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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 May 2017 (26.05.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/086986 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US2015/061798

(22) International Filing Date:
20 November 2015 (20.11.2015)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventors: **GHOSHAL, Asish**; 131 Nimitz Dr., Apt. #15, West Lafayette, Indiana 47906 (US). **LI, Jun**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **MARWAH, Manish**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **KIM, Mijung**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **ULANOV, Alexander**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US).

(74) Agents: **HARTMANN II, Kenneth R.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road Mail Stop 79, Fort Collins, Colorado 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,

[Continued on next page]

(54) Title: PARTITIONED GRAPH PROCESSING

(57) Abstract: Methods, apparatus, systems and articles of manufacture involved in processing graph data distributed in partitions of a memory. In examples herein, a graph partition manager manages processing iterations of local data in each partition executed by local graph updaters of the partitions. Replicate node data is aggregated by the graph partition manager and returned to the local graph updaters for subsequent iterations.

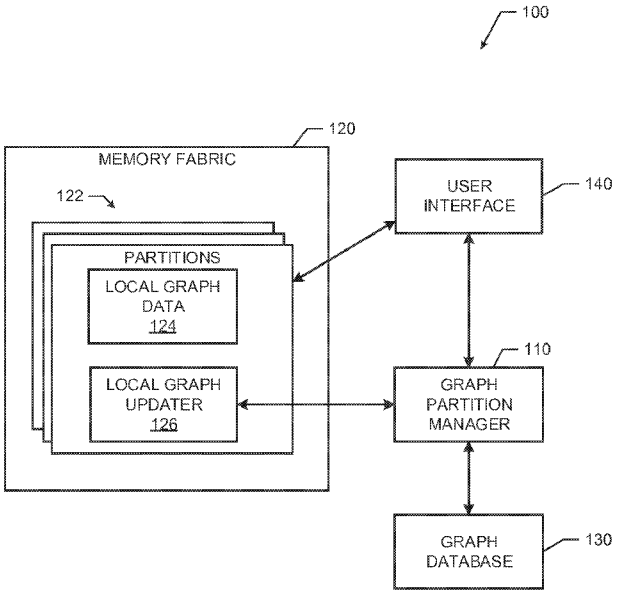


FIG. 1

WO 2017/086986 A1

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))*

PARTITIONED GRAPH PROCESSING

BACKGROUND

[0001] A graph is a representation of a set of data (e.g., Big Data). An example graph may include a plurality of nodes and edges connecting the plurality of edges. The graph may be processed by executing nodes in accordance with characteristics of edges linked to the nodes and related nodes linked by the edges.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] FIG. 1 is a schematic diagram of an example partitioned graph processing system including an example partition manager constructed in accordance with examples herein.

[0003] FIG. 2 is a block diagram of an example graph partition manager that may be used to implement the graph partition manager of FIG. 1.

[0004] FIG. 3 is a flowchart representative of example machine readable instructions that may be executed to implement the graph partition manager of FIG. 2.

[0005] FIG. 4 is another flowchart representative of other example machine readable instructions that may be executed to implement graph partition manager of FIG. 2.

[0006] FIG. 5 is a block diagram of an example processor platform capable of executing the instructions of FIGS. 3 and/or 4 to implement the graph partition manager of FIG. 2.

[0007] Wherever possible, the same reference numbers will be used throughout the drawing(s) and accompanying written description to refer to the same or like parts.

DETAILED DESCRIPTION

[0008] Examples disclosed herein involve processing a graph having graph data distributed across a plurality of partitions of a memory (e.g., a shared memory). In examples herein, local graph data of the partitions is processed by local graph updaters. Replicate node data within the graph data (e.g., node data for a same node that is in a plurality of partitions) is aggregated for each iteration and returned to the local graph updaters for subsequent iterations.

[0009] In graph processing, cluster-based computing environments include a relatively small number of processing units with relatively small amounts of memory in each cluster. In examples herein, a multi-core environment, in combination with a shared memory enable relatively short inter-process communication latency while holding increased amount of in-memory data structures that provide better data locality. Accordingly, examples herein enable increased speeds for graph processing over distributed partitions that run on multi-core and shared memory machines by having several local graph updaters process local graph data of corresponding data partitions of a shared memory.

[0010] An example method includes receiving node data for replicate nodes of a graph. The example node data may be generated from first processing iterations of graph data distributed in a plurality of graph partitions received from respective local graph updaters of the plurality of graph partitions of a shared memory; aggregating beliefs of the processed replicate nodes; and instructing the local graph updaters of the graph partitions to update beliefs of the replicate nodes for second processing iterations of the graph partitions.

[0011] As used herein, graph processing (or processing a graph) involves processing nodes of a graph until all nodes (or a threshold percentage of the nodes) have been processed, which is known as convergence of the graph processing. Convergence occurs when all nodes (or a threshold percentage of the nodes) have been processed such that a particular characteristic of the nodes has reached a fixed point (or value), and, thus, does not change with subsequent iterations. In examples herein, parallel graph processing or

processing graphs in parallel (e.g., synchronous graph processing) may involve several iterations of processing a set of nodes in parallel (at a same time or within a same time period). For example, in a first iteration, a first set of nodes may be processed and in a second iteration a second set of nodes may be processed. In examples herein, the second set of nodes may or may not include nodes related or linked to the first set of nodes.

[0012] As used herein, asynchronous graph processing involves processing graphs based on priority of nodes of a graph. The example priority or priority of (or associated with) a node refers to an importance that the node be processed sooner rather than later. For example, it is desirable that a node having a higher priority be processed before a node having a lower priority. Priority may be ranked or rated on any suitable scale, (e.g., from 1 to 10, 10 being highest priority, 1 being lowest priority (or vice versa). And finally, hybrid graph processing or hybrid synchronous/asynchronous graph processing refers to a combination of synchronous graph processing and asynchronous graph processing to process a graph.

[0013] Furthermore, in examples herein, graph data of a graph includes edge data and node data corresponding to the edges and nodes of the graph, respectively. The edge data includes edge information, such as message data, characteristic data, etc. The node data includes node information, such as node characteristics, node attributes, priority, etc.

[0014] FIG. 1 is a schematic diagram of an example partitioned graph processing system 100 including an example graph partition manager 110 constructed in accordance with examples herein. The example partitioned graph processing system 100 includes the graph partition manager 110, a memory fabric 120, a graph database 130, and a user interface 140. The example memory fabric 120 includes a plurality of partitions 122 (which may be referred to collectively herein as the partitions 122). Each of the plurality of partitions include local graph data 124 and a local graph updater 126 (e.g., an iteration processor, a virtual machine, etc.). In examples herein, the graph partition manager 110 facilitates graph processing of graph data distributed among the plurality of partitions 122.

