

(51) International Patent Classification:

G06F 9/315 (2006.01)

(21) International Application Number:

PCT/US2014/039162

(22) International Filing Date:

22 May 2014 (22.05.2014)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

61/826,380

22 May 2013 (22.05.2013)

US

(71) Applicant: NEURALA, INC. [US/US]; The Photonics Center, 8 St. Mary's Street, Suite 613, Boston, Massachusetts 02215 (US).

(72) Inventors: GORSHECHNIKOV, Anatoly; 15 Jefferson Street, Newton, Massachusetts 02458 (US). VERSACE, Massimiliano; 846 East 3rd Street, Boston, Massachusetts 02127 (US). VERSACE, Heather Ames; 846 East 3rd Street, Boston, Massachusetts 02127 (US). LIVITZ, Genady; 61 Brookside Avenue, Belmont, Massachusetts 02478 (US).

(74) Agents: TEJA, Joseph et al.; Cooley LLP, 1299 Pennsylvania Ave., Suite 700, Washington, District of Columbia 20004 (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, LT, 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, 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).

[Continued on next page]

(54) Title: METHODS AND APPARATUS FOR ITERATIVE NONSPECIFIC DISTRIBUTED RUNTIME ARCHITECTURE AND ITS APPLICATION TO CLOUD INTELLIGENCE

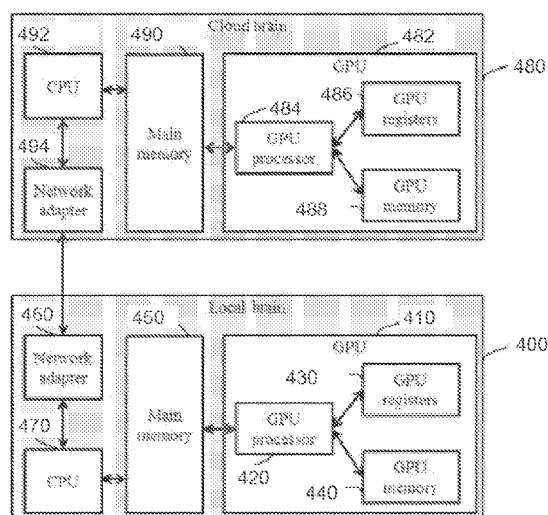


Figure 4

(57) Abstract: The system and methods disclosed herein include a runtime architecture that takes a non-specific set of systems of differential equations, distributes them across the network, and iteratively integrates them through time with a possibility to output the results on every iteration. Embodiments of the disclosed system may be used for neural computation or any other suitable application. Embodiments can be used as a standalone engine or as part of another computational system for massively parallel numerical integration of a data-driven dynamical system.



Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

**Methods and Apparatus for Iterative Nonspecific Distributed Runtime Architecture
and its Application to Cloud Intelligence**

CROSS-REFERENCE TO RELATED PATENT APPLICATION

[0001] This application claims priority, under 35 U.S.C. § 119(e), from U.S. Application No. 61/826,380, filed May 22, 2013, and entitled “Iterative Nonspecific Distributed Runtime Architecture,” which application is hereby incorporated herein by reference in its entirety.

GOVERNMENT SUPPORT

[0002] This invention was made with government support under Contract No. FA8750-12-C-0123 awarded by the U.S. Air Force. The government has certain rights in the invention.

BACKGROUND

[0003] Current computer hardware central processor architectures and often even graphic processor enabled workstations are often unable to achieve the highly parallel requirements needed to perform computationally expensive massive data-driven algorithms, especially those that are based on dynamic systems approach.

SUMMARY

[0004] The system and methods described herein are based on a network of processing nodes that can be individual heterogeneous workstations or a cluster of homogeneous computer boards. An example of the system enables a runtime architecture based on this hardware to load a nonspecific set of systems of differential equations, distribute them across network nodes according to hardware capabilities of each node, and iteratively integrate them through time with a possibility to output the results on every iteration. An exemplary system may be used for a variety of applications, including but not limited to neural computations. For instance, an exemplary system can be used as a standalone computational engine engaged on a hardware cluster directly or as a part of other computational software that at some point uses massively parallel numerical integration of data driven dynamical systems on a network of remote computational hardware. Furthermore, a resulting computational engine can serve as a remote (e.g., cloud) back end that can engage remote hardware resources to support local autonomous hardware devices such as robots, automatic household devices, mobile navigators, and other devices.

[0005] In some implementations, a method for processing sensory data in parallel may comprise receiving, at a master node, a request to process incoming sensor data, receiving, from the master node, a selection of a set of slave nodes, the set of slave nodes selected from a set of nodes capable of processing the incoming sensor data, and establishing a connection between the master node and the set of slave nodes. For at least one slave node in the set of slave nodes, the implementation may further include determining a dynamic system model for a portion of the incoming sensor data received by the at least one slave node, distributing, from the master node to the at least one slave node, a representation of the dynamic system model and a representation of corresponding graph connections between slave nodes in the set of slave nodes, instantiating at least one local data structure for processing the incoming sensor data based at least in part upon the representation of the dynamic system model distributed earlier, connecting the at least one slave node to other slave nodes in the set of slave nodes according to the representation of the graph connections distributed earlier, receiving a signal from the master node to process the incoming sensor data; and iteratively processing the incoming sensor data using the at least one local data structure instantiated previously.

[0006] In some implementations, the process may also comprise selecting the set of slave nodes automatically, or selecting the set of slave nodes according to user input. In some implementations, distributing the representation of the dynamic system model may further comprise assigning at least one system of equations corresponding to the dynamical system model to the at least one slave node based on at least one of computational power of the at least one slave node, computational complexity of the at least one system of equations, and communication between parts of a dynamical system represented by the dynamical system model. Distributing the representation of the dynamic system model may also comprise assigning, by the at least one slave node, a set of computational threads for processing at least a portion of the incoming sensor data.

[0007] In some implementations, connecting the at least one slave node to other slave nodes may further comprise determining an operating mode of the master node, and connecting the master node to the at least one slave node based on the operating mode of the master node. In some implementations, this may further comprise connecting the master node to the at least one slave node via a network connection if the master node is configured to operate in a networked mode, and compiling the slave node into the master node if the master node is configured to operate in a standalone mode. In some implementations the master node is

configured to operate in a networked mode or a standalone mode based on available resources, and is configured to operate in the networked mode if the master node is able to connect to a network and if there is at least one slave node connected to the network.

[0008] In some implementations, processing the data may further comprise storing results of the iterative processing in a data queue in the at least one slave node for communication with the other slave nodes in the set of slave nodes, and storing results of the iterative processing in a storage medium for off-line analysis.

[0009] An exemplary system for processing and forwarding slave node data may comprise a first slave node operatively coupled to an outgoing axonal queue associated with a system of equations processed on the first slave node, a shadow axonal queue corresponding to respective outgoing axonal queues of other slave nodes configured to provide input data to the first slave node, a set of synaptic queues corresponding to the system of equations processed on the first slave node, the set of synaptic queues configured to preprocess the input data received from the shadow axonal queue, and a processing device, operably coupled to the set of synaptic queues, to apply the system of equations to the input data preprocessed by the synaptic queue so as to generate at least one output data value. The system may further comprise a local outgoing axonal queue, operatively coupled to the processing device, to receive the at least one output data value from the processing device and to provide the at least one output data value to a second shadow axonal queue in at least one of the first slave node or a second slave node. The shadow axonal queue may be configured to receive at least one data value from a corresponding outgoing axonal queue operatively coupled to a slave node.

[0010] The outgoing axonal queues and shadow axonal queues can be implemented with a standard template library (STL) class called multimap that holds key-value pairs. The key in the key-value pair may be event production time and the value in the key-value pair can include for example receiver address and output value. STL multimap does automatic sorting by key, so this queue may have output events sorted by their production times. The synaptic queue can also be implemented with a standard template library (STL) class called multimap that holds key-value pairs, except that the receiver address is no longer needed, the timing represents delivery time rather than production time, and the value in the key-value pair is adjusted according to the parameters of the synapse.

[0011] An exemplary system for aggregating and processing sensor data from a robot comprising a local processor that implements a local brain may comprise at least one slave node, located on a computer network, to implement a cloud brain for receiving and processing the sensor data from the robot, the first processor including a main memory module to store the sensor data from the robot, and a communications device, operably coupled to the main memory module, to transmit at least a portion of the sensor data to the local brain implemented in the local processor of the robot and to receive, from the local brain, at least one data value based at least in part on the at least a portion of the sensor data and on a current state of the local brain.

[0012] The main memory module may store a representation of at least one of an outgoing axonal queue associated with a system of equations processed on a first slave node, a shadow axonal queue corresponding to respective outgoing axonal queue of a second slave node configured to provide input data to the first slave node, a synaptic queues corresponding to the system of equations processed on the first slave node, the set of synaptic queues configured to preprocess the input data received from the shadow axonal queue, and a data structure for processing the sensor data from the robot. The communications device is operably coupled to another local brain implemented in another robot and configured to transmit the at least one data value to the other local brain. The communications device may be further configured to transmit data from a dedicated memory block in the main memory module to the other local brain and to receive, from the other local brain, at least one other data value based at least in part on the data from the dedicated memory block.

[0013] It should be appreciated that all combinations of the foregoing concepts and additional concepts discussed in greater detail below (provided such concepts are not mutually inconsistent) are contemplated as being part of the inventive subject matter disclosed herein. In particular, all combinations of claimed subject matter appearing at the end of this disclosure are contemplated as being part of the inventive subject matter disclosed herein. It should also be appreciated that terminology explicitly employed herein that also may appear in any disclosure incorporated by reference should be accorded a meaning most consistent with the particular concepts disclosed herein.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The skilled artisan will understand that the drawings primarily are for illustrative purposes and are not intended to limit the scope of the inventive subject matter described herein. The drawings are not necessarily to scale; in some instances, various aspects of the inventive subject matter disclosed herein may be shown exaggerated or enlarged in the drawings to facilitate an understanding of different features. In the drawings, like reference characters generally refer to like features (e.g., functionally similar and/or structurally similar elements).

[0015] Figures 1A and 1B are diagrams of a networked system and a standalone system each implementing Iterative Nonspecific Distributed Runtime Architectures, respectively, according to illustrative implementations.

[0016] Figure 2 is a flow chart of the control flow through the systems of Figures 1A and 1B, according to one illustrative implementation.

