



(51) International Patent Classification:

B25J 9/16 (2006.01)

G05B 13/02 (2006.01)

(21) International Application Number:

PCT/US2022/026403

(22) International Filing Date:

26 April 2022 (26.04.2022)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

63/180,609

27 April 2021 (27.04.2021)

US

(71) Applicant: NVIDIA CORPORATION [US/US]; 2788

San Tomas Expressway, Santa Clara, California 95051 (US).

(72) Inventors: RATLIFF, Nathan, Donald; 2788 San Tomas

Expressway, Santa Clara, California 95051 (US). VAN

WYK, Karl; 2788 San Tomas Expressway, Santa Clara,

California 95051 (US). XIE, Man; 2788 San Tomas Ex-

pressway, Santa Clara, California 95051 (US). LI, An-

qi; 2788 San Tomas Expressway, Santa Clara, California

95051 (US). RANA, Muhammad, Asif; 2788 San Tomas

Expressway, Santa Clara, California 95051 (US).

(74) Agent: WYCKOFF, Tim, R. et al.; c/o IP Docketing Dept.,

Davis Wright Tremaine LLP, 920 Fifth Avenue, Suite 3300, Seattle, Washington 98104-1610 (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, DJ, DK, DM, DO,

DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN,

HR, HU, ID, IL, IN, IR, IS, IT, JM, JO, JP, KE, KG, KH,

KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA,

MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI,

NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU,

RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM,

TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM,

ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH,

GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ,

UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ,

TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK,

EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV,

MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,

TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,

KM, ML, MR, NE, SN, TD, TG).

(54) Title: POLICY LAYERS FOR MACHINE CONTROL

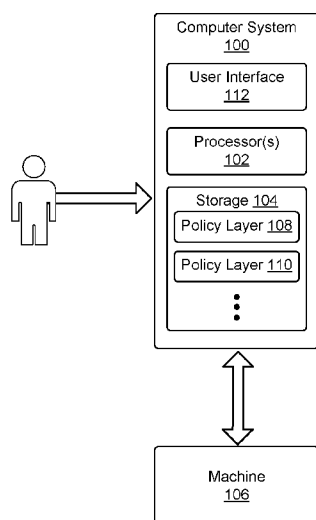


FIG. 1

(57) Abstract: Apparatuses, systems, and techniques provide a policy that can be executed to cause a machine to move. In at least one embodiment, a first policy layer is provided to cause the machine to execute a first motion that causes the machine to accelerate to reach an unbiased state. A second policy layer is provided to cause the machine to execute a second motion without influencing the unbiased state to be reached by machine. The policy can comprise the first and second policy layers.

Declarations under Rule 4.17:

- *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
- *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

POLICY LAYERS FOR MACHINE CONTROL

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Application No. 63/180,609, entitled "POLICY LAYERS FOR MACHINE CONTROL," filed April 27, 2021, the entire contents of which is incorporated herein by reference.

TECHNICAL FIELD

[0002] At least one embodiment pertains to control of a machine. For example, at least one embodiment pertains to a machine, such as a robot, that is controlled based on a policy.

BACKGROUND

[0003] Many machines are programmed to use one or more end effectors to manipulate one or more objects. For example, a machine, such as a robot, may utilize a gripper or other implement to apply force to an object thereby causing movement of that object. For instance, the machine may utilize the gripper or other implement to displace an object without necessarily grasping that object. In addition, the machine may utilize the gripper to pick up an object from a first location, move the object to a second location, and place the object at the second location. The complexity of the tasks performed by machines is increasing. The programming associated with these machines is evolving in response to that increased complexity, and to ensure that the machines perform their tasks correctly and safely.

BRIEF DESCRIPTION OF DRAWINGS

[0004] FIG. 1 illustrates an example of a system environment to control a machine, according to at least one embodiment;

[0005] FIG. 2 illustrates an example of a system environment to control a robot to initiate engaging an object, according to at least one embodiment;

[0006] FIG. 3 illustrates an example of a system environment to control a robot to engage an object, according to at least one embodiment;

[0007] FIG. 4 illustrates another example of a system environment to control a robot, according to at least one embodiment;

according to at least one embodiment;

[0009] FIG. 6 illustrates additional details of the system environment to control the robot, according to at least one embodiment;

5 [0010] FIG. 7 illustrates an example of a process to cause a computer-implemented action, such as causing a machine to move, in accordance with an embodiment;

[0011] FIG. 8 illustrates another example of a process to cause a computer-implemented action, such as causing a machine to move, in accordance with an embodiment;

10 [0012] FIG. 9A illustrates inference and/or training logic, according to at least one embodiment;

[0013] FIG. 9B illustrates inference and/or training logic, according to at least one embodiment;

[0014] FIG. 10 illustrates training and deployment of a neural network, according to at least one embodiment;

15 [0015] FIG. 11 illustrates an example data center system, according to at least one embodiment;

[0016] FIG. 12A illustrates an example of an autonomous vehicle, according to at least one embodiment;

20 [0017] FIG. 12B illustrates an example of camera locations and fields of view for the autonomous vehicle of FIG. 12A, according to at least one embodiment;

[0018] FIG. 12C is a block diagram illustrating an example system architecture for the autonomous vehicle of FIG. 12A, according to at least one embodiment;

[0019] FIG. 12D is a diagram illustrating a system for communication between cloud-based server(s) and the autonomous vehicle of FIG. 12A, according to at least one embodiment;

25 [0020] FIG. 13 is a block diagram illustrating a computer system, according to at least one embodiment;

[0021] FIG. 14 is a block diagram illustrating a computer system, according to at least one embodiment;

- [0023] FIG. 16 illustrates a computer system, according to at least one embodiment;
- [0024] FIG. 17A illustrates a computer system, according to at least one embodiment;
- [0025] FIG. 17B illustrates a computer system, according to at least one embodiment;
- 5 [0026] FIG. 17C illustrates a computer system, according to at least one embodiment;
- [0027] FIG. 17D illustrates a computer system, according to at least one embodiment;
- [0028] FIG. 17E and 17F illustrate a shared programming model, according to at least one embodiment;
- [0029] FIG. 18 illustrates exemplary integrated circuits and associated graphics processors,
10 according to at least one embodiment;
- [0030] FIGS. 19A and 19B illustrate exemplary integrated circuits and associated graphics processors, according to at least one embodiment;
- [0031] FIGS. 20A and 20B illustrate additional exemplary graphics processor logic according to at least one embodiment;
- 15 [0032] FIG. 21 illustrates a computer system, according to at least one embodiment;
- [0033] FIG. 22A illustrates a parallel processor, according to at least one embodiment;
- [0034] FIG. 22B illustrates a partition unit, according to at least one embodiment;
- [0035] FIG. 22C illustrates a processing cluster, according to at least one embodiment;
- [0036] FIG. 22D illustrates a graphics multiprocessor, according to at least one
20 embodiment;
- [0037] FIG. 23 illustrates a multi-graphics processing unit (GPU) system, according to at least one embodiment;
- [0038] FIG. 24 illustrates a graphics processor, according to at least one embodiment;
- [0039] FIG. 25 is a block diagram illustrating a processor micro-architecture for a
25 processor, according to at least one embodiment;
- [0040] FIG. 26 illustrates a deep learning application processor, according to at least one embodiment;

according to at least one embodiment;

[0042] FIG. 28 illustrates at least portions of a graphics processor, according to one or more embodiments;

5 [0043] FIG. 29 illustrates at least portions of a graphics processor, according to one or more embodiments;

[0044] FIG. 30 illustrates at least portions of a graphics processor, according to one or more embodiments;

10 [0045] FIG. 31 is a block diagram of a graphics processing engine of a graphics processor in accordance with at least one embodiment;

[0046] FIG. 32 is a block diagram of at least portions of a graphics processor core, according to at least one embodiment;

[0047] FIGS. 33A and 33B illustrate thread execution logic including an array of processing elements of a graphics processor core according to at least one embodiment;

15 [0048] FIG. 34 illustrates a parallel processing unit ("PPU"), according to at least one embodiment;

[0049] FIG. 35 illustrates a general processing cluster ("GPC"), according to at least one embodiment;

20 [0050] FIG. 36 illustrates a memory partition unit of a parallel processing unit ("PPU"), according to at least one embodiment;

[0051] FIG. 37 illustrates a streaming multi-processor, according to at least one embodiment;

[0052] FIG. 38 is an example data flow diagram for an advanced computing pipeline, in accordance with at least one embodiment;

25 [0053] FIG. 39 is a system diagram for an example system for training, adapting, instantiating and deploying machine learning models in an advanced computing pipeline, in accordance with at least one embodiment;

[0054] FIG. 40 includes an example illustration of an advanced computing pipeline 3910A for processing imaging data, in accordance with at least one embodiment;

an ultrasound device, in accordance with at least one embodiment;

[0056] FIG. 41B includes an example data flow diagram of a virtual instrument supporting an CT scanner, in accordance with at least one embodiment;

5 [0057] FIG. 42A illustrates a data flow diagram for a process to train a machine learning model, in accordance with at least one embodiment; and

[0058] FIG. 42B is an example illustration of a client-server architecture to enhance annotation tools with pre-trained annotation models, in accordance with at least one embodiment.

10

DETAILED DESCRIPTION

[0059] Programming a machine, such as a robot or vehicle, to execute a desired movement is an important task. Specifically, the programming of the machine should ensure that the machine is able to perform a desired movement. Furthermore, the programming should ensure that the desired movement is performed in a predictable and safe manner.

15

[0060] In at least one embodiment, a policy to control a machine, such as cause the machine to move, is designed using a layered approach. Specifically, the policy is generated from a plurality of policy layers. Each policy layer in the policy can control a desired movement of the machine. In at least one embodiment, a base or first policy layer of the policy is to control the movement of the machine from rest to some destination associated with a work or task space or area of the machine. The base policy layer is designed to cause the machine to reach a static or unbiased state at the destination. Specifically, the base policy layer is designed to cause the machine to come to rest at the destination. In at least one embodiment, a subsequent layer or second layer is added to the policy that includes the first layer. The second layer is to control the movement of the machine from the unbiased state or destination reached using the first layer of the policy. Alternatively, the second layer can be added to the policy including the first layer to cause the machine to avoid at least one obstacle, such as an object. In at least one embodiment, the second layer is to cause movement of the machine without influencing the biasing of the machine caused by the first layer. For example, in at least one embodiment, the second layer of the policy can cause the machine to move without influencing an unbiased state reached by the machine based on the first layer of the policy. A policy implemented using one or more policy layers may be referred to herein as a geometric fabric. In at least one embodiment, the geometric fabric is to

20

25

30

one embodiment, each policy layer causes a machine to execute at least motion or movement, and each successive policy layer builds on a motion(s) or movement(s) caused by one or more previously executed policy layers.