[0015] The example user interface 140 facilitates user interaction with the graph partition manager 110 and/or the memory fabric 120. For example, the user interface 140 may include user input(s) (e.g., a keyboard, a mouse, a trackball, a touchscreen, etc.) and/or user output(s) (e.g., a display, a touchscreen, speakers, indicators, etc.). In examples herein, the user interface 140 enables a user to access and control settings of the graph partition manager 110. For example, the user interface 140 may request processing of a particular graph from the graph database 130. Furthermore, the user interface 140 enables a user to indicate or set processing settings (e.g., synchronous settings, asynchronous settings, hybrid settings, etc.) to process the selected graph or particular types of graphs/graph data in accordance with examples herein.

[0016] The example graph database 130 stores graph data for graphs. The example graph database 130 may be implemented by at least a storage device, computing device, network device, etc. that may store or provide graph data for graphs that are to be processed in accordance with examples herein. The example graph partition manager 110 may receive/retrieve graph data for graphs from the graph database 130. In some examples, the graph partition manager 110 may partition the graph data into the plurality of partitions 122 of the memory fabric 120. Accordingly, the graph partition manager 110 may load the partitioned graph data into respective partitions of the plurality of partitions 122 using any suitable technique.

[0017] The example memory fabric 120 of FIG. 1 stores the graph data in the partitions 122 of the memory fabric. The memory fabric 120 may be implemented by a non-volatile memory (NVM), such as a memristor, a phase change memory, a dynamic random access memory (DRAM), or any other suitable memory fabric/device. In some examples, the memory fabric 120 may be a shared memory fabric that is accessible to a plurality of processing engines, devices, or systems (e.g., a plurality of graph partition managers 110, a plurality of shuffle engines, etc.). The example partitions 122 of the memory fabric 120 store graph data for a graph or a plurality of graphs. For the sake of readability, though examples herein may refer to graph data for a single graph

being distributed across the partitions 122, the partitions 122 may store graph data for a plurality of graphs.

[0018] In examples herein, the graph partition manager 110 may partition a graph from the graph database 130 such that edge data for edges of a graph is distributed across different partitions of the plurality of partitions 122. Accordingly, edge data for a same edge may not be located in more than one partition of the plurality of partitions 122. Furthermore, node data corresponding to nodes linked by edges of the edge data is included in the corresponding partitions 122. Thus for example, if edge data for a first edge is located in a first partition of the plurality of partitions 122, node data corresponding the nodes linked by the first edge is also stored in the first partition. Thus, in examples herein, because the nodes linked by the first edge may also be linked to other nodes via additional edges, node data for the same nodes may be replicated (i.e., duplicated) across multiple partitions of the plurality of partitions 122. Such nodes may be referred to herein as replicate nodes.

[0019] The graph partition manager 110 facilitates graph processing of graph data for a graph stored in the plurality of partitions 122 by accounting for replicated node data in the plurality of partitions 122. In some examples, the graph partition manager 110 may partition graph data for a graph from the graph database 130 into the plurality of partitions 122 in order to lessen (e.g., lower or minimize) a number of replicate nodes across the plurality of partitions 122. For example, some nodes may have an order of magnitude greater number of edges than other nodes (i.e., more edges may be associated with some nodes that others). Accordingly, in such an example, edge data for edges of the nodes may be distributed across the partitions 122 (along with node data corresponding to those edges) to relatively minimize the number of replicate nodes. The example graph partition manager 110 may identify (e.g., flag, mark, etc.) replicate nodes so that the local graph updater 126 may identify replicate nodes when performing local processing of the local graph data 124).

[0020] The example partitions 122 each include local graph data 124 and a local graph updater 126. The local graph updater 126 iteratively processes the example local graph data 124. In examples herein, each process iteration

may involve the local graph updater 126 processing the local graph data 124 (e.g., calculating beliefs of nodes of edges in the local graph data 124) and a reconciliation process implemented by the graph partition manager, as further described below to account for replicate nodes.

[0021] In examples herein, the local graph updater 126 may perform a belief propagation calculation on the local graph data 124. For example, the local graph updater 126 may aggregate message data of edges for each node in the local graph data. For example, the local graph updater 126, for each node *i* may multiply a node potential p_i with a product of all messages from neighbor nodes to calculate belief b_i for each node, as follows:

$$b_i = p_i \prod_{j \in \Gamma(i)} m_{ji} \quad (1)$$

wherein, $\Gamma(i)$ denotes the neighbor nodes of node *i*. Subsequently, the local graph updater 126 may send messages from each node *i* to neighbor nodes *j*. The example message may be:

$$m_{ij} = \sum \frac{b_i}{m_{ji}} \Psi_{ij} \quad (2)$$

wherein Ψ_{ij} is the potential defined on the edge from vertex *i* to vertex *j*, and the summation is over the states of the node *i*.

[0022] In examples herein, for non-replicate nodes (i.e., nodes that are in a single partition of the partitions 122), the local graph updater 126 may perform each partition locally within the corresponding partition 122. However, for replicate nodes (which may be identified by a node identifier, flag, mark, table, etc.), all edges may not be incident on the node locally within a partition (as the edges are distributed across the partitions 122). Accordingly, for replicate nodes, the local graph updaters 126 may skip replicate nodes and but still mark such nodes as processed.

[0023] The graph partition manager 110 of FIG. 1 may determine whether convergence of a graph has been reached. For example, the local graph updaters 126 may provide convergence information to the graph partition manager 110 when local graph data 124 of the corresponding local graph updater 126 has converged. Accordingly, the graph partition manager 110 may monitor when the local graph data 124 for all of the partitions 122 (or at least a

threshold percentage of the partitions 122) has converged, and end graph processing of the graph.

[0024] FIG. 2 is a block diagram of an example graph partition manager 110 that may be used to implement the graph partition manager 110 of FIG. 1. The example graph partition manager 110 of FIG. 2 includes an update receiver 210, a replicate aggregator 220, and partition updater 230. In examples herein, graph partition manager 110 receives/retrieves sets of updated node data corresponding to replicate nodes marked as processed by the local graph updaters 126 during iterations of processing the local graph data 124. The updated node data is received from the plurality of graph partitions 122 via the update receiver 210. The example replicate aggregator 220 aggregates node data for the set of replicate nodes, and the partition updater 230 instructs the local graph updaters 126 to update the node data for the replicate nodes with the aggregated node data of the replicate nodes for subsequent iterations.