[0017] Figures 3A and 3B illustrate data flow on the slave hardware nodes of system of Figure 1A and standalone system of Figure 1B, according to illustrative implementations. Figure 3A shows data flow during system and model setup. Figure 3B shows the data flow during the computation.

[0018] Figure 3C is a flow chart of a method for event processing, according to one illustrative implementation.

[0019] Figure 3D is a flow chart of event communications between hardware nodes on the network, according to one illustrative implementation.

[0020] Figure 4 is a diagram of an illustrative implementation of a cloud server (brain) and a local robot, according to one illustrative implementation.

[0021] Figure 5 is a diagram of an illustrative implementation of a cloud server (brain) and a plurality of local robots, according to one illustrative implementation.

[0022] Figure 6 is a diagram of an illustrative implementation of a cloud server (brain) with separation of processors and memory dedicated to a plurality of robots, according to one illustrative implementation.

[0023] Figure 7 is a diagram of an illustrative implementation of transferring knowledge to a first robot from a portion of a cloud server (brain) dedicated to a second robot, according to one illustrative implementation.

[0024] Figure 8 is a diagram of an illustrative implementation of sharing knowledge on a cloud server (brain) without copying network structure.

DETAILED DESCRIPTION

[0025] The system and methods described herein relate to a hardware computational engine consisting of two or more computational nodes such as workstations or cluster components connected through a wired or wireless network. In some implementations, the engine underlies and subserves neural simulations and/or other simulations. The system employs network communication management, thread management, event based communication, over the network data exchange, and data structures used to facilitate the functionality herein.

[0026] In some implementations, the system includes a computer readable memory onto which processor executable instructions are written. The computer readable memory can store a plurality of libraries. For example, the system on a slave computational node may include: a pthreads library that is used to manage multithreaded control flow; a standard template library (STL) that may be used for structuring the data; and a CUDA or OpenCL library to control execution on graphic processors. In some implementations, the graphics processor unit (GPU) libraries may not be required. The system on the master node may include additional libraries for graphic user interface and data visualization and/or other additional libraries and/or like information.

[0027] Figure 1A is a system diagram of a networked implementation 100 of an Iterative Nonspecific Distributed Runtime Architecture (INDRA). The networked implementation 100 includes a single master node 102 receiving commands through an application programming interface (API) and controlling an arbitrary number of slave nodes 106a-b. Figure 1B shows a standalone version 108 of an INDRA receiving commands through the same API (master part) 110 and executing the commands through a simplified version of a slave node 112 created and connected to locally by the master node (e.g., the slave node 112 may be directly merged into the master node).

[0028] As illustrated in Figure 1A, the structure of the system 100 comprises a single master node 102 and an arbitrary number of slave nodes 106a, 106b, ... (collectively, slave

nodes 106). The master node 102 is a computing hardware component where a user or an external application can instantiate the engine control structures (those engine control structures including but not limited to a parser that converts user provided description of a dynamical system in machine readable data structures, a network communication manager that establishes connections with slaves, and a command and synchronization module that passes commands (e.g. run, stop, reset) and synchronization signals from master to the slaves). It can be implemented as a user controlled workstation, portable electronic device, or as an autonomous electronic device such as robot. The engine control structures may initiate engine execution on remote computer systems serving as slave nodes 106 to the engine, gather the information about available hardware resources for each slave (CPU cores, GPUs, memory), and distribute the dynamical systems, which may take the form of neuronal populations (or systems of differential or algebraic equations) and corresponding synapses (event-based dependencies between these equations) across the slave nodes 106.

[0029] As used herein, a dynamical system is a set of systems of differential and or algebraic equations that describe some natural process through time. A neural model is a subset of a dynamical system where equations represent neuronal dynamics. A neuronal population is a part of a neural model that comprises at least one neuronal element and for all neuronal elements the governing system of equations is identical up to a parameter choice. A synapse is a dependency between two neuronal elements that can be in the same or different populations such that it connects a source neuronal element with the target neuronal element and passes output events from the source to the target according to some rule defined by the user.

[0030] The above tasks correspond to the System Setup and Model Initialization phases of the example flow chart in Figure 2 and example data exchange diagram in Figure 3A. During these stages the slave nodes 106 receive network communication from the master node control structures and instantiate local data structures, allocate hardware memory, query processing devices, and otherwise prepare to perform the computation on the local hardware.

[0031] The dynamical system may comprise a set of populations, where each population is a collection of identical systems of equations, and where each element in a population is a computational element.

[0032] Each computational element may have output variables that affect the rest of the system. For example, in the case of a neural model, the output of a neuron may be those

output variables. A computational element also may have several internal variables that are used to compute output variables, but are not exposed to the rest of the system, not even to other elements of the same population, typically. Each of these variables is represented as a CPU or GPU memory chunk. An exemplary difference between output variables and internal variables is their access: output variables may be made available to other computational elements either directly or through generation of output events, while internal variables may be hidden from other computational elements.

[0033] An example implementation of a computational element data structure may be a TVariable class. TVariable class may be based on a TBasicBuffer template class, which may be part of a libSMARTIO (Sensory-Motor Asynchronous Real Time Input Output) library that provides the basic structure and connectivity between populations in the dynamical system. Other portions of TBasicBuffer may allow the system to acquire sensory input from the environment (TInputBuffer) and deliver action output to the agent (TOutputBuffer), both in real time. In some implementations, TBasicBuffer objects may have a name, a number of elements, an index of a non-CPU device, and may facilitate saving variable values, accessing values for various computations, and/or may have other attributes or methods. In some implementations, a TVariable data structure may include connections to an input source, a number of equations associated with the data structure, an event connection that may look for events on a local receiver and process them, and/or may include other attributes and/or methods.

[0034] TVariable may also provide memory allocation and methods to maintain a population (array) of variables governed by a system of equations. For example, if a dynamical system has a system of equations with variables x_1 , x_2 , and x_3 , an example implementation may use a TVariable-based class. If there are 100 processes in the dynamical system, each of which is governed by the same system of equations with these three variables, then TVariable-based class may contain 100 elements with 3-component system each. Classes derived from TVariable class may perform initialization procedures, assignment of pointers to equations and numerical methods, and implement different computational backends based on available hardware such as GPU or CPU. They may also provide different implementations of the above methods depending on whether the system is used in a standalone or networked mode.

[0035] Output variables are usually accessed by any element in the system during every time step. The value of the output variable that is accessed by other elements of the system corresponds to the value computed on the previous, not the current, time step. This is realized by dedicating two CPU or GPU memory chunks to output variables—one holds the value computed during the previous time step and is accessible to all computational elements during the current time step, another is not accessible to other elements and is used to accumulate new values for the variable computed during the current time step. In-between time steps these two CPU or GPU memory chunks are switched, so that newly accumulated values serve as accessible input during the next time step, while the old input is replaced with new values of the variable. This switch is implemented by swapping the address pointers to respective CPU or GPU memory chunks. The portion of the time step during which the system computes new values is termed a Compute Semicycle in Figure 2 and the portion of the time step when pointers are switched and new values become available to the rest of the system is termed a Post Semicycle (e.g., see Figure 2).

[0036] Internal variables are computed and used within the same computational element. There is no chance of a race condition in which the value is used before it is computed or after it has already changed on the next time step because within an element the processing is sequential. Therefore, it is possible to compute the new value of internal variable and store it in the same memory location in the memory bank where the old value was read from during the computation.

[0037] During the computation, the master node 102 utilizes network connections with the slave nodes 106 to synchronize the processing across the nodes 102, 106. This corresponds to Compute and Post Semicycles of the example flow chart in Figure 2 and example data flow chart in Figure 3B. In some implementations, the system includes a user interface on the master node that allows direct control of the software operations through computer peripherals such as keyboards, mice, trackballs and other devices. The system can also include an external application 104 that provides information about neural populations. The system may be accessed through API calls by a third party software.

[0038] Slave nodes may have a network manager that establishes point-to-point links between their respective hardware devices for sending and receiving of events. The system may also include a thread manager that controls creation, execution, synchronization and/or destruction of working threads within each slave node to maximize hardware utilization. A

list of these threads is shown in the Figures 2, 3A, and 3B. The threading structure in the depicted example implementation is based on US Patent No. 8,648,867, which is hereby incorporated herein by reference in its entirety. An example threading structure can include a disk thread that controls the disk output of a node (utilizing a local hard drive of a workstation or a shared disk space of a cluster), opens and closes the files, and does the output for the state variables that are marked for output by the user. Another example thread is communication thread that controls the exchange of output events between neuronal populations or systems of equations within the node and across the network. Other example threads may include non-central processing unit (CPU) computational threads which may be used for computation of neuronal populations on suitable processing devices (e.g., GPUs). In some embodiments, one or more Non-CPU Computational threads are created for each utilized device on the node. Further examples may include CPU Computational threads, which may be used for computation of neuronal populations or systems of equations on available cores of the central processing unit of the slave node. In some implementations, the number of CPU computational threads may be determined by taking the number of cores on the node, subtracting the number of all other threads including the main thread and use the result as the number of CPU Computational threads. In some implementations, for small populations or systems of equations having only one CPU Computational thread might be more efficient.

[0039] Figure 1B illustrates a standalone version 110 in which the control structures of the master node and the computational structures of the slave node are hosted together in (or merged together into) a single software unit that is executed on a standalone workstation or other computer hardware. In such implementations of the system, communications between the systems of equations may happen on the same node. In this implementation of the system a user or external application may initiate the simulation and thread manager directly, and the network management and across-network synchronization may be removed from the system. Everything under the thread manager may remain identical to a slave node 106 in the networked version 100 shown in Figure 1A. In some implementations, an INDRA system may allow switching between instantiating a networked implementation 100 and a standalone version 110 depending on available resources at the moment of instantiation.

[0040] Figure 2 illustrates the control flow through a standalone version of the system (e.g., as shown in Figure 1A). An external application may initiate the engine by starting a simulation and/or thread manager (202). The thread manager may fork and initialize a

requested number of computational threads plus one disk I/O thread (204) and one communication thread (206). During thread initialization, the thread manager may pass network addresses, number of CPU cores (208) and GPU devices (210) to the communication and/or computational threads on the node as illustrated in Figure 3A. These threads may become initialized and gather on the initialization barrier 212 waiting for the simulation manager to prepare the model and simulation setup. The physical location of the simulation manager may differ between the networked implementation and the standalone implementation. While in the standalone version it may reside on the same hardware as the thread manager; in the networked version the simulation manager may reside on the master node while thread managers may reside only on the slave nodes.