5 [0061] As indicated, motion generation is important for the control of machines, such as robots and vehicles. Specifically, fast, reactive motion is important for most modern tasks, especially in highly dynamic and uncertain collaborative environments. The geometric fabrics described herein provide direct construction of stable robotic behavior in modular parts. Geometric fabrics define a nominal behavior for machines, independent of a specific
10 task, capturing cross task commonalities like joint limit avoidance, obstacle avoidance, and redundancy resolution. In at least one embodiment, the described geometric fabrics cause a machine to execute one or more movements through a policy layered approach implemented by a geometric fabric.

[0062] In at least one embodiment, a machine is expected to reach a target point. For
15 example, a robot may be expected to reach a target point using the robot's end effector, such as a gripper. The robot's movement to reach the target point should avoid obstacles and limits of the robot's joints. The geometric fabric to cause the movement by the robot should resolve redundancy intelligently, implement global navigation heuristics, and the geometric fabric may need to shape the gripper's path to approach a target from a specific direction. In at least
20 one embodiment, the geometric fabric includes a plurality of policy layers, where each layer has an associated differential equation that is used to implement a movement that the robot is to execute. In at least one embodiment, the differential equation associated with each layer of the geometric fabric is a second order differential equation. The second order differential equation of each layer can be a second order differential equation with an associated metric
25 tensor that optimizes when forced. In at least one embodiment, layers of the geometric fabric are based on Lagrangian and/or geometric formulations. Lagrangians include associated equations of motion, and general nonlinear geometries can include a special class of Finsler geometries. At a high-level, a nonlinear geometry is a differential equation whose solutions define a set of paths (not speed-dependent trajectories, but speed-independent paths). An
30 example is a Riemannian geometry, a type of Finsler geometry, whose paths are length minimizing under the Riemannian length measure.

geometric fabric, can be generated through a nonlinear geometry and energizing that geometry to find a representation of the geometry including of a set of trajectories along the paths that are energy conserving under a particular Lagrangian energy (e.g., an energization transform on the geometry). The nonlinear geometry defines the basic nominal behavior of the machine (i.e., the paths it will follow when unperturbed), and the Lagrangian energy defines the nature of how forces act to push the machine away from those nominal paths. Specifically, the energy has an associated energy tensor that acts as a generalized mass matrix that defines how the machine accelerates away from the geometric paths when forced.

[0064] In at least one embodiment, a computer system can be provided that generates a policy or geometric fabric that includes policy layers. The computer system can include at least a processor and storage, and a user can operate the computer system to cause the computer system to generate the policy. In at least one embodiment, the computer system includes a user interface (UI) that the user leverages to generate the policy layers that are included in the policy. For example, the user can use the UI to generate a first policy layer to cause a machine, such as a robot, to generate a first motion that reaches an unbiased state. For example, the first policy layer can cause a machine to reach a steady state or come to rest within an area in which the machine is to operate. The first policy layer can cause a machine to reach the unbiased data based on one or more parameters. In at least one embodiment, a first parameter can be associated with a joint of the machine. The first parameter can prevent the machine from exceeding a joint limitation of the machine. The one or more parameters can also include a second parameter, where the second parameter is a coordinate within the area in which the machine is to operate. In at least one embodiment, the first policy layer causes the machine to come to rest at the coordinate within the area in which the machine is to operate.

[0065] The user can use the UI to generate a second policy layer included in the policy. In at least one embodiment, the second policy layer is to cause the machine to generate a second motion. The second motion can build on the first motion that the machine executes based on the first policy layer. In at least one embodiment, the second motion associated with the second policy layer does not affect the biasing associated with the first motion caused by the first policy layer. For example, the second motion caused by the second policy layer does not influence the first motion from reaching the unbiased state. In at least one embodiment, the

acceleration and/or velocity associated with the first motion provided by the first policy layer.

[0066] FIG. 1 illustrates an example of a system environment, according to at least one embodiment. In at least one embodiment, the system environment includes a computer system 100. The computer system 100 can include one or more processors 102 and computer storage 104. The one or more processors 102 can include a graphical processing unit described below. In at least one embodiment, the computer storage 104 can include a memory that stores executable instructions that, as a result of being executed by the one or more processors 102, cause the computer system 100 to control a machine 106. In at least one embodiment, the computer system 100 can cause the machine 106 to perform one or more motions or movements. In at least one embodiment, the computer system 100 can cause the machine 106 to perform one or more motions or movements based on one or more geometric fabrics stored in the storage 104. In at least one embodiment, the one or more geometric fabrics stored in the storage 104 comprise one or more policy layers 108 and 110. There can be many policy layers stored in the storage 104.

[0067] In at least one embodiment, the machine 106 is a robot. The robot can be an articulated robot that includes one or more arms. In at least one embodiment, the machine 106 is a vehicle, such as an automobile that can be controlled by user. In at least one embodiment, one or more movements of the automobile are controlled by policy layers of the computer system 100. In at least one embodiment, one or more of the policy layers in the storage 104 can be combined together to provide a policy or geometric fabric that causes the machine 106 to perform one or more movements or motions.

[0068] In at least one embodiment, the policy layer 108 comprises a differential equation to cause the machine 106 to perform a desired movement or motion. The differential equation can be a second order differential equation. In at least one embodiment, the differential equation is homogeneous of degree two. Specifically, the differential equation, in at least one embodiment, comprises one or more trajectories that cause machine 106 to execute a motion. In addition, the differential equation comprises a path consistency property ensuring that one or more integral curves starting from a given position and having a particular velocity will follow a desired motion path. In at least one embodiment, the policy layer 110 also comprises a differential equation to cause the machine 106 to perform a desired movement or motion after the movement or motion caused by the policy layer 108. The differential equation can

is homogeneous of degree two. Specifically, the differential equation, in at least one embodiment, comprises one or more trajectories that cause machine 106 to execute a motion that builds on the motion caused by the policy layer 108. In addition, the differential equation
5 comprises a path consistency property ensuring that one or more integral curves starting from a given position and having a particular velocity will follow a desired motion path.

[0069] The policy layers 108 and/or 110 can be implemented through a UI 112.

Specifically, a user can interface with the computer system 100 through the UI 112 to generate the policy layers 108 and/or 110. In an embodiment, the policy layers 108 and/or
10 110, through the UI 112, can be constructed in parts distributed across a transformed tree of a relevant task space or area in which the machine 106 is to operate. In at least one embodiment, each of the policy layers 108 and 110 provide for stable machine movement to unbiased one or more destinations due to their construction as nonlinear geometries of one or more paths. In at least one embodiment, the policy layer 110 and the machine movements
15 caused thereby build upon the policy layer 108 and the machine movements caused thereby. Designing and implementing the policy layers 108 and 110 in a layer-wise manner mitigates design complexity and allows for independently controlling execution speed by accelerating the machine 106 along a direction of motion without affecting the overall quality of the machine's 106 movement behavior.

[0070] In at least one embodiment, one or more policies of the storage 104 are designed by the user and combined together to generate an overall policy or geometric fabric that is conveyed to the machine 106. This policy conveyed to the machine 106 causes the machine, when processed by the machine 106, to move from any initial configuration to a desired target position. In at least one embodiment, the policy or geometric fabric can cause the
25 machine 106 to move from any one of a plurality of initial or starting states to a destination state, such as an unbiased state. During movement, the policy or geometric fabric can cause the machine 106 to adapt its motion based on one or more environmental conditions, such as one or more obstacles. Furthermore, the policy or geometric fabric can cause machine 106 to execute one or more motions or movements based on joint restrictions, stiffness, and/or other
30 parameters of the machine 106. In at least one embodiment, generation of the policy layers 108 and/or 110 can be aided by one or more learning technologies, such as one or more neural networks.

embodiment. In at least one embodiment, the system environment includes a computer system 200. The computer system 200 can include one or more processors 202 and computer storage 204. The one or more processors 202 can include a graphical processing unit
5 described below. In at least one embodiment, the computer storage 204 can include a memory that stores executable instructions that, as a result of being executed by the one or more processors 202, cause the computer system 200 to control a robot 206. In at least one embodiment, a 7-DoF robotic arm, Franka Emika Panda, is used for a grasping task. In at least one embodiment, the computer system 200 can cause the robot 206 to perform one or
10 more motions or movements. In at least one embodiment, the computer system 200 can cause the robot 206 to perform one or more motions or movements based on one or more geometric fabrics stored in the storage 204. In at least one embodiment, the one or more geometric fabrics stored in the storage 204 comprise one or more policy layers 208. There can be many policy layers stored in the storage 204. The computer system 200 can also include a UI 212
15 that can be used by a user to generate or create the policy layer 208 that can cause the robot 206 to perform one or more motions or movements.

[0072] In at least one embodiment, the robot 206 is an articulated robot, such as the above referenced Franka robot. In at least one embodiment, the robot 206 is controlled, at least in part, by a controller computer system that implements a neural network. The controller
20 computer system can be implemented by the computer system 200. In at least one embodiment, the controller computer system includes a memory that stores executable instructions that, as a result of being executed by the one or more processors, cause the system to grasp an object. In at least one embodiment, the controller computer system implements policies and a neural network trained to generate control signals which cause the
25 articulated robot to grasp an object.

[0073] In the example illustrated in FIG. 2, the robot 206 includes an arm 214 connected to a base. In at least one embodiment, the arm 214 is connected to a wrist 216. In at least one embodiment, a gripper 218 is mounted on the wrist 216. In at least one embodiment, an object 220 is to be grasped by the gripper 218 under the control of the computer system 200
30 and one/or more policies of the computer system 200. In at least one embodiment, a camera is mounted to the wrist 216, and the camera is mounted so that the view of the camera is directed along the axis of the gripper 218 toward the object 220 to be grasped. In at least one

with a view of the work area, such as the task space in which the robot 206 is to operate.