[0025] The example update receiver 210 may serve as a buffer for receiving updated graph data from the local graph updaters 126 of the plurality of graph partitions 122. The replicate aggregator 220 aggregates the updated node data from the graph data that corresponds to replicate nodes. In examples herein, replicate nodes may have been identified (e.g., following a partitioning of the graph by the graph partition manager 110 or other graph manager) and/or marked using any suitable techniques. In some examples, the graph partitioning manager 110 may dynamically partition the graph after each iteration. In such examples, the replicate nodes may be different for each iteration, and the replicate aggregator 220 may need to analyze the node data to identify replicate nodes (e.g., by identifying same node data corresponding to a same node in multiple partitions, by determining calculated hash codes from node data for nodes from separate partitions are the same, etc.). Accordingly, the graph partition manager 110 may determine and/or identify replicate nodes in the graph data form the local graph updaters 126 using any suitable techniques.

[0026] In examples herein, the replicate aggregator 220 of FIG. 2 may aggregate the node data for the replicate nodes by adding, combining,

averaging, etc. the updated data such that different/separate updates to the node data in each partition for each iteration is considered/taken into account for a subsequent iteration. Once the node data for the replicate nodes is aggregated, the replicate aggregator 220 provides the aggregated replicate node data to the partition updater 230. In examples herein, the replicate aggregator 220 may be implemented by a shuffle engine by retrieving (e.g., mapping into buckets) node data for replicate nodes (e.g., nodes marked as replicate nodes) from a processing iteration, and reducing the replicate nodes to generate by aggregating the replicate node data.

[0027] The example partition updater 230 provides the aggregated replicate node data back to the local updaters 126 of the 122 of FIG. 1. From there, the local updaters 126 may update the node data corresponding to the replicate nodes in the partition for any subsequent iterations. For example, the local updater 126 may recalculate the above belief propagation calculations of Equations (1) and Equation (2) for the above using the aggregated node data (e.g., aggregated beliefs) for the replicate nodes. Upon recalculating the above, the local updaters 126 may continue to a subsequent processing iteration of the local graph data 124.

[0028] In some examples herein, the partition updater 230 may provide update strategies (e.g., graph processing types) to be implemented by the local graph updaters. For example, the partition updater 230 may instruct the local graph updaters 126 to process the local graph data 122 using synchronous graph processing, asynchronous graph processing, or a hybrid synchronous/asynchronous graph processing. Examples herein may consider the following when selecting or providing instructions to the local graph updaters 126 to use particular update strategies (graph processing types): communication volume, number of iterations to reach convergence, and total time. In examples herein, the synchronous graph processing strategy processes all nodes once in each partition which may involve relatively increased communication volume with relatively less iterations to reach convergence. The asynchronous processing strategy involves processing a subset of unconverged nodes in each partition. The example set of nodes for

asynchronous processing may be selected from a priority queue having the unconverged nodes of the local graph data 122 in decreasing order of residue (a greatest difference between two successive messages received over all neighbor nodes). The hybrid synchronous/asynchronous graph processing strategy combines the synchronous and asynchronous strategy which may involve communication volume and number of iterations between the synchronous and asynchronous strategies, but lower total time to reach convergence.

[0029] Additionally or alternatively, the partition updater 230 may instruct the local graph updaters 126 to switch from one update strategy used for a previous iteration to another updater strategy for a subsequent iteration. For example, in the hybrid synchronous/asynchronous strategy, a first number of iterations may be synchronous and the remaining iterations may be asynchronous. The example number of synchronous operations may be tunable or adjustable depending on the graph type (which may be empirically determined). In some example, empirical data may be generated for graph types to determine appropriate (e.g., optimal) update strategies for processing graph data distributed across the partitions 122 in accordance with examples herein.

[0030] Furthermore, processing or update scheduling may be tuned to selectively update a proportion (or percentage) of replicate nodes and non-replicated. For example, for a particular one of the partitions 122, in each iteration, a certain number of replicate nodes and non-replicate nodes may be processed for asynchronous updates where the number of updates is $\mathrm{Min}\left(\alpha(u_r) + (1-\alpha)(u-u_r),\ K\right)$ where α is a percentage of the replicate nodes processed, u_r is the number of replicate nodes, u is the total number of nodes, and K is the max value for the updates. The percentage of replicate nodes processed (α) may be empirically determined to find a suitable (e.g., optimal) update strategy.

[0031] Accordingly, in examples herein, a processing iteration to calculate belief propagation of nodes of a graph distributed among the partitions 122 includes the local updater 126 updating the local graph data 124 (e.g.,

calculating the beliefs of the nodes in the local graph data by processing the edge data of edges of the nodes and the node data for the nodes), providing the updated local graph data to the graph partition manager 110 to update the local graph data with aggregated node data for the replicate nodes, and continuing to the next iteration (and/or checking for convergence of the graph).

[0032] While an example manner of implementing the graph partition manager 110 of FIG. 1 is illustrated in FIG. 2, at least one of the elements, processes and/or devices illustrated in FIG. 2 may be combined, divided, re-arranged, omitted, eliminated and/or implemented in any other way. Further, the update receiver 210, the replicate aggregator 220, the partition updater 230, and/or, more generally, the example graph partition manager 110 of FIG. 2 may be implemented by hardware and/or any combination of hardware and executable instructions (e.g., software and/or firmware). Thus, for example, any of the update receiver 210, the replicate aggregator 220, the partition updater 230, and/or, more generally, the example graph partition manager 110 could be implemented by at least one of an analog or digital circuit, a logic circuit, a programmable processor, an application specific integrated circuit (ASIC), a programmable logic device (PLD) and/or a field programmable logic device (FPLD). When reading any of the apparatus or system claims of this patent to cover a purely software and/or firmware implementation, at least one of the update receiver 210, the replicate aggregator 220, and/or the partition updater 230 is/are hereby expressly defined to include a tangible machine readable storage device or storage disk such as a memory, a digital versatile disk (DVD), a compact disk (CD), a Blu-ray disk, etc. storing the executable instructions. Further still, the example graph partition manager 110 of FIG. 2 may include at least one element, process, and/or device in addition to, or instead of, those illustrated in FIG. 2, and/or may include more than one of any or all of the illustrated elements, processes and devices.

[0033] Flowcharts representative of example machine readable instructions for implementing the graph partition manager 110 of FIG. 2 is shown in FIGS. 4 and 5. In this example, the machine readable instructions comprise a program/process for execution by a processor such as the

processor 512 shown in the example processor platform 500 discussed below in connection with FIG. 5. The program/process may be embodied in executable instructions (e.g., software) stored on a tangible machine readable storage medium such as a CD-ROM, a floppy disk, a hard drive, a digital versatile disk (DVD), a Blu-ray disk, or a memory associated with the processor 512, but the entire program/process and/or parts thereof could alternatively be executed by a device other than the processor 512 and/or embodied in firmware or dedicated hardware. Further, although the example program is described with reference to the flowchart illustrated in FIG. 4 and/or 5, many other methods of implementing the example graph partition manager 110 may alternatively be used. For example, the order of execution of the blocks may be changed, and/or some of the blocks described may be changed, eliminated, or combined.