[0041] Model and simulation setup on the system may be accomplished through interaction between the simulation manager with an external application and/or user (214). When the parameterization of the model is substantially complete (216), the user may notify the simulation and thread manager that the model is ready and they may proceed to the initialization barrier. This may allow threads to enter the next stage (e.g., model initializations 218 and 302 in Figures 2 and 3A, respectively). At this stage a model and simulation description may be parsed by the simulation manager (220) on the master node and passed on to various threads on the slave nodes either through network connection if the system is in networked mode or directly if the system is in single node mode. On the slave node, the information is distributed as depicted in Figure 3A.

[0042] Population names are used in disk I/O thread to create corresponding files for the result output (304). List of populations that have outputs is used in communication thread to establish input and output queues on the slave node to communicate between connected populations (306). Equations and relating to the data are communicated to computational threads to instantiate the corresponding data structures (308). On the slave nodes, components of the model may be assigned to different computational threads depending on the hardware configuration of a particular slave.

[0043] After instantiation of the components, the communication queues between these components may be defined (224) as illustrated in Figure 3A (310). In some implementations, this information is passed to computational threads (e.g., model components) and/or to communication threads (e.g., connectivity pattern between components), where the respective data structures are created and initialized 226. After the

initialization, the initial values of output variables might be passed to disk I/O thread for output as illustrated in Figure 3A (312). Next, disk I/O thread, communication thread, and computational threads may then enter the main simulation loop of the system, so they signal their readiness to the simulation manager and wait on the Post barrier for the Run signal from the simulation manager.

[0044] The main simulation loop of the system may be controlled by an external application on the master node through a set of API commands that it can pass to the simulation manager. The commands may include:

- Quit, which may cause the simulation manager to exit the main loop and inform the slave thread managers locally (standalone version) or through network communication (networked version). The thread managers may inform other threads that the engine is exiting, wait for them to clean up and exit, and then exit the engine.
- Reset, which causes the reset of the components and connections in the model to their initial states. In some implementations, there are several versions of Reset depending on whether the user wants to reinitialize activations (current values) of components, connectivity strengths, or both.
- Re-create, which may cause the engine to return to the model and simulation setup stage and re-parse all the data structures. This command may also redistribute the components of the model to different nodes and/or computing hardware.
- Run, which enters the computational loop. This command may have a parameter that defines whether a user wants a single step though computation, a defined number of steps, or a continuous calculation.
- Stop, which may interrupt the currently executing computational loop after finishing the current cycle.

[0045] In some embodiments, iterations through a computational loop may include two phases or semi-cycles. During the Compute Semicycle, the model components may read their inputs from read-only output memory locations of other components, compute their own new values (228) and write them into their own write-only memory locations. During a Post Semicycle 230, the values from write-only locations in the component may be moved to its read-only locations to be available for others on the next Compute Semicycle. In some implementations, to reduce memory copying, the pointers to read-only and write-only memory locations are interchanged and the actual data is not copied.

[0046] In some embodiments the communication between neuronal populations or systems of equations during the computational cycles may proceed as shown in Figure 3B. During the Compute Semicycle output events produced in either CPU 312 or non-CPU 314 computational threads are pushed into queues (316) corresponding to these populations' outgoing event queues 232. At the same time a communication thread might receive events (318) from other nodes as well as from local output event queues and insert them into corresponding input event queues. During the Post Semicycle the communication thread delivers (320) the events from input queues to target populations and sends the output queues (322) to disk I/O thread for the output. At the same time the values of the variables can be communicated from the slaves to the master (324) for visualization.

[0047] Figure 3C provides a further elaboration on an example embodiment of the system in terms of event queues engaged by a single population on a single slave node, while Figure 3D shows the same communication scheme from the network standpoint for a slave node with multiple populations. Each neuronal population or system of equations has one outgoing queue termed a local axonal queue 326, where the events produced by this population 328 are inserted during a first Compute Semicycle 303 after synchronization 301. In some implementations, local (outgoing) axonal queues can be implemented with a standard template library class called multimap that holds key-value pairs. For this queue the key may be event production time and the value can include, for example, receiver address and output value. STL multimap may facilitate automatic sorting by key, so this queue may have all output events sorted by their production time. For each receiver of events produced by this population, there is also a shadow axonal queue 330 located on the node where the receiver resides, which may be implemented in a manner similar to local axonal queues. To save memory, there may be only one shadow axonal queue per sender per node, from which all receivers may receive their respective input events coming from this particular sender. During the first Post Semicycle 303, the events are sent from the local axonal queues 326 to shadow axonal queues 330. During a first Compute Semicycles 305 following the first Post Semicycle 303, the events are collected by each node's shadow axonal queue 332. During the next Post Semicycle 307, the events are delivered from shadow axonal queues to corresponding receivers through input (synaptic) queues 334, which may be implemented in a manner similar to local axonal queues, but which may not require a receiver address, may represent timing via delivery time rather than production time, and may adjust the value according to the parameters of the synapse. As a result of this scheme, each output event can

be delivered to the shadow axonal queues 330 as early as two cycles from its production (i.e., at the end of a second Compute Semicycle 309). The length of each semicycle in real time is determined by the computational load on the system, network load and latency, and may be bound from below by the slowest component. For simple models without much computational load using a standalone system, the semicycle length can be as short as 10 microseconds per a pair of Compute and Post Semicycles.

[0048] In the case of dynamical systems, however, especially those simulating neural populations, there can be axonal delay, and events are often delivered on the millisecond time scale, while the time steps of numerical integration of these equations are significantly shorter for reasons of numerical stability. Thus event delivery often may happen many more than two cycles from their production. The example system presented herein takes advantage of that by bundling the events between nodes in packages and sending these packages over the network only when the first event in the package is due for the delivery. Bundling and sending packages reduces network communication overhead.

[0049] The CPU based computational threads may be organized differently from non-CPU based threads. A component of the model that represents a massive data-parallel population may be passed as a whole to non-CPU thread and to the hardware device it controls to reduce cross-device communication. In some embodiments, the CPU thread population may be split into as many parts as there are CPU threads to balance the CPU load. Thus, initialization and posting for CPU populations may happen as a whole population across threads operation, while computation is done in parts by respective CPUs.

[0050] In some implementations some slave nodes can reside in a cloud environment and other slave nodes can reside on individual robots, while a master node can reside on a user device, such as a tablet device, smartphone, laptop, or desktop computer. This may enable running large-scale robotic artificial brains, where part of the computation occurs on the processors on board of the robot (e.g., a CPU and a GPU on a Single Board Computer(SBC); another example can be a portable handheld device installed onboard and controlling the robot with their respective CPU and GPU), and part of the computation occurs on a server connected via a tethered or wireless link. In some implementations the on-board computation device may be referred to as a “Local brain” and the remote computation device may be referred to as a “Cloud brain” (e.g., see Figure 4), but in both cases these may be hosting slave nodes, while the master node may allow the user to control the system remotely.

[0051] Referring to Figure 4, the Local brain 400 includes (but is not limited to) a GPU 410, which includes in turn a GPU processor 420 and associated GPU registers 430 and GPU memory 440, a Main Memory 450, a CPU 470, and a Network adapter 460.

[0052] The Cloud brain 480 includes (but is not limited to) one or more interconnected computers whether encased separately or within a cluster rack, each of which may include a GPU 482, which includes in turn a GPU processor 484 and associated GPU registers 486 and GPU memory 488, a Main Memory 490, a CPU 492, and a Network adapter 494.

[0053] In general, the Cloud brain 480 has a larger computational capacity with respect to the Local brain 400. Communication between the Local brain 400 and the Cloud brain 480 is established through the network using existing protocols which include but are not limited to Wi-Fi, Bluetooth, and others, and allows for intermediary routers that will convert wired communication with the Cloud brain 480 into wireless communication with the mobile robot. On top of these communication protocols there may be a communication layer that allows information passing between slave nodes described above.

[0054] Neural models can be distributed between slaves in the Local brain 400 and the Cloud brain 480 based on several considerations, for instance, to process some of the neural stages on board in order to reduce network communication load, or minimize the reaction time of the robot (e.g., computing optic flow for collision avoidance).

[0055] A cloud brain 480 may connect to one or more robots. Figure 5 illustrates the case of one slave node, but the method illustrated below can be generalized to an arbitrary number of slave nodes.

[0056] Figure 5 illustrates an example of how Local brains can update the knowledge of the Cloud brain, and how the resulting collective knowledge can be then used by each individual Local brain. The examples below pertain visual input, but the same principle can be applied to all sensory modalities and in general to any neural network stage present in the Robot brain.

[0057] In Figure 5, Local brains 1 (500a), 2 (500b), and 3 (500c) are running on Robot 1, 2, and 3, respectively, in order to learn the identity of three different objects. Robot 1 (502a), 2 (502b) and 3 (502c) can belong to the same user, or to respective users. Since, in some implementations, the robots may be sharing knowledge, the robots' users may, *a priori*, agree to share it. Such implementations may be useful to users within a company where all

robots share company-specific knowledge, as a non-limiting example. In this example, the Cloud brain 504, which can reside in one or multiple slave nodes, has a Main Memory 506 connected to a plurality of CPU or non-CPU devices 508a-e, and the resulting brain jointly learns the identity of the objects encountered by each robot. Given each Robot is controlled by the Cloud Brain, the knowledge gained via one Robot input is available to all Robot connected to the same Cloud brain.

[0058] Figure 6 illustrates an example of an aggregate Cloud brain 602, where there exists a physical separation of the processors 610 and the memory 606, which includes memory blocks 608a–608c (collectively, dedicated memory blocks 608) and processors 604a–604f (collectively, processors 604) that form respective cloud brains 610a–610c (collectively, cloud brains 610) dedicated to each Robot 600a–600c. This version can be used by users who do not wish to share the robotic knowledge on a constant basis. In some implementations sharing may still be possible through specific requests, e.g., if all of the users involved in the exchange of information agree to share. In this example, each Robot 600 may have a separate set of processors 604a–604f and associated memory blocks 608a–608c available that may not overlap with other Robots.