[0074] In at least one embodiment, the computer system 200 executes the policy layer 208 to cause the robot 206 to move the arm 214. In at least one embodiment, the policy layer 208, when executed by the computer system 200, causes the robot 206 to move the arm 214 in a straight line or a substantially straight line. The policy layer 208 implements one or more parameters to at least ensure that joint limits, such as a joint limit associated with the arm 214 at the wrist 216, are observed. In at least one embodiment, the policy layer 208 causes the robot 206 to move the arm 214 to a coordinate point within the work area of the robot 206.

The arm 214, according to the policy layer 208, comes to rest or an unbiased state at the coordinate point within the work area of the robot 206. In at least one embodiment, the arm 214 performs a motion, in accordance with the policy layer 208, to cause the gripper 218 to come to rest in the vicinity of the object 220. In at least one embodiment, the object 220 can be located in a box, cubby, or other container. The policy layer 208, if executed alone, may cause the arm 214 and/or the gripper 218 to come into contact with the box, cubby, or the other container. Additional one or more policy layers built on top of the policy layer 208 are designed to prevent the arm 214 and/or the gripper 218 from contacting the box, cubby, or the other container that may hold the object 220.

[0075] FIG. 3 illustrates an example of a system environment, according to at least one embodiment. In FIG 3, the computer system 200 is illustrated as including a policy layer 308 in a computer storage 204. The policy layer 308 can be created or generated by a user to build upon the movement of the robot 206 caused by the policy layer 208. In at least one embodiment, the policy layer 308, when executed by the computer system 200 to cause the robot 206 to execute one or more movements, builds upon movements by the robot 206 in accordance with the policy layer 208. In at least one embodiment, the policy layer 308 does not influence the biasing caused by the policy layer 208.

[0076] In at least one embodiment, as illustrated in FIG. 3, the policy layer 306 can cause the wrist 216 of the arm 214 to pivot or rotate toward the object 220. This pivot or rotation of the wrist 216 can allow the gripper 218 to grasp the object 220.

[0077] In at least one embodiment, the described computer systems can be integrated with the described machines and robots. Alternatively, in at least one embodiment, the computer systems can be a separate systems designed to control the described machines and robots.

embodiment. In at least one embodiment, the system environment includes a computer system 400 and a robot 402. In at least one embodiment, the computer system 400 and the robot 402 are integrated together. Alternatively, the computer system 400 and the robot 402 can each be a separate computerized system comprising one or more processors and one or more computer storages that comprise computer-executable instructions. In at least one embodiment, the computer system 400 can comprise one or more of the system elements described in detail herein.

[0079] In at least one embodiment, the computer system 400 comprises a policy 404. The policy 404 can also be referred to as a geometric fabric in accordance with the disclosed techniques and methodologies. In at least one embodiment, the policy 404 is associated with one or more computer storages of the computer system 400. The policy 404 can comprise a plurality of policy layers 406-410. One or more the policy layers 406-410 can be generated by the techniques and methodologies disclosed herein. In at least one embodiment, one or more of the policy layers 406-410 is implemented by a user of the computer system 400. The policy 404 can be used to control movements executed by the robot 402 and/to cause the robot 402 to execute one or more movements. In at least one embodiment, the policy 404 is executed based on a layer by layer approach, where each of the policy layers 406-410 builds on one or more movements caused by a prior executed layer associated with the policy 404.

[0080] In at least one embodiment, referring to FIG. 4, layer 406 of the policy 404 is to cause a robotic arm 412 to move generally in the arm movement direction illustrated by the broken line shown in FIG. 4. In at least one embodiment, the layer 406 causes the arm 412 to move in a manner that avoids joint limitation parameters of the arm 412. Furthermore, the layer 406 can cause the arm 412 to move, based at least on trajectory, acceleration and/or velocity parameters, and in consideration of redundancy resolution parameters, posture control parameters associated with the arm 412 and/or cause and end-effector of the arm 412 to move to one or more desired positions. In at least one embodiment, the layer 406 causes the arm 412 to move from or to a box 414, to a box 416, and then to a box 418. As described herein, subsequent movements of the arm 412, caused by other layers of the policy 404, are designed to cause the arm 412 and its associated end-effector to grasp or otherwise interface with one or more objects associated one or more of the boxes 414-418. A goal of the policy 404 is to cause the arm 412 to grasp the one or more objects associated with one or more of the boxes 414-418 while avoiding coming into contact with the box and any other obstacle

can include a task space surface, such as a table 420 to support the boxes 414-418.

[0081] As illustrated in FIG. 5, the computer system 400 can cause the robot arm 412 to execute one or more movements based on the layer 408. In at least one embodiment, the one or movements caused by the layer 408 build upon and work in conjunction with the movements caused by the layer 406. In at least one embodiment, the layer 408 includes one or more trajectory, acceleration and/or velocity parameters that cause the end-effector, such as a gripper, of the arm 412 to move into and out of one or more of the boxes 414, 416, and 418. However, in at least one embodiment, the combined movements caused by the combination of layers 406 and 408 can still cause the arm 412 to contact obstacles, such as one or more the boxes 414-418.

[0082] As illustrated in FIG. 6, the computer system 400 can cause the robot arm 412 to execute one or more movements based on the layer 410. In at least one embodiment, the one or more movements caused by the layer 410 build upon and work in conjunction with the movements caused by the layers 406 and 408. In at least one embodiment, the layer 410 includes one or more trajectory, acceleration and/or velocity parameters to cause the end effector, such as a gripper, of the arm 412 to move into and out of one or more of the boxes 414, 416, and 418 without contacting one or more of the surfaces associated with one or more of the boxes 414, 416, and 418.

[0083] The following description will provide in-depth technical details for generating geometric fabrics, which can be used to provide the types of policies and policy layers described in the foregoing. In general, a policy layer of a fabric is an unbiased spectral semi-spray, i.e. a second order differential equation with an associated metric tensor that optimizes when forced. A fabric can be derived based on Lagrangian and geometric formulations. The description also covers the general class of spectral semi-sprays (specs), Lagrangians and their equations of motion, and general nonlinear geometries including the special class of Finsler geometries. At a high-level, a nonlinear geometry is a differential equation whose solutions define a set of paths (not speed-dependent trajectories, but speed-independent paths). An example is a Riemannian geometry, a type of Finsler geometry, whose paths are length minimizing under the Riemannian length measure.

[0084] Many Lagrangians have associated energies defined by their Hamiltonians; the main result of this disclosure is that a fabric having at least one policy layer can be defined by

geometry consisting of a set of trajectories along the paths that are energy conserving under a particular Lagrangian energy (e.g., an energization transform on the geometry). The nonlinear geometry defines the basic nominal behavior of the system (the paths it will follow when unperturbed), and the Lagrangian energy defines the nature of how forces act to push the system away from those nominal paths. Specifically, the energy has an associated energy tensor that acts as a generalized mass matrix that defines how the system accelerates away from the geometric paths when forced. Since the resulting system is a fabric, forcing it with a potential (and damping) is assured to optimize the potential. The following text also provides the necessary conditions for Lagrangian equations of motion to define a fabric, also referred herein to as a Lagrangian fabric. The term system, as used herein, can include at least a compute implemented system that comprises at least one geometric fabric and a machine, such as a robot, that is controlled at least in part based on the geometric fabric and the one or more policy layers of that geometric fabric.

[0085] Furthermore, the energization transform commutes with pullbacks across differentiable maps showing that it is possible to either energize in the codomain of the differentiable map and pull the resulting system back, or is possible to pull the energy and geometry back independently and then energize the resulting pulled back geometry in the domain, and both result in the same geometric fabric. This result, in conjunction with an analysis on minimum damping conditions needed to ensure optimization when following alternative velocity profiles during optimization, shows that nonlinear geometries and associated energies can be placed on a transform tree, and that the geometries as acceleration policies can be used to design behaviors and the energies to design spectral priority weights that define what the behaviors care about and how they combine with one another.

[0086] In the context of Riemannian Motion Policies (RMPs), this result means that geometric optimization fabrics can be used as a formal provably stable toolset for designing a flexible RMPs that separate the acceleration policy design (nonlinear geometry design) from the priority specification (energy design). On top of that, is shown that these geometric formulation methods allow for modulating speed, independent of metric choice, resulting in smooth and consistent provably stable behavior. Moreover, this formal separation of behavior into a task independent geometric fabric and a task specific forcing potential acts as a way to factor out common behavioral elements spanning many tasks allowing the geometric fabric to

generalization of learned potentials following recent results on learning RMPs.

[0087] The concepts and notation around differential geometry are complex. This description uses a notation that avoids the typical coordinate-free or tensor based notations of differential geometry in lieu of an advanced calculus notation. The manifolds on which equations are derived are well-defined with respect to the standard constructions of differential geometry, and equations are derived simply with respect to a coordinate system as is common in physics.

[0088] There is a lot of terminology introduced in this description, so the following text collects the definitions into one place for quick reference. Furthermore, the text provides a list of the different types of fabrics encountered throughout the description along with a concise taxonomy of their closure status under operations of the spec algebra defined herein.

[0089] The following is a list of terms defined in this description, each listed with a brief contextual note:

1. Spectral semi-spray, or spec for short: A pair $(M(x, \dot{x}), f(x, \dot{x}))$ representing a differential equation $M\ddot{x} + f = 0$. Operations of pullback and combination define an associated spec algebra over transform trees.
2. Equations of motion: The equation $\partial_{\ddot{x}\ddot{x}}^2 \mathcal{L} + \partial_{\dot{x}\dot{x}} \mathcal{L} \dot{x} - \partial_x \dot{\mathcal{L}} = 0$ given by applying the Euler-Lagrange equation to a stationary Lagrangian $\mathcal{L}(x, \dot{x})$.
3. Forcing a system: Adding the gradient of a potential function, often along with a damper, to a spec. If the original spec is $M\ddot{x} + f = 0$, the forced system is $M\ddot{x} + f = -\partial_x \psi - B\dot{x}$.
4. A nonlinear geometry: A geometrically consistent system of speed independent paths defined by a differential equation $\ddot{x} + h_2(x, \dot{x}) = 0$, where h_2 is homogeneous of degree 2 in velocity, known as the (geometry) generator. Each generator has an associated geometric form $P_x^\perp [\ddot{x} + h_2(x, \dot{x})] = 0$ known as the corresponding geometric equation.
5. Finsler structure: A Lagrangian that is positive when $\dot{x} \neq 0$ and homogeneous of degree 1 in velocity. Such Lagrangians define action integrals that are “path length” like, in that they are a positive measure that is invariant to time-reparameterization of

measure. A Finsler structure can also be thought of as a geometric Lagrangian since it is a specific form of Lagrangian that captures the notion of a positive speed-independent path measure with a locally unique minimum.