[0034] The example process 300 of FIG. 3 begins with an initiation of the graph partition manager 110 (e.g., upon startup, upon instructions from a user, upon startup of a device implementing the graph partition manager 110 (e.g., the partitioned graph processing system 100), etc.). The example process 300 may be executed to process a graph having graph data distributed across a plurality of partitions (e.g., the partitions 122) in accordance with examples herein. At block 310, the update receiver 210 receives replicated nodes of a graph in node data generated from first processing iterations of graph data distributed in the plurality of graph partitions 122. The example first processing iterations of block 310 may be belief propagation calculations in each of the partitions 122. At block 320, the example replicate aggregator 220 aggregates (e.g., reduces) beliefs for the replicated nodes.

[0035] At block 330, the partition updater 230 instructs the local graph updaters 126 of the graph partitions to update beliefs of the replicated nodes for second processing iterations of the graph partitions (e.g., a subsequent processing iteration of the graph). After block 330, the example process 300 ends, although the example process 300 may be iteratively executed for a subsequent iteration of processing the graph.

[0036] The example process 400 of FIG. 4 begins with an initiation of the graph partition manager 110. The example process 400 may be executed to

process an iteration of a graph having graph data distributed to the plurality of partitions 122. At block 410, the partition updater 230 instructs the local graph updaters 126 of the partitions 122 to perform next graph processing iterations of graph data in the graph partitions 122. At block 420, the replicate aggregator 220 determines whether node data includes node data for replicated nodes in the first graph processing iterations. If no replicate nodes are in the updated graph data, then control returns to block 410 and the next iteration may be executed. If replicate nodes are in the updated graph data, the replicate aggregator 220, at block 430, aggregates the node data (e.g., reduces beliefs of the replicate nodes) across the graph partitions that include the replicate nodes.

[0037] At block 440 of FIG. 4, the partition updater 230 instructs the local graph updaters 126 to update node data of the replicate nodes with the aggregate node data in the graph partitions 122 for second graph processing iterations (e.g., subsequent execution of the process 400). After block 440, the example process 400 ends. In some examples, after block 440, control may iteratively return to block 410 until the graph manager 110 determines that the local graph data 124 for the graph has converged.

[0038] As mentioned above, the example processes of FIGS. 3 and 4 may be implemented using coded instructions (e.g., computer and/or machine readable instructions) stored on a tangible machine readable storage medium such as a hard disk drive, a flash memory, a read-only memory (ROM), a compact disk (CD), a digital versatile disk (DVD), a cache, a random-access memory (RAM) and/or any other storage device or storage disk in which information is stored for any duration (e.g., for extended time periods, permanently, for brief instances, for temporarily buffering, and/or for caching of the information). As used herein, the term tangible machine readable storage medium is expressly defined to include any type of machine readable storage device and/or storage disk and to exclude propagating signals and to exclude transmission media. As used herein, "computer readable storage medium" and "machine readable storage medium" are used interchangeably. Additionally or alternatively, the example processes of FIGS. 3 and 4 may be implemented using coded instructions (e.g., computer and/or machine readable instructions)

stored on a non-transitory computer and/or machine readable medium such as a hard disk drive, a flash memory, a read-only memory, a compact disk, a digital versatile disk, a cache, a random-access memory and/or any other storage device or storage disk in which information is stored for any duration (e.g., for extended time periods, permanently, for brief instances, for temporarily buffering, and/or for caching of the information). As used herein, the term non-transitory machine readable medium is expressly defined to include any type of machine readable storage device and/or storage disk and to exclude propagating signals and to exclude transmission media.

[0039] As used herein, when the phrase "at least" is used as the transition term in a preamble of a claim, it is open-ended in the same manner as the term "comprising" is open ended. As used herein the term “a” or “an” may mean “at least one,” and therefore, “a” or “an” do not necessarily limit a particular element to a single element when used to describe the element. As used herein, when the term “or” is used in a series, it is not, unless otherwise indicated, considered an “exclusive or.”

[0040] FIG. 5 is a block diagram of an example processor platform 500 capable of executing the instructions of FIGS. 3 and 4 to implement the graph partition manager 110 of FIG. 2. The example processor platform 500 may be or may be included in any type of apparatus, such as a server, a personal computer, or any other type of computing device.

[0041] The processor platform 500 of the illustrated example of FIG. 5 includes a processor 512. The processor 512 of the illustrated example is hardware. For example, the processor 512 can be implemented by at least one integrated circuit, logic circuit, microprocessor or controller from any desired family or manufacturer.

[0042] The processor 512 of the illustrated example includes a local memory 513 (e.g., a cache). The processor 512 of the illustrated example is in communication with a main memory including a volatile memory 514 and a non-volatile memory 516 via a bus 518. The volatile memory 514 may be implemented by Synchronous Dynamic Random Access Memory (SDRAM), Dynamic Random Access Memory (DRAM), RAMBUS Dynamic Random

Access Memory (RDRAM) and/or any other type of random access memory device. The non-volatile memory 516 may be implemented by flash memory and/or any other desired type of memory device. Access to the main memory 514, 516 is controlled by a memory controller.

[0043] The processor platform 500 of the illustrated example also includes an interface circuit 520. The interface circuit 520 may be implemented by any type of interface standard, such as an Ethernet interface, a universal serial bus (USB), and/or a peripheral component interconnect (PCI) express interface.

[0044] In the illustrated example, at least one input device 522 is connected to the interface circuit 520. The input device(s) 522 permit(s) a user to enter data and commands into the processor 512. The input device(s) can be implemented by, for example, an audio sensor, a microphone, a camera (still or video), a keyboard, a button, a mouse, a touchscreen, a track-pad, a trackball, isopoint and/or a voice recognition system.

[0045] At least one output device 524 is also connected to the interface circuit 520 of the illustrated example. The output device(s) 524 can be implemented, for example, by display devices (e.g., a light emitting diode (LED), an organic light emitting diode (OLED), a liquid crystal display, a cathode ray tube display (CRT), a touchscreen, a tactile output device, a light emitting diode (LED), a printer and/or speakers). The interface circuit 520 of the illustrated example, thus, may include a graphics driver card, a graphics driver chip or a graphics driver processor.

[0046] The interface circuit 520 of the illustrated example also includes a communication device such as a transmitter, a receiver, a transceiver, a modem and/or network interface card to facilitate exchange of data with external machines (e.g., computing devices of any kind) via a network 526 (e.g., an Ethernet connection, a digital subscriber line (DSL), a telephone line, coaxial cable, a cellular telephone system, etc.).