[0059] Figure 7 illustrates an example of how the knowledge derived from the portion of the Cloud brain 602 dedicated to a robot 600 can be transferred to another robot by copying and pasting 708 the model architecture and the associated weight matrices processed by the corresponding processors 604 and stored in the respective dedicated memory block 608 or the Main memory 606.

[0060] Figure 8 illustrates an example of how dedicated Cloud brains can share knowledge without copying or pasting network structure or weights. In the example, a dedicated Cloud brain 610a has been exposed to the visual presentation of three cars, coded by the network represented in 816, which includes neurons coding for different views of the cars (e.g., Object views 818), of the collective object “Car” (e.g., Object category 820), and the representation of an associated verbal label to this object (e.g., “Car” 822). These neural representations stored in Main memory 606 of the dedicated Cloud brain 610a and processed by processors 604a and 604b can be reactivated, for instance in a “sleep” cycle (or off-cycle, namely when the Robot is not active), in the following way:

- A “Car” label is activated in the Cloud brain 610a;

- This leads to the activation of 1 or more Object categories in Cloud brain 610a associated with that name;
- Each Object category has 1 or more Object views, represented the neural representation of an object associated with that object view, e.g. multiple views of a car (in the figure, only one view of each car class is represented);
- Each Object view and its associated label are then presented to the Cloud brain 610c;
- In the the Cloud brain 610c, previous knowledge was already present in 814 (in the example, the “Car” category already included one example);
- As a result of this learning, the Cloud brain 610c now has knowledge of all objects learned from Cloud brain 610a plus its own knowledge (814), and can further process the information via processors 610e and 610f.

[0061] The same process can be applied for different data samples of the same object or environment, e.g., when different Robots have collected several views of the same object from different angles, or when several robots have explored the same environment with different paths.

[0062] Conclusion

[0063] While various inventive embodiments have been described and illustrated herein, those of ordinary skill in the art will readily envision a variety of other means and/or structures for performing the function and/or obtaining the results and/or one or more of the advantages described herein, and each of such variations and/or modifications is deemed to be within the scope of the inventive embodiments described herein. More generally, those skilled in the art will readily appreciate that all parameters, dimensions, materials, and configurations described herein are meant to be exemplary and that the actual parameters, dimensions, materials, and/or configurations will depend upon the specific application or applications for which the inventive teachings is/are used. Those skilled in the art will recognize, or be able to ascertain using no more than routine experimentation, many equivalents to the specific inventive embodiments described herein. It is, therefore, to be understood that the foregoing embodiments are presented by way of example only and that, within the scope of the appended claims and equivalents thereto, inventive embodiments may be practiced otherwise than as specifically described and claimed. Inventive embodiments of the present disclosure are directed to each individual feature, system, article, material, kit, and/or method described herein. In addition, any combination of two or more such features, systems, articles, materials, kits, and/or methods, if such features, systems, articles, materials,

kits, and/or methods are not mutually inconsistent, is included within the inventive scope of the present disclosure.

[0064] The above-described embodiments can be implemented in any of numerous ways. For example, embodiments of designing and making the coupling structures and diffractive optical elements disclosed herein may be implemented using hardware, software or a combination thereof. When implemented in software, the software code can be executed on any suitable processor or collection of processors, whether provided in a single computer or distributed among multiple computers.

[0065] Further, it should be appreciated that a computer may be embodied in any of a number of forms, such as a rack-mounted computer, a desktop computer, a laptop computer, or a tablet computer. Additionally, a computer may be embedded in a device not generally regarded as a computer but with suitable processing capabilities, including a Personal Digital Assistant (PDA), a smart phone or any other suitable portable or fixed electronic device.

[0066] Also, a computer may have one or more input and output devices. These devices can be used, among other things, to present a user interface. Examples of output devices that can be used to provide a user interface include printers or display screens for visual presentation of output and speakers or other sound generating devices for audible presentation of output. Examples of input devices that can be used for a user interface include keyboards, and pointing devices, such as mice, touch pads, and digitizing tablets. As another example, a computer may receive input information through speech recognition or in other audible format.

[0067] Such computers may be interconnected by one or more networks in any suitable form, including a local area network or a wide area network, such as an enterprise network, and intelligent network (IN) or the Internet. Such networks may be based on any suitable technology and may operate according to any suitable protocol and may include wireless networks, wired networks or fiber optic networks.

[0068] The various methods or processes (e.g., of designing and making the coupling structures and diffractive optical elements disclosed above) outlined herein may be coded as software that is executable on one or more processors that employ any one of a variety of operating systems or platforms. Additionally, such software may be written using any of a number of suitable programming languages and/or programming or scripting tools, and also

may be compiled as executable machine language code or intermediate code that is executed on a framework or virtual machine.

[0069] In this respect, various inventive concepts may be embodied as a computer readable storage medium (or multiple computer readable storage media) (e.g., a computer memory, one or more floppy discs, compact discs, optical discs, magnetic tapes, flash memories, circuit configurations in Field Programmable Gate Arrays or other semiconductor devices, or other non-transitory medium or tangible computer storage medium) encoded with one or more programs that, when executed on one or more computers or other processors, perform methods that implement the various embodiments of the invention discussed above. The computer readable medium or media can be transportable, such that the program or programs stored thereon can be loaded onto one or more different computers or other processors to implement various aspects of the present invention as discussed above.

[0070] The terms “program” or “software” are used herein in a generic sense to refer to any type of computer code or set of computer-executable instructions that can be employed to program a computer or other processor to implement various aspects of embodiments as discussed above. Additionally, it should be appreciated that according to one aspect, one or more computer programs that when executed perform methods of the present invention need not reside on a single computer or processor, but may be distributed in a modular fashion amongst a number of different computers or processors to implement various aspects of the present invention.

[0071] Computer-executable instructions may be in many forms, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Typically the functionality of the program modules may be combined or distributed as desired in various embodiments.

[0072] Also, data structures may be stored in computer-readable media in any suitable form. For simplicity of illustration, data structures may be shown to have fields that are related through location in the data structure. Such relationships may likewise be achieved by assigning storage for the fields with locations in a computer-readable medium that convey relationship between the fields. However, any suitable mechanism may be used to establish a relationship between information in fields of a data structure, including through the use of pointers, tags or other mechanisms that establish relationship between data elements.

[0073] Also, various inventive concepts may be embodied as one or more methods, of which an example has been provided. The acts performed as part of the method may be ordered in any suitable way. Accordingly, embodiments may be constructed in which acts are performed in an order different than illustrated, which may include performing some acts simultaneously, even though shown as sequential acts in illustrative embodiments.

[0074] All definitions, as defined and used herein, should be understood to control over dictionary definitions, definitions in documents incorporated by reference, and/or ordinary meanings of the defined terms.

[0075] The indefinite articles “a” and “an,” as used herein in the specification and in the claims, unless clearly indicated to the contrary, should be understood to mean “at least one.”

[0076] The phrase “and/or,” as used herein in the specification and in the claims, should be understood to mean “either or both” of the elements so conjoined, i.e., elements that are conjunctively present in some cases and disjunctively present in other cases. Multiple elements listed with “and/or” should be construed in the same fashion, i.e., “one or more” of the elements so conjoined. Other elements may optionally be present other than the elements specifically identified by the “and/or” clause, whether related or unrelated to those elements specifically identified. Thus, as a non-limiting example, a reference to “A and/or B”, when used in conjunction with open-ended language such as “comprising” can refer, in one embodiment, to A only (optionally including elements other than B); in another embodiment, to B only (optionally including elements other than A); in yet another embodiment, to both A and B (optionally including other elements); etc.

[0077] As used herein in the specification and in the claims, “or” should be understood to have the same meaning as “and/or” as defined above. For example, when separating items in a list, “or” or “and/or” shall be interpreted as being inclusive, i.e., the inclusion of at least one, but also including more than one, of a number or list of elements, and, optionally, additional unlisted items. Only terms clearly indicated to the contrary, such as “only one of” or “exactly one of,” or, when used in the claims, “consisting of,” will refer to the inclusion of exactly one element of a number or list of elements. In general, the term “or” as used herein shall only be interpreted as indicating exclusive alternatives (i.e. “one or the other but not both”) when preceded by terms of exclusivity, such as “either,” “one of,” “only one of,” or “exactly one of.” “Consisting essentially of,” when used in the claims, shall have its ordinary meaning as used in the field of patent law.

[0078] As used herein in the specification and in the claims, the phrase “at least one,” in reference to a list of one or more elements, should be understood to mean at least one element selected from any one or more of the elements in the list of elements, but not necessarily including at least one of each and every element specifically listed within the list of elements and not excluding any combinations of elements in the list of elements. This definition also allows that elements may optionally be present other than the elements specifically identified within the list of elements to which the phrase “at least one” refers, whether related or unrelated to those elements specifically identified. Thus, as a non-limiting example, “at least one of A and B” (or, equivalently, “at least one of A or B,” or, equivalently “at least one of A and/or B”) can refer, in one embodiment, to at least one, optionally including more than one, A, with no B present (and optionally including elements other than B); in another embodiment, to at least one, optionally including more than one, B, with no A present (and optionally including elements other than A); in yet another embodiment, to at least one, optionally including more than one, A, and at least one, optionally including more than one, B (and optionally including other elements); etc.

[0079] In the claims, as well as in the specification above, all transitional phrases such as “comprising,” “including,” “carrying,” “having,” “containing,” “involving,” “holding,” “composed of,” and the like are to be understood to be open-ended, i.e., to mean including but not limited to. Only the transitional phrases “consisting of” and “consisting essentially of” shall be closed or semi-closed transitional phrases, respectively, as set forth in the United States Patent Office Manual of Patent Examining Procedures, Section 2111.03.

CLAIMS

1. A method of processing sensory data in parallel, comprising:
 - (A) receiving, at a master node, a request to process incoming sensor data;
 - (B) receiving, from the master node, a selection of a set of slave nodes, the set of slave nodes selected from a set of nodes capable of processing the incoming sensor data;
 - (C) establishing a connection between the master node and the set of slave nodes; and, for at least one slave node in the set of slave nodes:
 - (D) determining a dynamic system model for a portion of the incoming sensor data received by the at least one slave node;
 - (E) distributing, from the master node to the at least one slave node, a representation of the dynamic system model and a representation of corresponding graph connections between slave nodes in the set of slave nodes;
 - (F) instantiating at least one local data structure for processing the incoming sensor data based at least in part upon the representation of the dynamic system model distributed in (E);
 - (G) connecting the at least one slave node to other slave nodes in the set of slave nodes according to the representation of the graph connections distributed in (E);
 - (H) receiving a signal from the master node to process the incoming sensor data; and
 - (I) iteratively processing the incoming sensor data using the at least one local data structure instantiated in (F).
2. The method of claim 1, wherein (B) further comprises selecting the set of slave nodes automatically.
3. The method of claim 1, wherein (B) further comprises selecting the set of slave nodes according to user input.
4. The method of claim 1, wherein (E) further comprises:
 - assigning at least one system of equations corresponding to the dynamical system model to the at least one slave node based on at least one of: computational power of the at least one slave node, computational complexity of the at least one system of equations; and

communication between parts of a dynamical system represented by the dynamical system model.