6. A Finsler geometry is a nonlinear geometry defined by the equations of motion of a Finsler structure. Its generator is known as a Finsler generator and is given by the equations of motion of the corresponding Finsler energy.

7. Lagrangian energy: The Hamiltonian of a general Lagrangian. This energy is defined as $\mathcal{H}_{\mathcal{L}} = \partial_{\dot{x}} \mathcal{L}^T \dot{x} - \mathcal{L}$ and in general might differ from the Lagrangian. The energy is not to be confused with the Lagrangian itself. Only when the Hamiltonian matches the Lagrangian is that the case (such as in Finsler energies below). When the context of the Lagrangian is clear, energy is often used.

8. Finsler energy: The energy form $L_e = \frac{1}{2} L_g^2$ of a Finsler structure $\mathcal{L}_g$. In this special case, the Lagrangian energy (Hamiltonian) associated with $\mathcal{L}_e$ is $\mathcal{L}_e$, itself. When the context of the Finsler structure $\mathcal{L}_g$ is known, often referred to the Finsler energy as the energy or energy form of $\mathcal{L}_g$. The Finsler energy is always homogeneous of degree 2 in $\dot{x}$ and can be used to define the Finsler geometry as $\mathcal{L}_g = \sqrt{2\mathcal{L}_e}$ if the defining properties hold for the derived Finsler structure $\mathcal{L}_g$.

9. Bending a fabric: Adding an energy conserving geometric term (homogeneous of degree 2) to a geometric fabrics' generator. The resulting generator generates a distinct geometry, but the resulting generator still conserves the same energy and remains a fabric. "Bending" is different from "forcing" (see above).

10. Metric tensor of a Lagrangian: Defined as the Hessian of the Lagrangian $\mathcal{L}$ used to define the energy $M = \partial_{\dot{x}\dot{x}}^2 \mathcal{L}$.

11. Energization or energization transform: Given a nonlinear geometry and a Lagrangian function (defining an energy), the operation of energization transforms the geometry into a geometric or semi-geometric fabric given by the Lagrangian's metric tensor and a geometry generator that represents the given nonlinear geometry but conserves the given notion of energy. The resulting fabric is either a geometric or

general Lagrangian energy, respectively.

12. Energized fabric: The energy conserving fabric resulting from an energization transform. When the energy is a Finsler energy, the fabric is a bent Finsler fabric referred to as a bent Finsler representation.

[0090] The following is a list of the classes of fabrics defined throughout this description:

1. Optimization fabric or fabric for short: A spec (M, f) which optimizes when forced. i.e. $M_e \ddot{x} + f + \partial_x \psi + B \dot{x} = 0$ optimizes psi.
2. Conservative fabric: A fabric defined by an energy conserving equation of the form $M_e \ddot{x} + f_e + f_f = 0$ where (M_e, f_e) come from the energy Lagrangian $\mathcal{L}_e(x, \dot{x})$ and f_f is a zero work contribution.
3. Lagrangian fabric: A class of conservative fabric defined by the equations of motion of an energy Lagrangian. If the fabric is defined more specifically by a Finsler energy, it is called a Finsler fabric.
4. Energized fabric: A fabric formed by energizing a differential equation. If a Finsler energy is used to energize the differential equation, it is called a Finsler energized fabric.
5. Geometric fabric: A fabric formed by energizing a geometry generator with a Finsler energy.

[0091] Let x and $\dot{x}$ denote a position and velocity in a task space $\mathcal{X}$, assumed to be represented in some chosen coordinates. Optimization fabrics instances of systems of the form $M \ddot{x} + f = 0$, where $M(x, \dot{x})$ is symmetric and invertible and $f(x, \dot{x})$ are both functions of both position and velocity, with the property that they optimize when forced with a potential, which is discuss in detail in the following description. First, though, a brief overview of the broader class of differential equations is provided.

[0092] A (non-spectral) semi-spray by itself is a differential equation of the form $\ddot{x} + h(x, \dot{x}) = 0$, and the above equation can be expressed as $\ddot{x} + M^{-1}f = 0$ to fit this form, but it is critical explicitly represent M to track how it transforms between spaces, as discussed in the following.

expression can be derived for the covariant transformation of a spec $(M, f)_X$ on the codomain X to a spec $(\tilde{M}, \tilde{f})_Q$ on the domain Q . Denoting the map's Jacobian as $J = \partial_q \phi$, and noting $\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$, the covariant transformation of left hand side of the differential equation $M\ddot{x} + f = 0$ represented by the spec is:

$$J^T(M\ddot{x} + f) = J^T(M(J\ddot{q} + \dot{J}\dot{q}) + f) \quad (1)$$

$$= (J^T M J)\ddot{q} + J^T(f + \dot{J}\dot{q}) \quad (2)$$

$$= \tilde{M}\ddot{q} + \tilde{f}. \quad (3)$$

where $\tilde{M} = J^T M J$ and $\tilde{f} = J^T(f + \dot{J}\dot{q})$. This means the following covariant pullback operation can be derived:

$$\text{pull}_\phi(M, f)_X = (J^T M J, J^T(f + \dot{J}\dot{q}))_Q. \quad (4)$$

[0094] Similarly, since $M_1\ddot{x} + f_1 + (M_2\ddot{x} + f_2) = (M_1 + M_2)\ddot{x} + (f_1 + f_2)$, an associative and commutative summation operation of the form can be derived:

$$(M_1, f_1)_X + (M_2, f_2)_X = (M_1 + M_2, f_1 + f_2)_X \quad (5)$$

[0095] Note that the above operations are natural form of specs. They can also be viewed in canonical form, which amounts to $(M, -M^{-1}f)_X^c$ in the current setting where M is fully invertible. The expression $-M^{-1}f$ defines the acceleration, since the differential equation represented by the spec can be solved to give $\ddot{x} = -M^{-1}f$. In terms of this canonical acceleration form, the summation operation computes the combined acceleration as a metric weighted average of individual accelerations:

$$(M_1, a_1)_X^c + (M_2, a_2)_X^c = (M_1 + M_2, (M_1 + M_2)^{-1}(M_1 a_1 + M_2 a_2))_X^c \quad (6)$$

[0096] These operations of pullback and summation define the spec algebra.

[0097] Differentiable maps can be composed together to make a tree of spaces rooted at the configuration space $\mathcal{C}$ known as a transform tree, with directed edges denoting the differentiable maps and nodes given by the spaces that result for those differentiable maps.

[0098] It can be shown that, in the context of a transform tree, the space of specs becomes a compatible linear structure across the tree under the above defined spec algebra. That means

nodes and pulling them back and combining them recursively until a single resultant spec resides at the root.

[0099] Since specs form a compatible linear structure on the tree, this computation is independent of computational path, which means, at least for purposes of the theoretical analysis here, the transform tree can be seen as star-shaped, under which each node has a single independent map linking directly from root to the node. Such a star-shaped tree can be represented by a collection of n differentiable maps $\phi_i : Q \rightarrow \mathcal{X}_i, i = 1, \dots, n$. The pulled back and combined spec defined on Q representing a collection of specs $\{(M_i, f_i)\}_{i=1}^n$ defined on the n spaces is:

$$\sum_{i=1}^n \text{pull}_{\phi_i} (M_i, f_i)_{\mathcal{X}_i} = \left(\sum_i J_i^T M_i J_i, \sum_i J_i^T ((f_i + J_i \dot{q})) \right)_Q \quad (7)$$

which can be viewed as a metric weighted average of individual pulled back specs in canonical form.

[0100] A class of specs is said to be closed under tree operations, or just closed for short when the set of operations is clear, when applying the operations to elements of the class results in an element of the same class.

[0101] In particular, if a given class of specs is closed under tree operations, if the transform tree is populated with specs from that class, the spec that results at the root from pullback and combination is of the same class.

[0102] A spec can be forced by a position dependent potential function $f(x)$ using

$$M\ddot{x} + f = \partial_x \psi \quad (8)$$

where the gradient $-\partial_x \psi$ defines the force added to the system. In most cases, forcing an arbitrary spec does not result in a system that's guaranteed to converge to a local minimum of ψ . But when it does, the spec is optimizing and forms an optimization fabric or fabric for short. This section characterizes the class of specs that form fabrics using definitions and results of increasing specificity. These results are used to define geometric fabrics which constitute a concrete set of tools for fabric design.

[0103] Note that the accelerations of a forced system $\ddot{x} = -M^{-1}f - M^{-1}\partial_x \psi$ decompose into nominal accelerations of the system $-M^{-1}f$ and forced accelerations $-M^{-1}\partial_x \psi$.

– $\partial_x \psi$ acts to push the system away from the nominal path. It might be easy for potentials to push in some directions, but difficult to push in others. The metric M dictates the profile of how the potential function can push away from the system's nominal paths.

5 [0104] Definition 4.1. Let $\mathcal{X}$ be a manifold with a boundary. Its boundary is denoted $\partial\mathcal{X}$ and denote its interior as $\text{int}(\mathcal{X}) = \mathcal{X} \setminus \partial\mathcal{X}$.

[0105] Note that in all cases below, when referring to a manifold's boundary $\partial\mathcal{X}$, assume $\partial\mathcal{X}$ might be empty unless otherwise explicitly stated.

[0106] Definition 4.2. Let $\mathcal{X}$ denote a smooth manifold of dimension n . The space of all
10 velocities at a point x is denoted $\mathcal{T}_x\mathcal{X}$ and is known in manifold theory as the tangent space at x . It is often convenient to reference the set of all available positions and velocities across a manifold. That space is known as the tangent bundle and is denoted $\mathcal{T}\mathcal{X} = \coprod_{x \in \mathcal{X}} \mathcal{T}_x\mathcal{X}$ where $\coprod$ denotes the disjoint union. I.e. $(x, \dot{x}) \in \mathcal{T}\mathcal{X}$ if and only if $\dot{x} \in \mathcal{T}_x\mathcal{X}$ for some $x \in \mathcal{X}$. The boundary of a manifold $\partial\mathcal{X}$ is a separate smooth manifold of dimension $n - 1$, with
15 its own lower dimensional tangent bundle $\mathcal{T}\partial\mathcal{X} = \coprod_{x \in \partial\mathcal{X}} \mathcal{T}_x\partial\mathcal{X}$. Likewise, $\text{int}(\mathcal{X})$ is the manifold of dimension n consisting of all interior points with tangent bundle of consistent dimension n denoted $\mathcal{T}\text{int}(\mathcal{X}) = \coprod_{x \in \text{int}(\mathcal{X})} \mathcal{T}_x\mathcal{X}$. With these definitions, the complete manifold with a boundary is understood to be the disjoint union of the separate interior and boundary manifolds $\mathcal{X} = \text{int}(\mathcal{X}) \coprod \partial\mathcal{X}$, and its tangent bundle is the disjoint union of the
20 separate tangent bundles $\mathcal{T}\mathcal{X} = \mathcal{T}\text{int}(\mathcal{X}) \coprod \mathcal{T}\partial\mathcal{X}$.