[0047] The processor platform 500 of the illustrated example also includes at least one mass storage device 528 for storing executable instructions (e.g., software) and/or data. Examples of such mass storage

device(s) 528 include floppy disk drives, hard drive disks, compact disk drives, Blu-ray disk drives, RAID systems, and digital versatile disk (DVD) drives.

[0048] The coded instructions 532 of FIGS. 3 and 4 may be stored in the mass storage device 528, in the local memory 513 in the volatile memory 514, in the non-volatile memory 516, and/or on a removable tangible machine readable storage medium such as a CD or DVD.

[0049] From the foregoing, it will be appreciated that the above disclosed methods, apparatus and articles of manufacture enable graph processing of a graph with graph data distributed across a plurality of partitions of a memory (e.g., a shared memory of a system). In examples herein, local graph data is processed within each partition and node data corresponding to replicate nodes is aggregated and re-executed in each iteration of the graph processing. Furthermore, in examples herein, different update strategies (e.g., synchronous, asynchronous, hybrid synchronous/asynchronous) may be implement to process the local graph data for each iteration.

[0050] Although certain example methods, apparatus and articles of manufacture have been disclosed herein, the scope of coverage of this patent is not limited thereto. On the contrary, this patent covers all methods, apparatus and articles of manufacture fairly falling within the scope of the claims of this patent.

CLAIMS

What is claimed is:

1. A method comprising:

receiving node data for replicate nodes of a graph, the node data generated from first processing iterations of graph data distributed in a plurality of graph partitions and received from respective local graph updaters of the plurality of graph partitions of a shared memory;

aggregating beliefs of processed replicate nodes; and

instructing the local graph updaters of the graph partitions to update beliefs of the replicate nodes for second processing iterations of the graph partitions.

2. The method as defined in claim 1, wherein each of the graph partitions comprise edge data for different edges of the graph, the edge data for each graph partition being processed with node data for nodes of the edges in each partition to generate the node data.

3. The method as defined in claim 1, further comprising:

selecting a graph processing types to process the first processing iterations and the second processing iterations based on characteristics of the graph, the graph processing types comprising at least one of synchronous graph processing, asynchronous graph processing, or a hybrid synchronous/asynchronous graph processing.

4. The method as defined in claim 1, further comprising:

instructing the local graph updaters to process the first processing iterations using a first graph processing type; and

instructing the local graph updaters to process the second processing iterations using a second graph processing type that is different from the first processing type.

5. The method as defined in claim 4, wherein the first processing type comprises synchronous graph processing and the second processing type comprises asynchronous graph processing.

6. The method as defined in claim 4, wherein at least one of the first processing type or the second processing type comprises hybrid graph processing.

7. A non-transitory machine readable storage medium comprising instructions that, when executed, cause a machine to at least:

instruct respective local graph updaters of graph partitions to perform first graph processing iteration of graph data in the graph partitions, each of the graph partitions comprising edge data of different edges of a graph and node data corresponding to nodes connected by the edges of the edge data;

determine whether node data comprises node data for replicate nodes in the first graph processing iterations, the replicate nodes comprising nodes with

updated node data from a plurality of the first processing iterations of at least two of the graph partitions; and

when the node data is processed for replicate nodes in the first graph processing iterations:

aggregate the node data for the replicate nodes across the graph partitions that include the replicate nodes, and

instruct the local graph updaters to update node data with the aggregated node data for the replicate nodes in the respective graph partitions for second graph processing iterations.

8. The non-transitory machine readable storage medium of claim 7, wherein when the node data the instructions, when executed, further cause the machine to:

partition the graph into the graph partitions such that edge data for the edges of the graph is distributed among the graph partitions, such that edge data for each edge of the graph is in a single graph partition.

9. The non-transitory machine readable storage medium of claim 7, wherein when the node data the instructions, when executed, further cause the machine to:

instruct the local graph updaters to perform the second graph processing iterations using a different graph processing type than the first graph processing iterations.

10. The non-transitory machine readable storage medium of claim 7, wherein each of the graph partitions comprise edge data for different edges of the graph, the edge data for each graph partition being processed with node data for nodes of the edges in each partition to generate the node data.

11. An apparatus comprising:

an update receiver to receive graph processing data from first graph processing iterations of a graph, the graph distributed across a plurality of graph partitions in a shared memory, the first graph processing iterations executed in parallel by local graph updaters of each of the graph partitions;

a replicate aggregator to aggregate node data for replicate nodes of the graph processing data that is processed in the first processing iterations; and

a partition updater to instruct local graph updaters of the graph partitions to update the node data for the nodes in the graph partitions based on the aggregated node data.

12. The apparatus of claim 11, wherein the partition updater is further to instruct the local graph updaters of the graph partitions to perform second processing iterations using a different graph processing type than the first processing iterations.

13. The apparatus of claim 11, wherein each of the graph partitions comprise edge data for different edges of the graph, the edge data for each graph

partition being processed with node data for nodes of the edges in each partition to generate the node data.

