graph has been updated, continue with a second iteration of the graph update loop

FIG. 3

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/031499

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/48(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)
G06F 9/48; G06F 9/46; G06F 12/00; G06F 9/50

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: inference, graph, update, node locking

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2012-0297394 A1 (ALLEN, G. M. W.) 22 November 2012 See abstract; paragraphs [0042]-[0044]; and fig. 6.	1-15
A	US 2014-0013330 A1 (WANG, C. et al.) 09 January 2014 See abstract; paragraphs [0017]-[0022]; and fig. 1.	1-15
A	US 2013-0074080 A1 (JIMENEZ, J. C.) 21 March 2013 See abstract; paragraphs [0030]-[0038]; and figs. 5 and 6.	1-15
A	US 2013-0132961 A1 (LEHAVI, D. et al.) 23 May 2013 See abstract; paragraphs [0008]-[0025]; and fig. 2.	1-15
A	US 2010-0131953 A1 (DICE, D. et al.) 27 May 2010 See abstract; paragraphs [0049]-[0053]; and fig. 3.	1-15

☐ 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
03 February 2016 (03.02.2016)	**04 February 2016 (04.02.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
YU, Jintae
Telephone No. +82-42-481-8530

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/031499

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012-0297394 A1	22/11/2012	US 2013-0305253 A1	14/11/2013
		US 8850441 B2	30/09/2014
		US 8863136 B2	14/10/2014
US 2014-0013330 A1	09/01/2014	None	
US 2013-0074080 A1	21/03/2013	CN 102937916 A	20/02/2013
		US 9229770 B2	05/01/2016
US 2013-0132961 A1	23/05/2013	US 8887160 B2	11/11/2014
US 2010-0131953 A1	27/05/2010	US 8776063 B2	08/07/2014