[0107] Definition 4.3. Let $\mathcal{X}$ be a manifold with boundary $\partial\mathcal{X}$ (possibly empty). A spec $(M, f)_\mathcal{X}$ is said to be interior if for any interior starting point $(x_0, \dot{x}_0) \in \mathcal{T}\text{int}(\mathcal{X})$ the integral curve $x(t)$ is everywhere interior $x(t) \in \text{int}(\mathcal{X})$ for all $t \geq 0$.

[0108] Definition 4.4. Let $\mathcal{X}$ be a manifold with a (possibly empty) boundary. A spec $\mathcal{S} =$
25 $(M, f)_\mathcal{X}$ is said to be rough if all its integral curves $x(t)$ converge: $\lim_{t \rightarrow \infty} x(t) = x_\infty$ with $x_\infty \in \mathcal{X}$ (including the possibility $x_\infty \in \partial\mathcal{X}$). If $\mathcal{S}$ is not rough, but each of its damped variants $\mathcal{S}_B = (M, f + B\dot{x})$ is, where $B(x, \dot{x})$ is smooth and positive definite, the spec is said to be frictionless. A frictionless spec's damped variants are also known as rough variants of the spec.

$(M, f)_x$ be a spec. Then $(M, f) + \partial_x \psi$ is the spec's forced variant and forcing the spec with potential ψ . Also, ψ is finite if $\|\partial_x \psi\| < \infty$ everywhere on $\mathcal{X}$.

[0110] Definition 4.6. A spec $\mathcal{S}$ forms a rough fabric if it is rough when forced by a finite potential $\psi(x)$ and each convergent point x_∞ is a Karush-Kuhn-Tucker (KKT) solution of the constrained optimization problem $\min_{x \in \mathcal{X}} \psi(x)$. A forced spec is a frictionless fabric its rough variants form rough fabrics.

[0111] Lemma 4.7. If a spec $\mathcal{S}$ forms a rough (or frictionless) fabric then (unforced) it is a rough (or frictionless) spec.

[0112] Definition 4.8. A spec $\mathcal{S} = (M, f)_x$ is boundary conforming if the following hold:

1. $\mathcal{S}$ is interior.
2. $M(x, \dot{x})$ and $f(x, \dot{x})$ are finite for all $(x, \dot{x}) \in \mathcal{T}\mathcal{X}$ (more explicitly, for all $(x, \dot{x}) \in \mathcal{T}int(\mathcal{X})$ and $(x, \dot{x}) \in \mathcal{T}\partial\mathcal{X}$).
3. The inverse metric has finite limit $M^{-1} \rightarrow M_\infty^{-1}$ with $\|M_\infty^{-1}\| < \infty$ along any boundary limiting trajectory $x \rightarrow x_\infty \in \partial\mathcal{X}$.

[0113] A boundary conforming metric is a metric satisfying conditions (2) and (3) of this definition. Additionally, f is said to be boundary conforming with respect to M if M is a boundary conforming metric and $(M, f)_x$ forms a boundary conforming spec.

[0114] Note that this definition of boundary conformance implies that M either approaches a finite matrix along trajectories limiting to the boundary or it approaches a matrix that is finite along Eigen-directions parallel with the boundary's tangent space but explodes to infinity along the direction orthogonal to the tangent space. This means that either M_∞^{-1} is full rank or it is reduced rank and its column space spans the boundary's tangent space $\mathcal{T}_{x_\infty}\partial\mathcal{X}$.

[0115] Definition 4.9. A boundary conforming spec $(M, f)_x$ is unbiased if for every convergent trajectory $x(t)$ with $x \rightarrow x_\infty$ providing $V_\infty^T f(x, \dot{x}) \rightarrow 0$ where V_∞ is a matrix whose columns contain a basis $\mathcal{V}_\infty$ for $\mathcal{T}_{x_\infty}\mathcal{X}$. If the spec is not unbiased, it is biased. The term f alone as either biased or unbiased when the context of M is clear. Note that all unbiased specs must also be boundary conforming by definition. Therefore, the spec is unbiased with the implicit understanding that it is also, by definition, boundary conforming.

full set of n linearly independent vectors, so the condition $V_\infty^T f(x, \dot{x}) \rightarrow 0$ implies $f(x, \dot{x}) \rightarrow 0$. This is not the case for $x_\infty \in \partial\mathcal{X}$.

[0117] Since the definition of unbiased is predicated on the spec being boundary

5 conforming, the spectrum of the metric M is always finite in the relevant directions (all directions for interior points and directions parallel to the boundary for boundary points). The property of being unbiased is, therefore, linked to zero acceleration within the relevant subspaces. This property is used in the following theorem to characterize general fabrics.

[0118] Theorem 4.11 (General fabrics). Suppose $\mathcal{S} = (M, f)_x$ is a boundary conforming

10 spec. Then $\mathcal{S}$ forms a rough fabric if and only if it is unbiased and converges when forced by a potential $\psi(x)$ with $\|\partial\psi\| < \infty$ on $x \in \mathcal{X}_\partial$.

[0119] Proof. The forced spec defines the equation

$$M\ddot{x} + f = -\partial_x\psi, \quad (9)$$

where here absorb the damping term into f if this is the rough variant of a frictionless spec

15 (doing so does not affect the hypotheses on f).

[0120] First assume f is unbiased. Since x converges, $\dot{x} \rightarrow 0$ which means $\ddot{x} \rightarrow 0$ as well.

[0121] If $x(t)$ converges to an interior point $x_\infty \in \text{int}(\mathcal{X})$, M is finite so $M\ddot{x} \rightarrow 0$ since $\ddot{x} \rightarrow 0$. And since the spec is unbiased $f(x, \dot{x}) \rightarrow 0$ as $\dot{x} \rightarrow 0$. Therefore, the left hand side of Equation 9 approaches 0, so $\partial_x\psi \rightarrow 0$ satisfying (unconstrained) KKT conditions.

20 [0122] Alternatively, if $x(t)$ converges to a boundary point $x_\infty \in \partial\mathcal{X}$, then analyze the expression:

$$\ddot{x} = -M^{-1}(f + \partial_x\psi) \rightarrow 0. \quad (10)$$

since $\ddot{x} \rightarrow 0$. Since M is boundary conforming, the inverse metric limit M_∞^{-1} is finite, and since $\partial\psi$ is also finite on $\mathcal{X}_\partial$, the term $M^{-1}\partial\psi$ converges to the finite vector $M_\infty^{-1}\partial\psi_\infty$.

25 Therefore, by Equation 10 that gives $M^{-1}f \rightarrow M_\infty^{-1}f_\infty = M_\infty^{-1}\partial\psi_\infty$. At the limit, M_∞^{-1} has full rank across $\mathcal{T}_{x_\infty}\partial\mathcal{X}$, so the above limit equality implies $f_\infty^{//} = -\partial\psi_\infty^{//}$, where $f_\infty^{//}$ and $\partial\psi_\infty^{//}$ are the components of f_∞ and $\partial\psi_\infty$, respectively, lying in the boundary's tangent space $\mathcal{T}_{x_\infty}\partial\mathcal{X}$. Since f is unbiased and $x_\infty \in \partial\mathcal{X}$, the boundary parallel component $f_\infty^{//} = 0$ so it must be that $\partial\psi_\infty^{//} = 0$ as well. Therefore, $\partial\psi_\infty$ must either be orthogonal to $\mathcal{T}_{x_\infty}\partial\mathcal{X}$ or zero. If it is

$-f(x, \dot{x})$ must be interior near the boundary, so $= -\partial\psi_\infty$ must be exterior facing and balancing $-f_\infty$. That orientation in addition to the orthogonality implies that the limiting point satisfies the (constrained) KKT conditions.

- 5 [0123] Finally, to prove the converse, assume f is biased. Then there exists a point $x^* \in \mathcal{X}_\partial$ for which $f(x^*, 0) \neq 0$. An objective potential with a unique global minimum at x^* can be constructed. The forced system, in this case, cannot come to rest at x^* since f is nonzero there, so (M, f) is not guaranteed to optimize and is not a fabric.

- [0124] The above theorem characterizes the most general class of fabric and shows that all
10 fabrics are necessarily unbiased in the sense of Definition 4.9. The theorem relies on hypothesizing that the system always converges when forced, and is therefore more of a template for proving a given system forms a fabric rather than a direct characterization. Proving convergence is in general nontrivial. The specific fabrics introduced below will prove convergence using energy conservation and boundedness properties.

- 15 [0125] Note that the above theorem does not place restrictions on whether or not the metric or damping is finite in Eigen-directions orthogonal to the surface's tangent space. In practice, it can be convenient to allow those metrics to raise to infinity in those directions, so the effects of forces orthogonal to the boundary's surface are increasingly damped out by the large mass. Such metrics can induce smoother optimization behavior when optimizing to
20 boundary surface local minima.

- [0126] Definition 4.12. A stationary Lagrangian $\mathcal{L}(x, \dot{x})$ is boundary conforming (on $\mathcal{X}$) if its induced equations of motion $\partial_{\dot{x}\dot{x}}^2 \mathcal{L} \ddot{x} + \partial_{\dot{x}x} \mathcal{L} \dot{x} - \partial_x \mathcal{L} = 0$ under the Euler-Lagrange equation form a boundary conforming spec $\mathcal{S}_\mathcal{L} = (M_\mathcal{L}, f_\mathcal{L})_\mathcal{X}$ where $M_\mathcal{L} = \partial_{\dot{x}\dot{x}}^2 \mathcal{L}$ and $f_\mathcal{L} = \partial_{\dot{x}x} \mathcal{L} \dot{x} - \partial_x \mathcal{L}$. This spec is known as the Lagrangian spec associated with $\mathcal{L}$. $\mathcal{L}$ is additionally
25 unbiased if $\mathcal{S}_\mathcal{L}$ is unbiased. Since unbiased specs are boundary conforming by definition, an unbiased Lagrangian $\mathcal{L}$ is implicitly boundary conforming as well.