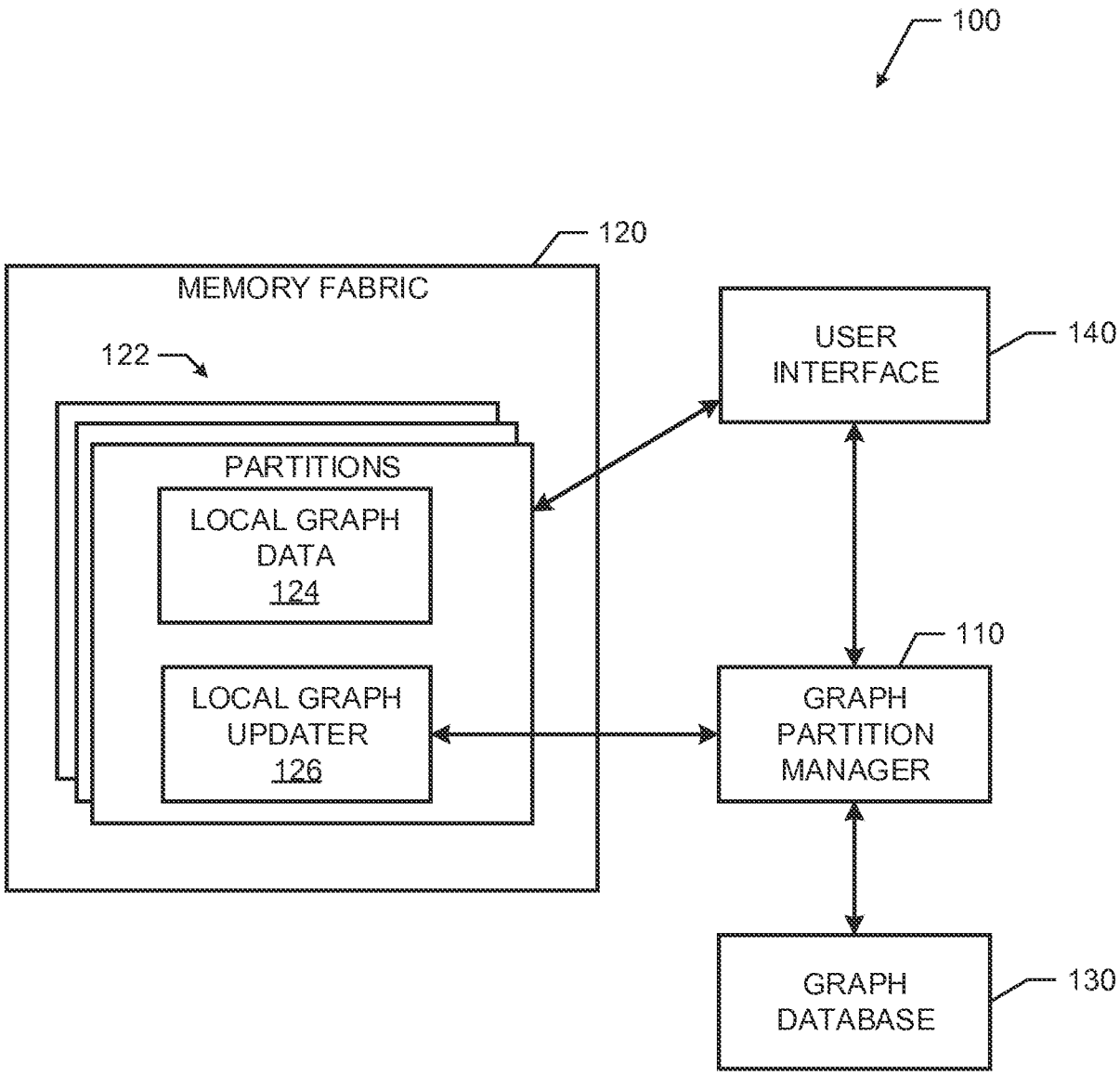

100
120
MEMORY FABRIC
122
PARTITIONS
LOCAL GRAPH DATA 124
LOCAL GRAPH UPDATER 126
USER INTERFACE
140
110
GRAPH PARTITION MANAGER
GRAPH DATABASE
130