5. The method of claim 4, wherein (E) further comprises:
assigning, by the at least one slave node, a set of computational threads for processing at least a portion of the incoming sensor data.
6. The method of claim 1, wherein (G) further comprises:
(G1) determining an operating mode of the master node; and
(G2) connecting the master node to the at least one slave node based on the operating mode of the master node.
7. The method of claim 6, wherein (C) comprises:
connecting the master node to the at least one slave node via a network connection if the master node is configured to operate in a networked mode; and
compiling the slave node into the master node if the master node is configured to operate in a standalone mode.
8. The method of claim 6, wherein the master node is configured to operate in a networked mode or a standalone mode based on available resources.
9. The method of claim 8, wherein the master node is configured to operate in the networked mode if (i) the master node is able to connect to a network and (ii) there is at least one slave node connected to the network.
10. The method of claim 1, further comprising:
(J) storing results of the iterative processing in a data queue in the at least one slave node for communication with the other slave nodes in the set of slave nodes.
11. The method of claim 1, further comprising:
(K) storing results of the iterative processing in a storage medium for off-line analysis.
12. A system for processing and forwarding slave node data, the system comprising:
(A) a first slave node operatively coupled to:
(A1) an outgoing axonal queue associated with a system of equations processed on the first slave node;

(A2) a shadow axonal queue corresponding to respective outgoing axonal queues of other slave nodes configured to provide input data to the first slave node;

(A3) a set of synaptic queues corresponding to the system of equations processed on the first slave node, the set of synaptic queues configured to preprocess the input data received from the shadow axonal queue; and

(A4) a processing device, operably coupled to the set of synaptic queues, to apply the system of equations to the input data preprocessed by the synaptic queue so as to generate at least one output data value; and

(B) a local outgoing axonal queue, operatively coupled to the processing device, to receive the at least one output data value from the processing device and to provide the at least one output data value to a second shadow axonal queue in at least one of the first slave node or a second slave node.

13. The system of claim 12, wherein the shadow axonal queue is configured to receive at least one data value from a corresponding outgoing axonal queue operatively coupled to a slave node.

14. A system for aggregating and processing sensor data from a robot comprising a local processor that implements a local brain, the system comprising:

at least one slave node, located on a computer network, to implement a cloud brain for receiving and processing the sensor data from the robot, the first processor including:

a main memory module to store the sensor data from the robot, and

a communications device, operably coupled to the main memory module, to transmit at least a portion of the sensor data to the local brain implemented in the local processor of the robot and to receive, from the local brain, at least one data value based at least in part on the at least a portion of the sensor data and on a current state of the local brain.

15. The system of claim 14, wherein the main memory module stores a representation of at least one of:

an outgoing axonal queue associated with a system of equations processed on a first slave node;

a shadow axonal queue corresponding to respective outgoing axonal queue of a second slave node configured to provide input data to the first slave node;

a synaptic queues corresponding to the system of equations processed on the first slave node, the set of synaptic queues configured to preprocess the input data received from the shadow axonal queue; and

a data structure for processing the sensor data from the robot.

16. The system of claim 14, wherein the communications device is operably coupled to another local brain implemented in another robot and configured to transmit the at least one data value to the other local brain.

17. The system of claim 16, wherein the communications device is further configured to transmit data from a dedicated memory block in the main memory module to the other local brain and to receive, from the other local brain, at least one other data value based at least in part on the data from the dedicated memory block.