- [0127] Definition 4.13. Let $\mathcal{L}_e(x, \dot{x})$ be a stationary Lagrangian with Hamiltonian $\mathcal{H}_e(x, \dot{x}) = \partial_{\dot{x}} \mathcal{L}_e^T x - \mathcal{L}_e$. $\mathcal{L}_e$ is an energy Lagrangian if $\mathcal{H}_e$ is nontrivial (not everywhere zero) and $\mathcal{H}_e(x, \dot{x})$ is finite on $\text{int}(\mathcal{X})$. An energy Lagrangian's equations of motion $M_e \ddot{x} + f_e = 0$ are often referred to as its energy equations with spec denoted $\mathcal{S}_e = (M_e, f_e)_\mathcal{X}$.
30

Lagrangian with energy $\mathcal{H}_e$. If for every trajectory $x(t)$ for which there exists a $t_0 < \infty$ such that $x(t_0) \in \partial\mathcal{X}$ and $x \notin \dot{T}_{x(t_0)}\partial\mathcal{X}$ (a boundary intersecting trajectory) $\lim_{t \rightarrow t_0} \mathcal{H}_e(x, \dot{x}) = \infty$ is provided, and $\mathcal{H}_e$ satisfies the boundary limiting condition.

5 [0129] Remark 4.15. When designing boundary conforming energy Lagrangians one must ensure the Lagrangian's spec is interior. To attain an interior spec, it is often helpful to design energies that prevent energy conserving trajectories from intersecting $\partial\mathcal{X}$. It can be shown that an energy Lagrangian's spec is interior if and only if its energy is boundary limiting, i.e. the energy approaches infinity for any boundary intersecting trajectory.

10 [0130] Lemma 4.16. Let $\mathcal{L}_e$ be an energy Lagrangian with energy $\mathcal{H}_e = \partial_{\dot{x}}\mathcal{L}_e^T\dot{x} - \mathcal{L}_e$. The energy time derivative is:

$$\dot{\mathcal{H}}_e = \dot{x}^T(M_e\ddot{x} + f_e), \quad (11)$$

where M_e and f_e come from the Lagrangian's equations of motion $M_e\ddot{x} + f_e = 0$.

[0131] Proof. The calculation is a straightforward time derivative of the Hamiltonian:

$$15 \quad \mathcal{H}_{\mathcal{L}_e} = \frac{d}{dt} [\partial_{\dot{x}}\mathcal{L}_e - \mathcal{L}_e] \quad (12)$$

$$= (\partial_{\dot{x}\dot{x}}\mathcal{L}_e\ddot{x} + \partial_{\dot{x}x}\mathcal{L}_e\dot{x})^T\dot{x} + \partial_{\dot{x}}\mathcal{L}_e^T\ddot{x} - (\partial_{\dot{x}}\mathcal{L}_e\ddot{x} + \partial_x\mathcal{L}_e^T\dot{x}) \quad (13)$$

$$= \dot{x}^T(\partial_{\dot{x}\dot{x}}\mathcal{L}_e\ddot{x} + \partial_{\dot{x}x}\mathcal{L}_e\dot{x} - \dot{\partial}_{\dot{x}}\mathcal{L}_e^T\dot{x}) \quad (14)$$

$$= \dot{x}^T(M_e\ddot{x} + f_e) \quad (15)$$

[0132] Lemma 4.17. Let $\mathcal{L}_e$ be an energy Lagrangian. Then $M_e\ddot{x} + f_e + f_f = 0$ is energy conserving if and only if $\dot{x}^T f_f = 0$. Such a spec $\mathcal{S} = (M_e, f_e + f_f)$ is said to be a conservative spec under energy Lagrangian $\mathcal{L}_e$.

[0133] Proof. This energy is conserved if its time derivative is zero. Substituting $\ddot{x} = -M_e^{-1}(f_e + f_f)$ into the Equation 11 of Lemma 4.16 and setting it to zero gives:

$$\mathcal{H}_{\mathcal{L}_e} = \dot{x}^T \left(M_e \left(-M_e^{-1}(f_e + f_f) \right) + f_e \right) \quad (16)$$

$$25 \quad = \dot{x}^T (-f_e - f_f + f_e) \quad (17)$$

$$= \dot{x}^T f_f = 0. \quad (18)$$

[0135] Proposition 4.18 (Conservative fabrics). Suppose $\mathcal{S} = (M_e, f_e, f_f)_x$ is a conservative unbiased spec under energy Lagrangian $\mathcal{L}_e$ with zero work term f_f . The $\mathcal{S}$ forms a frictionless fabric.

5 [0136] Proof. Let $\psi(x)$ be a lower bounded potential function with finite $\partial\psi$ on $\mathcal{X}_\partial$, and an expression for how the total $\mathcal{H}_e^\psi = \mathcal{H}_e + \psi(x)$ varies over time:

$$\dot{\mathcal{H}}_e^\psi = \dot{\mathcal{H}}_e + \dot{\psi} \quad (19)$$

$$= \dot{x}^T (M_e \ddot{x} + f_e) + \partial_x \psi^T \dot{x} \quad (20)$$

$$= \dot{x}^T (M_e \ddot{x} + f_e + \partial_x \psi). \quad (21)$$

10 [0137] With damping matrix $B(x, \dot{x})$ let $M_e \ddot{x} + f_e + f_f + \partial_x \psi + B\dot{x} = 0$ be the forced and damped variant of the conservative spec's system. Plugging that system into Equation 21 gives:

$$\dot{\mathcal{H}}_e^\psi = \dot{x}^T \left(M_e \left(-M_e^{-1} (f_e + f_f + \partial_x \psi + B\dot{x}) \right) + f_e + \partial_x \psi \right) \quad (22)$$

$$= \dot{x}^T (-f_e - \partial_x \psi - B\dot{x} + f_e + \partial_x \psi) - \dot{x}^T f_f \quad (23)$$

$$15 \quad = \dot{x}^T B\dot{x} \quad (24)$$

since all terms cancel except for the damping term. When B is strictly positive definite, the rate of change is strictly negative for $x \neq 0$. Since $\mathcal{H}_e^\psi = \mathcal{H}_e + \psi(x)$ is lower bounded and $\dot{\mathcal{H}}_e^\psi \leq 0$, and $\dot{\mathcal{H}}_e^\psi \rightarrow 0$ which implies $\dot{x} \rightarrow 0$ and the system is guaranteed to converge.

20 Therefore, since it is additionally boundary conforming and unbiased, by Theorem 4.11 it forms a fabric.

[0138] Finally, when $B = 0$ Equation 22 shows that total energy is conserved, so a system that starts with nonzero energy cannot converge. Therefore, the undamped system is a frictionless fabrics with rough variants defined by the added damping term.

[0139] Corollary 4.19 (Lagrangian and Finsler fabrics). If $\mathcal{L}_e(x, \dot{x})$ is an unbiased energy
25 Lagrangian, then $\mathcal{S}_e = (M_e, f_e)_x$ forms a frictionless fabric known as a Lagrangian fabric. When $\mathcal{L}_e$ is a Finsler energy, this fabric is known more specifically as a Finsler fabric.

additionally unbiased, by Proposition 4.18 it forms a frictionless fabric.

[0141] The following lemma collects some results around common matrices and operators that arise when analyzing energy conservation.

5 [0142] Lemma 4.20. Let $\mathcal{L}_e$ be an energy Lagrangian. Then with $p_e = M_e \dot{x}$

$$R_{p_e} = M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \quad (25)$$

has nullspace spanned by p_e and

$$R_{\dot{x}} = M_e - \frac{p_e p_e^T}{p_e^T M_e^{-1} p_e} \quad (26)$$

has nullspace spanned by $\dot{x}$. These matrices are related by $R_{\dot{x}} = M_e R_{p_e} M_e$ and the matrix

10 $M_e^{-1} R_{\dot{x}} = M_e R_{p_e} = P_e$ is a projection operator of the form

$$P_e = M_e^{\frac{1}{2}} [I - \hat{v} \hat{v}^T] M_e^{\frac{1}{2}} \quad (27)$$

where $v = M_e^{\frac{1}{2}} \dot{x}$ and $\hat{v} = \frac{v}{\|v\|}$ is the normalized vector. Moreover, $\dot{x}^T P_e f = 0$ for all $f(x, \dot{x})$

[0143] Proof. Right multiplication of R_{p_e} by p_e gives:

$$R_{p_e} p_e = \left(M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right) M_e \dot{x} \quad (28)$$

$$15 \quad = \dot{x} - \dot{x} \left(\frac{\dot{x}^T M_e \dot{x}}{\dot{x}^T M_e \dot{x}} \right) = 0 \quad (29)$$

and the nullspace is no larger than that since each matrix is formed by subtracting off a rank 1 term from a full rank matrix.