FIG. 1

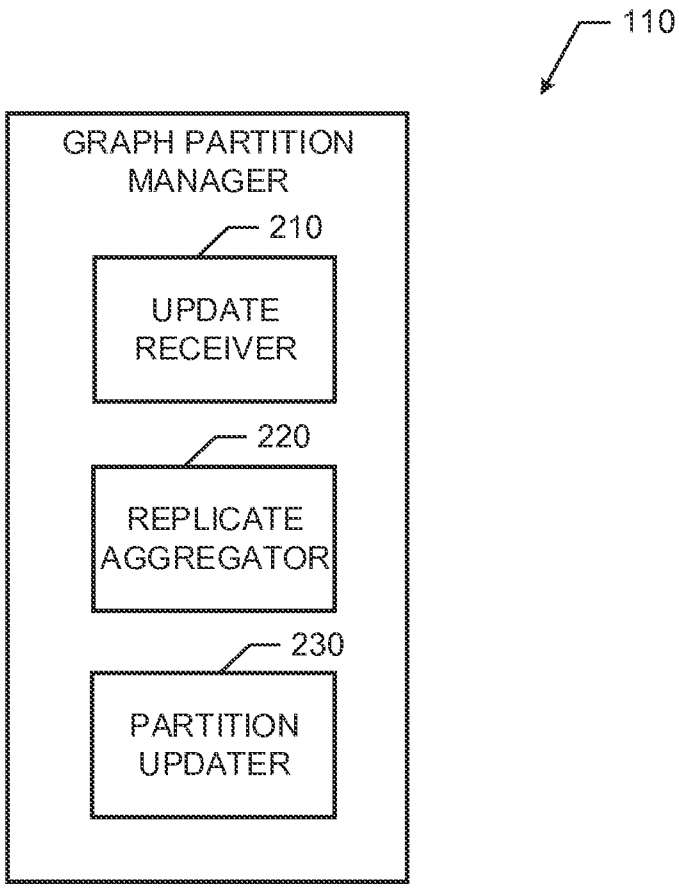

110
GRAPH PARTITION MANAGER
210
UPDATE RECEIVER
220
REPLICATE AGGREGATOR
230
PARTITION UPDATER

FIG. 2

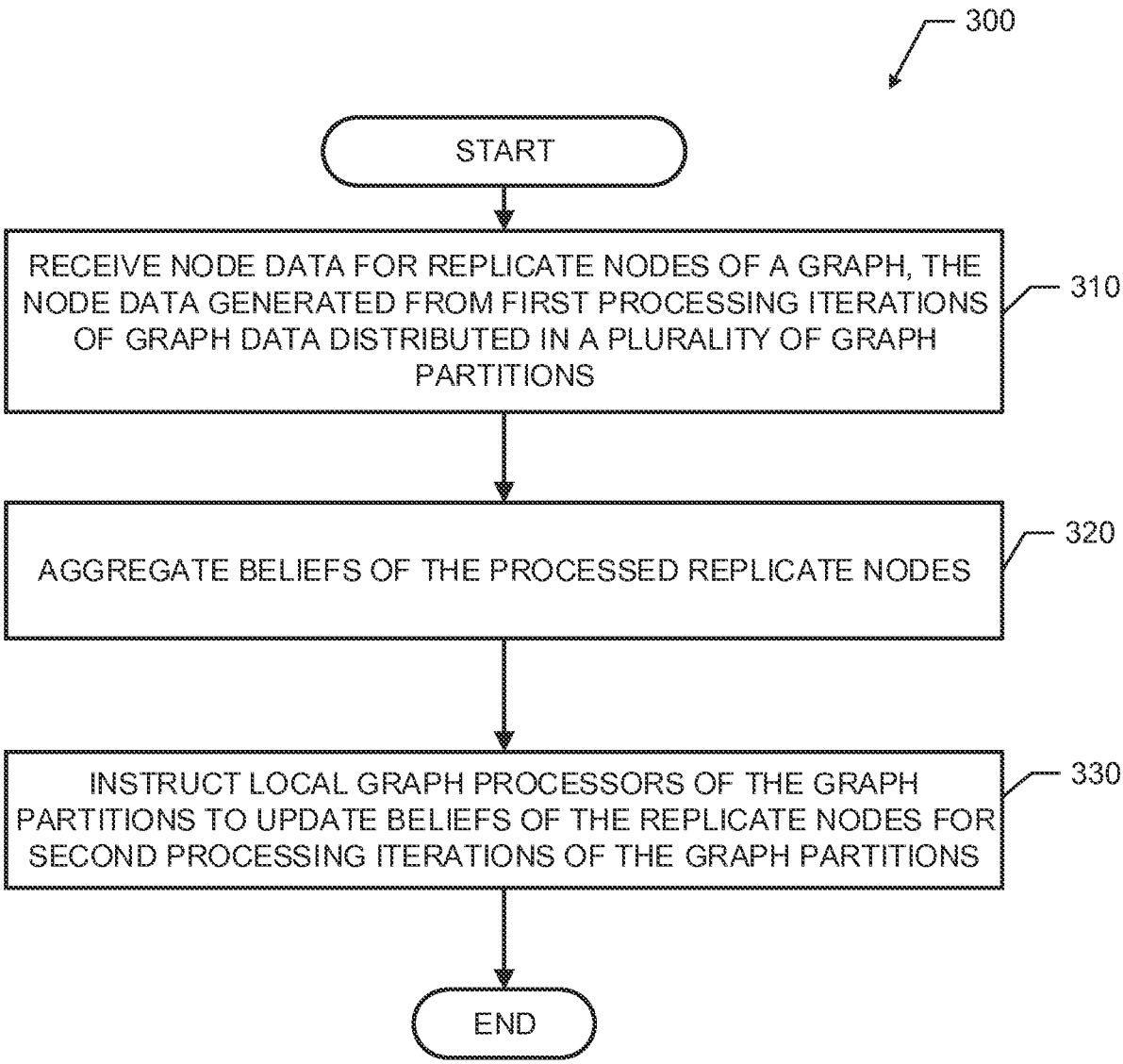

300
START
RECEIVE NODE DATA FOR REPLICATE NODES OF A GRAPH, THE NODE DATA GENERATED FROM FIRST PROCESSING ITERATIONS OF GRAPH DATA DISTRIBUTED IN A PLURALITY OF GRAPH PARTITIONS
310
AGGREGATE BELIEFS OF THE PROCESSED REPLICATE NODES
320
INSTRUCT LOCAL GRAPH PROCESSORS OF THE GRAPH PARTITIONS TO UPDATE BELIEFS OF THE REPLICATE NODES FOR SECOND PROCESSING ITERATIONS OF THE GRAPH PARTITIONS
330
END

FIG. 3

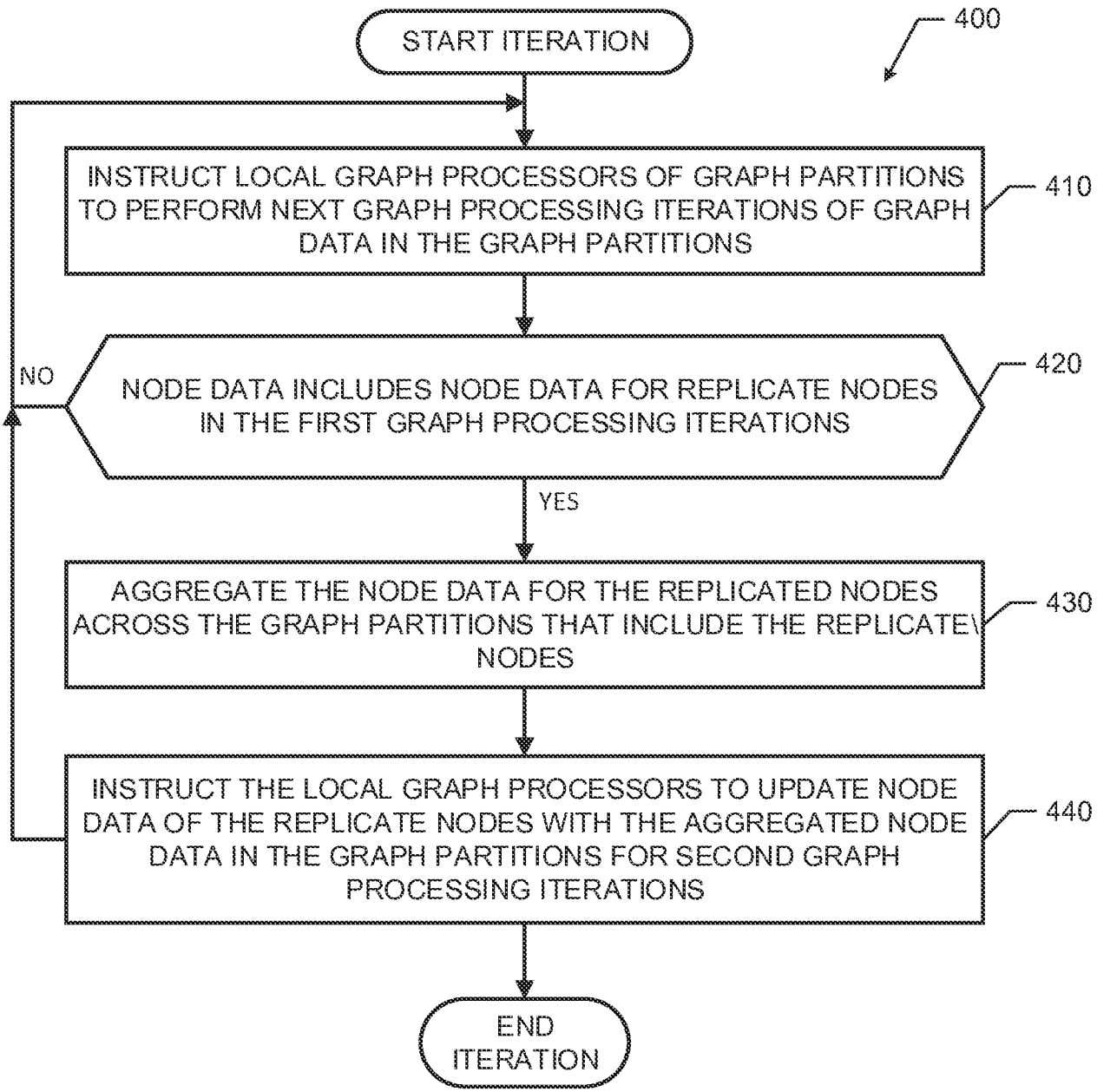

400
START ITERATION
INSTRUCT LOCAL GRAPH PROCESSORS OF GRAPH PARTITIONS TO PERFORM NEXT GRAPH PROCESSING ITERATIONS OF GRAPH DATA IN THE GRAPH PARTITIONS
410
NODE DATA INCLUDES NODE DATA FOR REPLICATE NODES IN THE FIRST GRAPH PROCESSING ITERATIONS
420
NO
YES
AGGREGATE THE NODE DATA FOR THE REPLICATED NODES ACROSS THE GRAPH PARTITIONS THAT INCLUDE THE REPLICATE\ NODES
430
INSTRUCT THE LOCAL GRAPH PROCESSORS TO UPDATE NODE DATA OF THE REPLICATE NODES WITH THE AGGREGATED NODE DATA IN THE GRAPH PARTITIONS FOR SECOND GRAPH PROCESSING ITERATIONS
440
END ITERATION