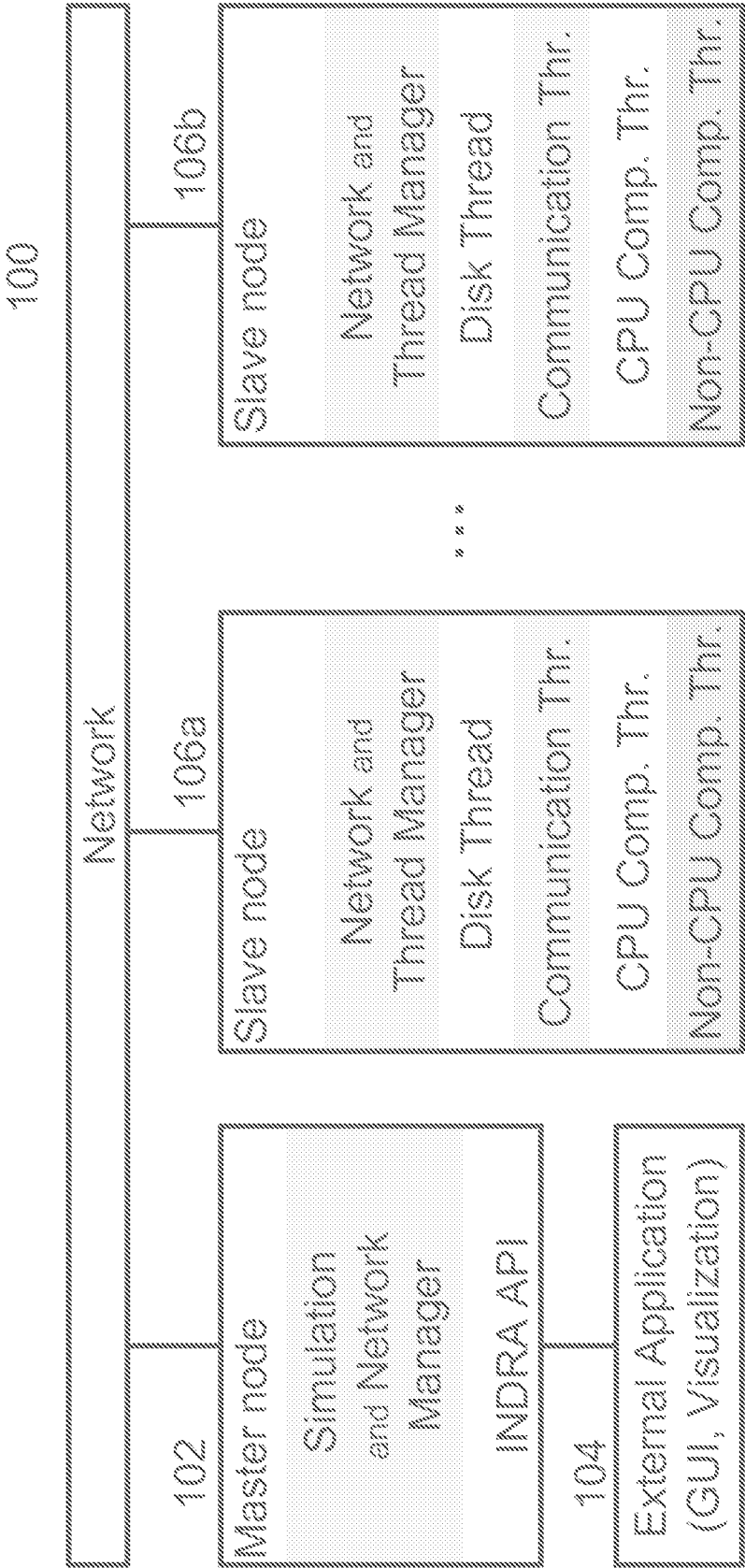


Figure 1A

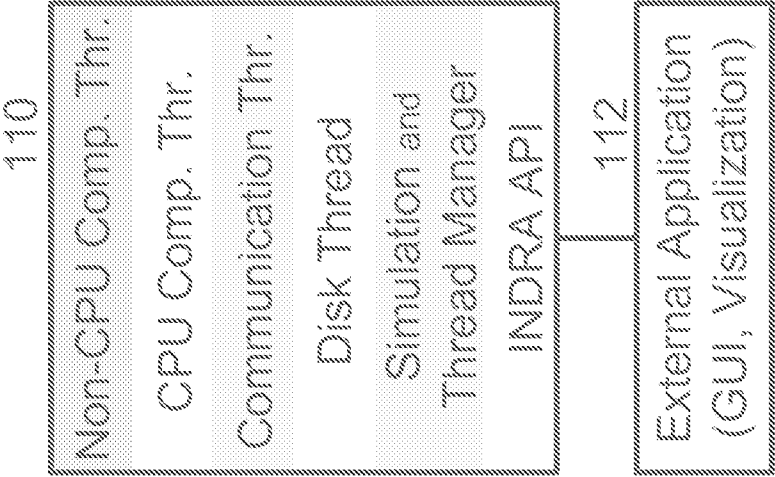


Figure 1B

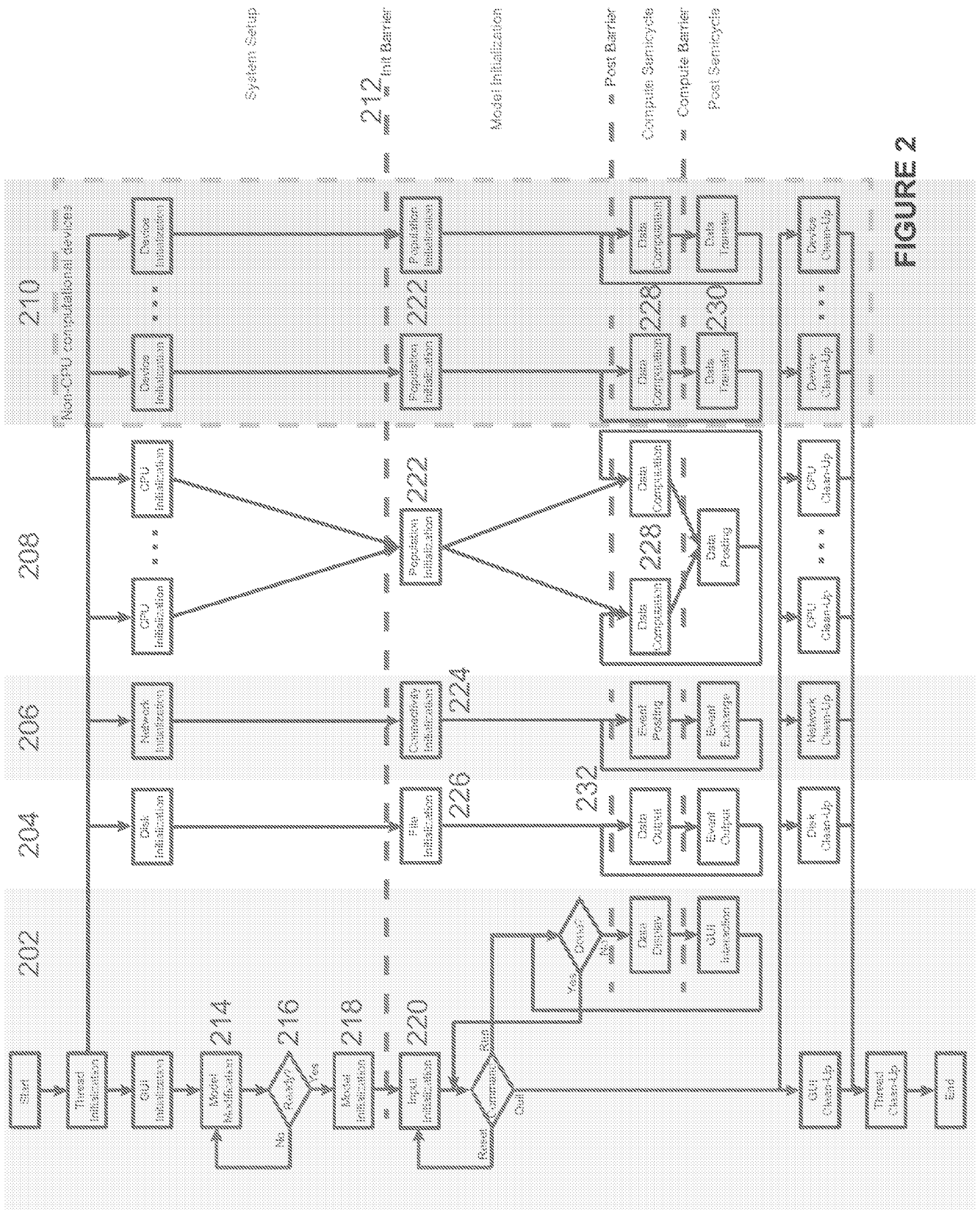