[0144] The relation between R_{p_e} and $R_{\dot{x}}$ can be shown algebraically:

$$M_e R_{p_e} M_e = M_e \left(M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right) M_e = M_e - \frac{M_e \dot{x} \dot{x}^T M_e}{\dot{x}^T M_e \dot{x}} \quad (30)$$

$$20 \quad = M_e - \frac{p_e p_e^T}{p_e^T M_e^{-1} p_e} = R_{\dot{x}} \quad (31)$$

[0145] Therefore, $R_{\dot{x}}$ has the same rank as R_{p_e} and its nullspace must be spanned by $\dot{x}$

since $R_{\dot{x}} \dot{x} = M_e R_{p_e} M_e \dot{x} = M_e R_{p_e} p_e = 0$.

$$M_e R_{p_e} = M_e \left(M_e^{-1} - \frac{\dot{x} \dot{x}^T}{\dot{x}^T M_e \dot{x}} \right) \quad (32)$$

$$= M_e^{\frac{1}{2}} \left(I - \frac{M_e^{\frac{1}{2}} \dot{x} \dot{x}^T M_e^{\frac{1}{2}}}{\dot{x}^T M_e^{\frac{1}{2}} M_e^{\frac{1}{2}} \dot{x}} \right) M_e^{-\frac{1}{2}} \quad (33)$$

$$= M_e^{\frac{1}{2}} \left(I - \frac{v v^T}{v^T v} \right) M_e^{-\frac{1}{2}} \quad (34)$$

$$5 \quad = P_e \quad (35)$$

since $\frac{v v^T}{v^T v} = \hat{v} \hat{v}^T$. Moreover,

$$P_e P_e = M_e^{\frac{1}{2}} \left(I - \frac{v v^T}{v^T v} \right) M_e^{-\frac{1}{2}} M_e^{\frac{1}{2}} \left(I - \frac{v v^T}{v^T v} \right) M_e^{-\frac{1}{2}} \quad (36)$$

$$= M_e^{\frac{1}{2}} \left(I - \frac{v v^T}{v^T v} \right) \left(I - \frac{v v^T}{v^T v} \right) M_e^{-\frac{1}{2}} \quad (37)$$

$$= M_e^{\frac{1}{2}} (I - \hat{v} \hat{v}^T) M_e^{-\frac{1}{2}} = P_e, \quad (38)$$

10 since $P_{\perp} = I - \hat{v} \hat{v}^T$ is an orthogonal projection operator. Therefore, $P_e^2 = P_e$ showing that it is a projection operator as well.

[0147] Finally, the following:

$$\dot{x}^T P_e = \dot{x}^T M_e R_{p_e} = \dot{x}^T M_e \left(M_e^{-1} - \frac{\dot{x} \dot{x}^T}{\dot{x}^T M_e \dot{x}} \right) \quad (39)$$

$$= \left[\dot{x}^T - \left(\frac{\dot{x}^T M_e \dot{x}}{\dot{x}^T M_e \dot{x}} \right) \dot{x}^T \right] = 0 \quad (40)$$

15 [0148] Therefore, for any f , $M_e \ddot{x} + f_e + f_f = 0$.

[0149] Corollary 4.21. Let $\mathcal{L}_e$ be an energy Lagrangian. Then for any $\tilde{f}_f(x, \dot{x})$, under the forcing term $f_f = P_e \tilde{f}_f$ the equations of motion:

$$M_e \ddot{x} + f_e + f_f = 0 \quad (41)$$

are energy conserving.

energy conserving.

[0151] Proposition 4.22 (System energization). Let $\ddot{x} + h(x, \dot{x}) = 0$ be a differential equation, and suppose $\mathcal{L}_e$ is any energy Lagrangian with equations of motion $M_e \ddot{x} + f_e = 0$ and energy $\mathcal{H}_e$. Then $\ddot{x} + h(x, \dot{x}) + \alpha_{\mathcal{H}_e} \dot{x} = 0$ where

$$\alpha_{\mathcal{H}_e} = -(\dot{x}^T M_e \dot{x})^{-1} \dot{x}^T [M_e h - f_e] \quad (42)$$

is energy conserving and differs from the original system by only an acceleration along the direction of motion. The new system can be expressed as:

$$M_e \ddot{x} + f_e + P_e [M_e h - f_e] = 0 \quad (43)$$

10 [0152] This system modification is known as the energization transform and is denoted using spec notation as

$$\mathcal{S}_h^{\mathcal{L}_e} = (M_e, f_e + P_e [M_e h - f_e])_x = \text{energize}_{\mathcal{L}_e} \{\mathcal{S}_h\} \quad (44)$$

where $\mathcal{S}_h = (I, h)_x$ is the spec representing $\ddot{x} + h(x, \dot{x}) = 0$.

[0153] Proof. Equation 11 of Lemma 4.16 gives the time derivative of the energy.

15 Substituting a system of the form $\ddot{x} + h(x, \dot{x}) = \alpha_{\mathcal{H}_e} \dot{x}$, setting it to zero, and solving for $\alpha_{\mathcal{H}_e}$ gives

$$\dot{\mathcal{H}}_e = \dot{x}^T [M_e (-h - \alpha_{\mathcal{H}_e} \dot{x}) + f] = 0 \quad (45)$$

$$\Rightarrow -\dot{x}^T M_e h - \alpha_{\mathcal{H}_e} \dot{x}^T M_e \dot{x} + \dot{x}^T f_e = 0 \quad (46)$$

$$\Rightarrow \alpha_{\mathcal{H}_e} = -\frac{\dot{x}^T M_e h - \dot{x}^T f_e}{\dot{x}^T M_e \dot{x}} \quad (47)$$

$$20 \quad = (-\dot{x}^T M_e \dot{x})^{-1} \dot{x}^T [M_e h - f_e]. \quad (48)$$

[0154] Substituting this solution for $\alpha_{\mathcal{H}_e}$ back in gives:

$$\ddot{x} = -h + \left(\frac{\dot{x}^T}{\dot{x}^T M_e \dot{x}} [M_e h - f_e] \right) \dot{x} \quad (49)$$

$$= -h + \left[\frac{\dot{x} \dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (M_e h - f_e). \quad (50)$$

difference away from f_e first. Doing so and moving all the terms to the left hand side of the equation gives

$$\ddot{x} + h - \left[\frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (M_e h - f_e) = 0 \quad (51)$$

$$5 \quad \Rightarrow \ddot{x} + M_e^{-1} [f_f + f_e] - \left[\frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (M_e M_e^{-1} [f_f + f_e] - f_e) = 0 \quad (52)$$

$$\Rightarrow M_e \ddot{x} + f_f + f_e - M_e \left[\frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (f_f + (f_e - f_e)) \quad (53)$$

$$\Rightarrow M_e \ddot{x} + f_e + M_e \left[M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] f_f \quad (54)$$

$$\Rightarrow M_e \ddot{x} + f_e + M_e R_{p_e} (M_e h - f_e), \quad (55)$$

where $f_f = M_e f_f - f_e$ is substituted back in back in. By Lemma 4.20 $M_e R_{p_e} = P_e$, to get

10 Equation 43.

[0156] Definition 4.23. A metric $M(x, \dot{x})$ is said to be boundary aligned if for any convergent $x(t)$ with $x_\infty \in \partial\mathcal{X}$ the limit $\lim_{t \rightarrow \infty} M^{-1}(x, \dot{x}) = M_\infty^{-1}$ exists and is finite, $\mathcal{T}_{x_\infty} \partial\mathcal{X}$ is spanned by a subset of Eigen-basis of M_∞^{-1} .

[0157] Lemma 4.24. Suppose M_e is boundary conforming. Then if (M_e, f) and (M_e, g) are
15 both unbiased, then $(M_e, \alpha f, \beta g)$ is unbiased.

[0158] Proof. Suppose $x(t)$ is a convergent trajectory with $x \rightarrow x_\infty$. If $x_\infty \in \text{int}(\mathcal{X})$, then $f \rightarrow 0$ and $g \rightarrow 0$, so $\alpha f + \beta g \rightarrow 0$. If $x_\infty \in \partial\mathcal{X}$, then $f = f_1 + f_2$ with $f_1 \rightarrow 0$ and $f_2 \rightarrow f_\perp \perp \mathcal{T}_{x_\infty} \partial\mathcal{X}$, and similarly with $g = g_1 + g_2$ with $g_1 \rightarrow 0$ and $g_2 \rightarrow g_\perp \perp \mathcal{T}_{x_\infty} \partial\mathcal{X}$. Then $\alpha f + \beta g = (\alpha f_1 + \beta f_2) + (\alpha g_1 + \beta g_2) \rightarrow \alpha g_1 + \beta g_2$ which is orthogonal to $\mathcal{T}_{x_\infty} \partial\mathcal{X}$ since
20 each component in the linear combination is. Therefore, $\alpha f + \beta g$ is unbiased.

[0159] Lemma 4.25. For every matrix A, there exists a constant $c > 0$ such that $\|Au\| \leq c\|u\|$ for every vector u, where $\|\cdot\|$ can be any norm.

[0160] In the below, for concreteness, take the matrix norm to be the Frobenius norm.

[0161] Lemma 4.26. Let M_e be boundary conforming. Then if f is unbiased with respect to
25 $M_e, P_e f$ is unbiased, where $P_e = M_e^{-\frac{1}{2}} [I - \hat{v}\hat{v}^T] M_e^{-\frac{1}{2}}$ is the projection operator defined in Equation 27.

is finite on $\text{int}(\mathcal{X})$, by Lemma 4.25 there exists a constant $c > 0$ such that $\|P_e f\| \leq c\|f\|$. Since $\|f\| \rightarrow 0$ as $t \rightarrow 0$, it must be that $\|P_e f\| \rightarrow 0$.

[0163] Consider now the case where $x_\infty \in \partial\mathcal{X}$. Since P_e is a projection operator, for every z there must be a decomposition into linearly independent components $z = z_1 + z_2$ with z_2 in the kernel such that $P_e z = P_e z_1 + P_e z_2 = z_1$ (so that $P_e^2 z = P_e z_1 z_1 + P_e z$). Thus, if z has a property if and only if all elements of a decomposition have that property, then $P_e z = z_1$ must have that property as well. Specifically, if z lies in a subspace (respectively, lies orthogonal to a subspace) then $P_e z = z_1$ lies in that subspace as well (respectively, lies orthogonal to a subspace).

[0164] The property of being unbiased is defined by the behavior of the decomposition $f = f^{//} + f^\perp$, where those components are respectively parallel and perpendicular to $\mathcal{T}_{x_\infty} \partial\mathcal{X}$ in the limit. Following the above outlined subscript convention to characterize the behavior of the projection, the following are provided:

$$P_e f = P_e(f^{//} + f^\perp) = P_e f^{//} + P_e f^\perp \quad (56)$$

$$= f_1^{//} + f_1^\perp \rightarrow f_1^\perp \quad (57)$$

since f being unbiased implies $f^{//} \rightarrow 0$ and hence $f_1^{//} \rightarrow 0$.

[0165] Therefore, in both cases, $P_e f$ satisfies the conditions of being unbiased if f does.

[0166] Lemma 4.27. Suppose $M_e(x, \dot{x})$ is boundary conforming and boundary aligned. If (I, h) is unbiased then $(M_e, M_e h)$ is unbiased.