FIG. 4

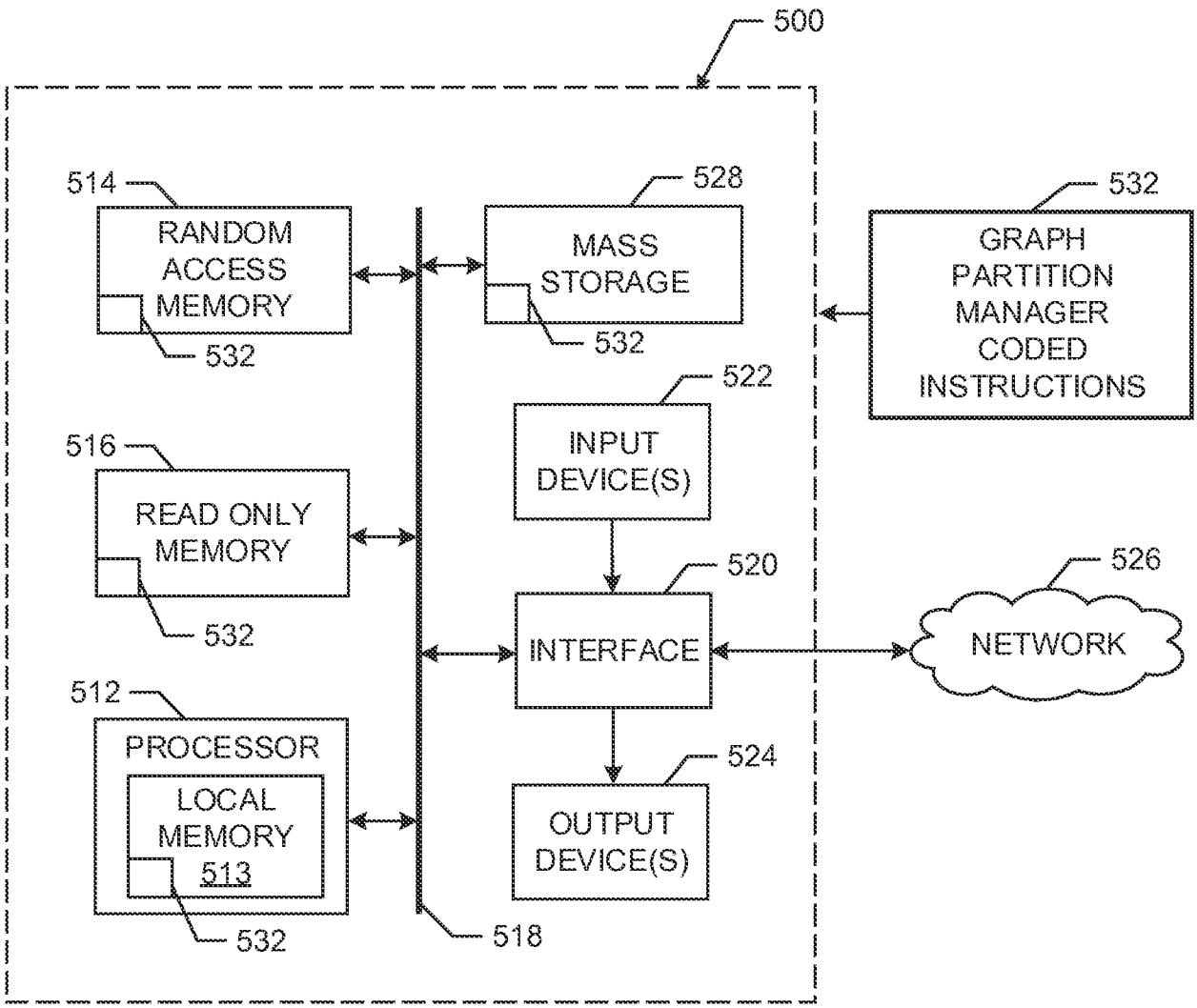

500
514
RANDOM ACCESS MEMORY
532
528
MASS STORAGE
532
532
GRAPH PARTITION MANAGER CODED INSTRUCTIONS
522
INPUT DEVICE(S)
516
READ ONLY MEMORY
532
520
526
NETWORK
INTERFACE
512
PROCESSOR
LOCAL MEMORY
513
532
524
OUTPUT DEVICE(S)
518

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/061798

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(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 17/30; G06Q 30/02; G06F 17/00; H04L 9/00; G06N 5/02

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: replicate node, graph, partition, node data, local graph updater, beliefs, update, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014-0122422 A1 (HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.) 01 May 2014 See paragraphs [0021]-[0022], [0028]-[0029], and [0038]; and figure 1.	1-13
A	US 2008-0155656 A1 (JOHN MARK AGOSTA et al.) 26 June 2008 See paragraphs [0023]-[0027] and figure 2.	1-13
A	US 7,216,123 B2 (SEPANDAR D. KAMVAR et al.) 08 May 2007 See column 4, lines 16-24; column 7, line 56 - column 8, line 31; and figures 1 and 6.	1-13
A	US 2013-0013567 A1 (CORNELIU MIHAIL CONSTANTINESCU et al.) 10 January 2013 See paragraphs [0019], [0032]-[0035], and [0048]-[0050]; and figures 5 and 7.	1-13
A	US 2014-0129320 A1 (TONY JEBARA et al.) 08 May 2014 See paragraphs [0221]-[0226].	1-13

☐ 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
19 August 2016 (19.08.2016)	**19 August 2016 (19.08.2016)**

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. +82-42-481-8578

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/061798

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014-0122422 A1	01/05/2014	US 9317518 B2	19/04/2016
US 2008-0155656 A1	26/06/2008	US 7921453 B2	05/04/2011
US 7216123 B2	08/05/2007	CN 101176092 A	07/05/2008
		EP 1625471 A2	15/02/2006
		EP 1625471 A4	19/03/2008
		US 2005-0027685 A1	03/02/2005
		US 2005-0033742 A1	10/02/2005
		US 7028029 B2	11/04/2006
		WO 2004-088475 A2	14/10/2004
		WO 2004-088475 A3	15/11/2007
US 2013-0013567 A1	10/01/2013	US 8538929 B2	17/09/2013
US 2014-0129320 A1	08/05/2014	WO 2012-138855 A1	11/10/2012