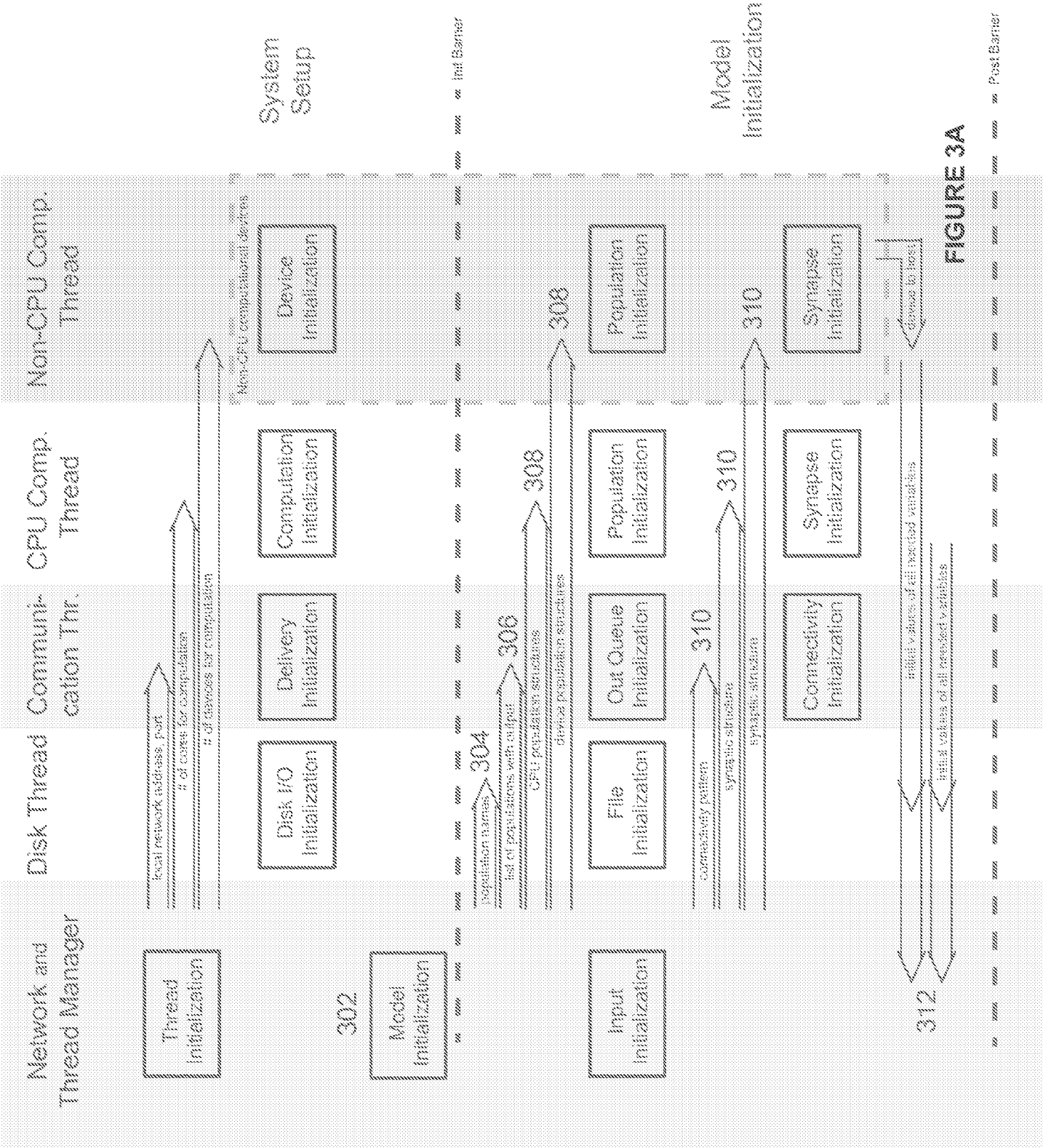


FIGURE 2



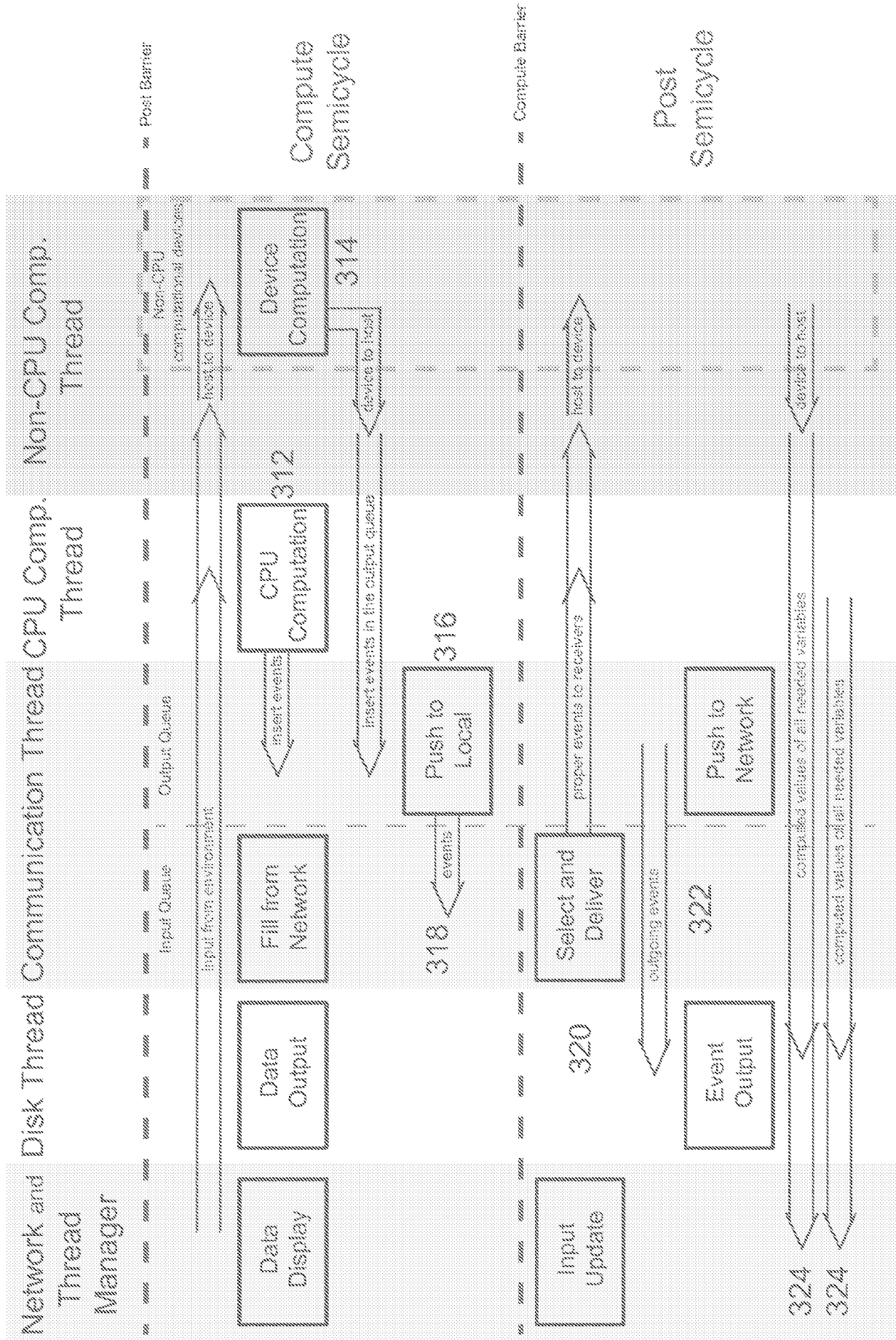
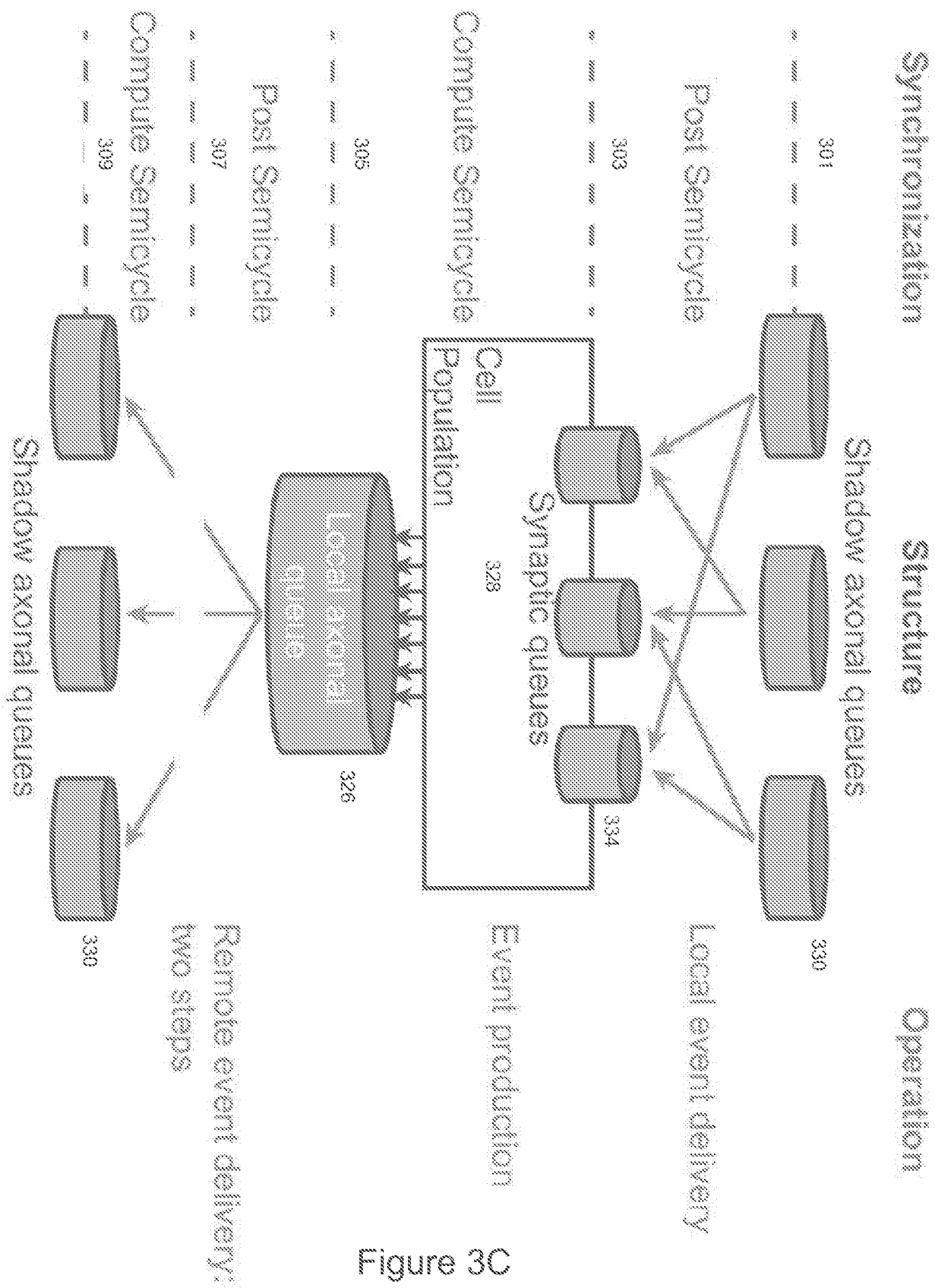


FIGURE 3B



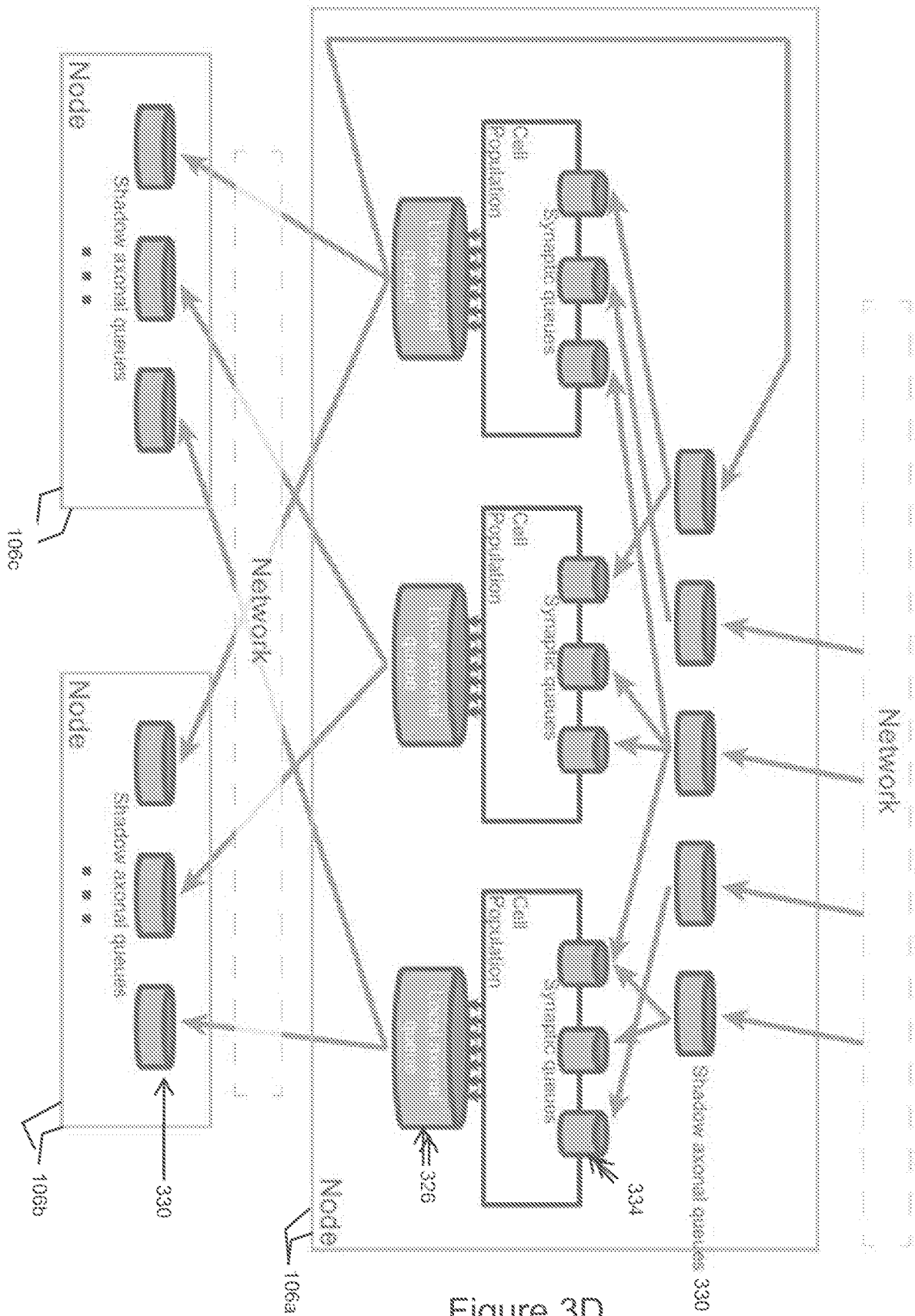


Figure 3D

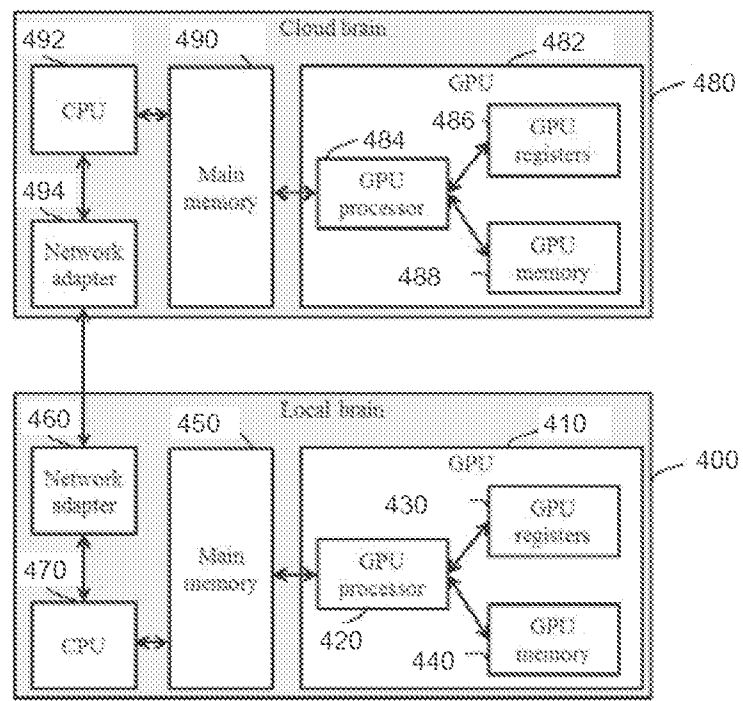


Figure 4

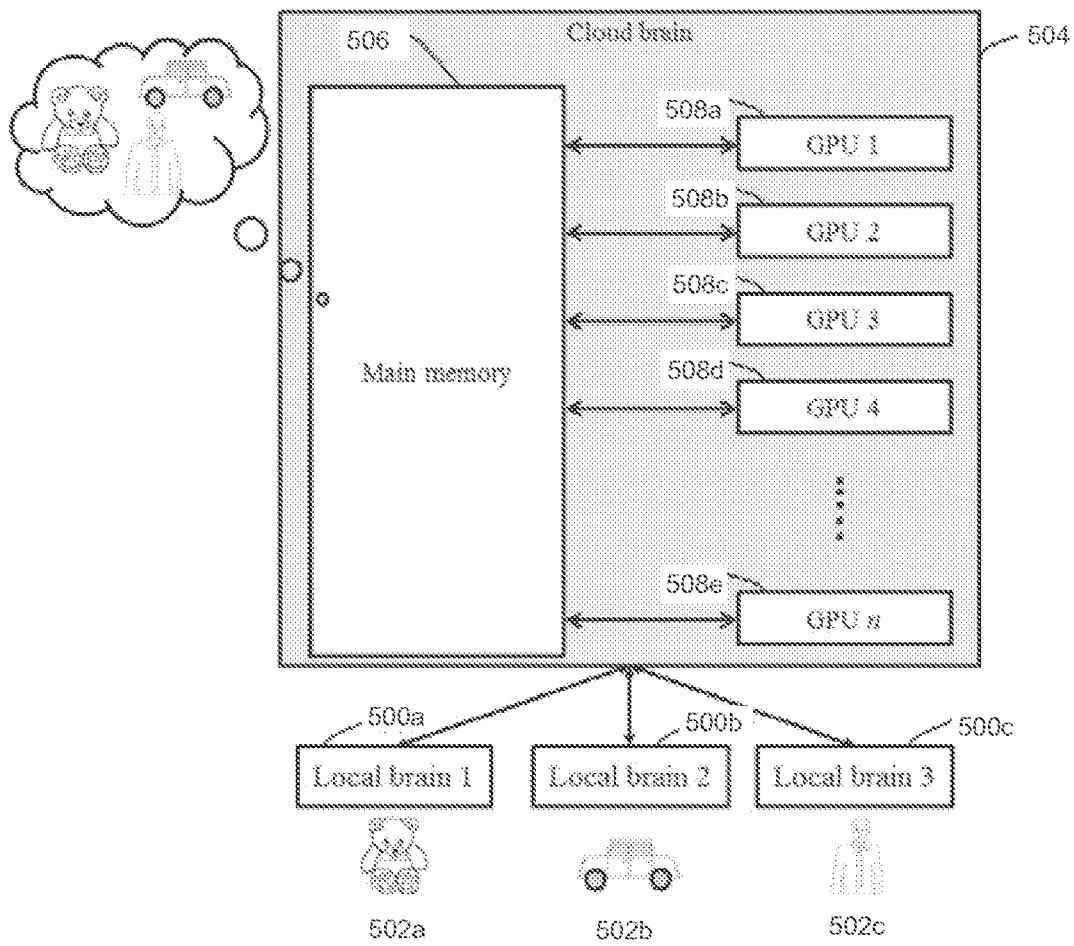


Figure 5

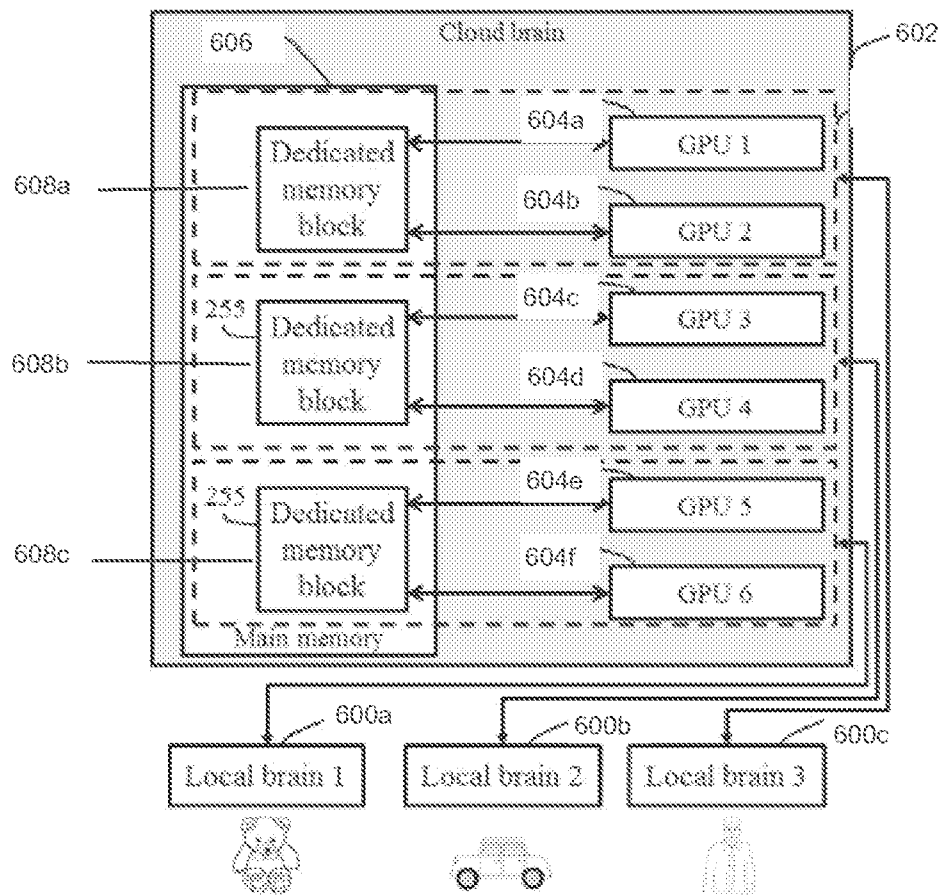


Figure 6

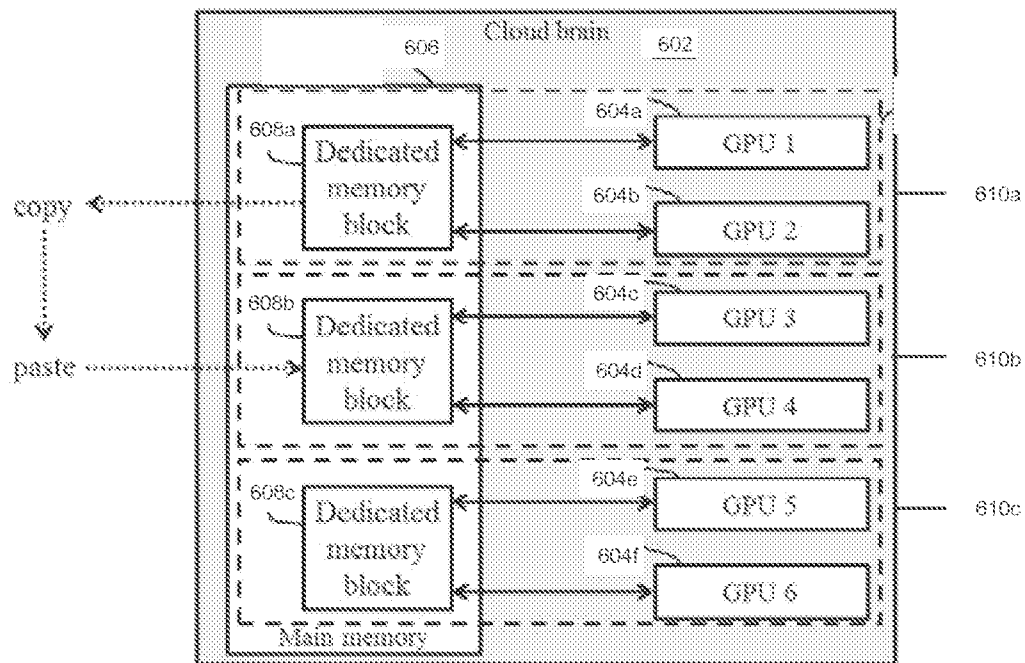


Figure 7

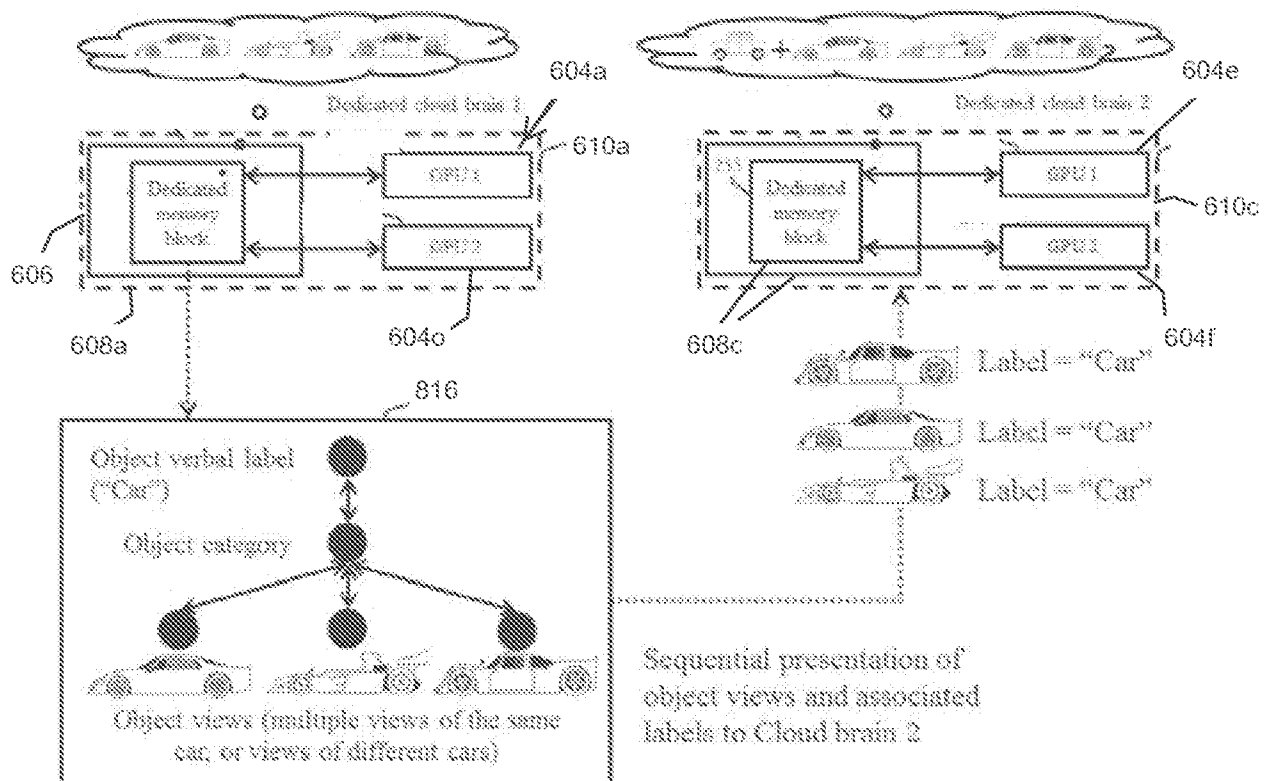


Figure 8