[0167] Proof. Let $x(t)$ be convergent with $x \rightarrow x_\infty$ as $t \rightarrow \infty$. If $x_\infty \in \text{int}(\mathcal{X})$, then since M_e is finite on $\mathcal{T}\text{int}(\mathcal{X})$, by Lemma 4.25 there exists a constant $c > 0$ such that $\|M_e h\| \leq c\|h\|$. Since $\|h\| \rightarrow 0$ as $t \rightarrow 0$, it must be that $\|M_e h\| \rightarrow 0$.

[0168] Since M_e is boundary aligned, there is a subset of Eigenvectors that span the tangent space in the limit as $x(t) \rightarrow x_\infty \in \partial\mathcal{X}$. Let $V_{//}$ denote a matrix containing the Eigenvectors that limit to spanning the tangent space, with $D_{//}$ a diagonal matrix containing the corresponding Eigenvalues. Likewise, let $V_\perp$ contain the remaining (perpendicular in the limit) Eigenvectors, with Eigenvalues $D_\perp$. The metric decomposes as $M_e = M_e^{//} + M_e^\perp$ with

$y_{\perp}$ are coefficients, and $y_{//} \rightarrow 0$ as $t \rightarrow \infty$. Therefore,

$$M_e h = (M_e^{//} + M_e^{\perp})(h^{//} + h^{\perp}) \quad (58)$$

$$= (V_{//} D_{//} V_{//}^T + V_{\perp} D_{\perp} V_{\perp}^T)(V_{//y//} + V_{\perp y\perp}) \quad (59)$$

$$5 \quad = V_{//} D_{//y//} + V_{\perp} D_{\perp y\perp} \quad (60)$$

$$\rightarrow V_{\perp} z \quad \perp \quad T_{x_{\infty}} \partial \mathcal{X} \quad \text{where } z = D_{\perp y\perp} \quad (61)$$

[0169] Therefore, when $x_{\infty} \in \partial \mathcal{X}$, the component parallel to the tangent space vanishes in the limit.

[0170] Lemma 4.28. Suppose $\mathcal{S}_h = (I, h)$ is an unbiased (acceleration) spec and $\mathcal{L}_e$ is an
10 unbiased energy Lagrangian with boundary aligned metric M_e . Then $\mathcal{S}_h^{\mathcal{L}_e} = \text{energize}_{\mathcal{L}_e}\{\mathcal{S}_h\}$ is unbiased.

[0171] Proof. M_e is boundary conforming by hypothesis on $\mathcal{L}_e$. Moreover, the energized equation takes the form

$$M_e \ddot{x} + f_e + P_e[M_e h - f_e] = 0 \quad (62)$$

15 as shown in Proposition 4.22. By Lemma 4.27 $M_e h$ is unbiased since h is unbiased, and likewise $M_e h - f_e$ is unbiased by Lemma 4.24 since f_e is unbiased by hypothesis on $\mathcal{L}_e$. That means $P_e[M_e h - f_e]$ is unbiased as well by Lemma 4.26. Finally, by again applying Lemma 4.24 it is shown that the entirety of $f_e + P_e[M_e h - f_e]$ is unbiased.

[0172] Theorem 4.29 (Energized fabrics). Let $\mathcal{L}_e$ be an unbiased energy Lagrangian with
20 boundary aligned $M_e = \partial_{\dot{x}\dot{x}}^2 \mathcal{L}_e$ and lower bounded energy $\mathcal{H}_e$, and let (I, h) be an unbiased spec. Then the energized spec $\mathcal{S}_h^{\mathcal{L}_e} = \text{energize}_{\mathcal{L}_e}\{\mathcal{S}_h\}$ given by Proposition 4.22 forms a frictionless fabric.

[0173] The following shows that Finsler geometries form conservative fabrics when the
underlying system is energy conserving, and in particular a broad class of unbiased systems
25 can be energized to turn them into fabrics. However, in general the energization transform affects the behavior of the system since systems generally are not invariant to the speed of traversal. Specifically, the system follows different paths when forced to slow down or speed up. An intuitive example of such a system is a particle in a gravitational field. When traveling

down along the direction of motion it will either break orbit or spiral into the mass; in both cases, the path will necessarily change.

[0174] In this section, a special class of differential equations are introduced called geometry generators that maintain geometric path consistency despite the speed of traversal. For this special class of equation, the path behavior of a system is invariant under energization, so an energized fabric formed by energizing a geometry generator follows the same path as the original system, and is itself a geometry generator. These energized fabrics are called geometric fabrics.

10 [0175] Definition 5.1 (Geometry generator). A geometry generator, or generator for short, is a second-order differential equation of the form:

$$\ddot{x} + h(x, \dot{x}) = 0 \quad (63)$$

where $h_2(x, \dot{x})$ is a smooth, covariant, map $h_2 : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^d$ that is positively homogeneous of degree 2 in velocities in the sense $h_2(x, \alpha \dot{x}) = \alpha^2 h_2(x, \dot{x})$ for $\alpha > 0$.

15 Solutions to the generator are called generating solutions or trajectories, and if those trajectories are guaranteed to conserve a known energy quantity, they are known as energy levels.

[0176] As a second-order differential equation, a generator's solutions are unique for specific initial values $(x_0, \dot{x}_0)$. A generator's degree 2 homogeneity means that all solutions to initial value problems of the form $(x_0, \alpha \hat{v}_0)$, where $\hat{v}_0$ is any unit vector defining a direction in space, follow the same path. Said another way, all generating trajectories starting from a given point x_0 with initial velocity pointing in the same direction $\hat{x}_0 = \alpha \hat{v}_0$ follow the same path.

25 [0177] Generators are often called sprays in differential geometry, although the term generator is more explicit about its role generating the geometry of a geometric equation, as defined next.

[0178] Definition 5.2 (Geometric equation). The geometric equation corresponding to a generator of the form $\ddot{x} + h_2(x, \dot{x}) = 0$ is an equation of the form

$$P_{\dot{x}}^{\perp}[\ddot{x} + h_2(x, \dot{x})] = 0 \quad (64)$$

known as geometric solutions or trajectories.

[0179] Any matrix $R_{\dot{x}}$ with nullspace spanned by $\dot{x}$ would suffice in this definition, the projector is chosen for clarity of its role.

5 [0180] While the generator's solutions are unique but follow the same path when the initial conditions' velocities point in the same direction, the geometric equation is a redundant equation (redundancy coming from the reduced rank matrix $P_{\dot{x}}^{\perp}$) whose solutions are the set of all trajectories following that single path. It can be shown that every solution to the generator equation is a smooth time-reparameterization of any generating trajectory.

10 [0181] Moreover, for a given geometric trajectory there exists a generating trajectory whose instantaneous velocity matches at a given point x , so the geometric trajectory can be viewed as speeding up and slowing down along the direction of motion to smoothly move between generating solutions, hence the name generating solution. When these generating solutions form energy levels, the geometric solution speeds up and slows down along the
15 direction of motion to smoothly move between energy levels of the system.

[0182] The geometric equation fully characterizes a geometry of paths. The equivalence class of solutions to problems $x_0, \alpha \hat{v}_0$ for $\alpha > 0$ are (locally) the set of reparameterizations of a one-dimensional smooth submanifold of the space. The collection of these speed-independent paths defines a nonlinear geometry on the space.

20 [0183] All solutions of the nullspace geometric equation given in Equation 64 can be expressed as:

$$\ddot{x} = h_2(x, \dot{x}) + \gamma(t)\dot{x} \quad (65)$$

where $\gamma(t)$ is any smooth function of time. This is an explicit expression showing that geometric solutions are formed by speeding up and slowing down along the direction of
25 motion $\dot{x}$.

[0184] Finsler geometry is the study of nonlinear geometries whose geometric equation is defined by the equations of motion of Finsler structure.

[0185] Definition 5.3 (Finsler structure). A Finsler structure is a stationary Lagrangian $\mathcal{L}_g(x, \dot{x})$ with the following properties:

2. Homogeneity: $\mathcal{L}_g$ is positively homogeneous of degree 1 in velocities in the sense $\mathcal{L}_g(\dot{x}, \alpha \dot{x}) = \alpha \mathcal{L}_g(\dot{x}, \dot{x})$ for $\alpha > 0$.

3. Energy tensor invertibility: $\partial_{\dot{x}\dot{x}}^2 \mathcal{L}_e$ is everywhere invertible, where $\mathcal{L}_e = \frac{1}{2} \mathcal{L}_g^2$.

5 $\mathcal{L}_e$ is known as the energy form of $\mathcal{L}_g$, and is sometimes called the Finsler energy.

[0186] Note that the first two conditions together mean that $\mathcal{L}(x, 0) = 0$.

[0187] Finsler structures might be more descriptively termed geometric Lagrangians since their geometric properties (invariance to time-reparameterization) stem directly from conditions on the Lagrangian and their effect on the resulting action.

10 [0188] Proposition 5.4 (Energy of a Finsler geometry). Let $\mathcal{L}_e = \frac{1}{2} \mathcal{L}_g^2$ be the Finsler energy of Finsler structure $\mathcal{L}_g$. The Hamiltonian (conserved quantity) of $\mathcal{L}_e$ is $\mathcal{L}_e$. Specifically,

$$\mathcal{H}_e = \partial_{\dot{x}} \mathcal{L}_e^T \dot{x} - \mathcal{L}_e = \mathcal{L}_e. \quad (66)$$

[0189] Proof. By Euler's theorem on homogeneous functions, if $f(y)$ is homogeneous of degree k , then $\partial_y f^T y = k f(y)$. That means for Lagrangians $\mathcal{L}(x, \dot{x})$, thus

$$15 \quad \partial_{\dot{x}} \mathcal{L}^T \dot{x} = k \mathcal{L}, \quad (67)$$

which means

$$\mathcal{H} = \partial_{\dot{x}} \mathcal{L}^T \dot{x} - \mathcal{L} = k \mathcal{L} - \mathcal{L} = (k - 1) \mathcal{L} \quad (68)$$

[0190] Since $\mathcal{L}_g$ is homogeneous of degree 1 in $\dot{x}$, $\mathcal{L}_e$ is homogeneous of degree 2 in $\dot{x}$, so for $\mathcal{L}_e$ the above analysis means $\mathcal{H}_e = (2 - 1) \mathcal{L}_e = \mathcal{L}_e$.

20 [0191] Equivalent forms of higher order energy can be defined as well via $\mathcal{L}_e^k = \frac{1}{k} \mathcal{L}_g^k$ and all of the results below also hold, but treat only $k = 2$ here for clarity of exposition.

[0192] Lemma 5.5 (Homogeneity of the Finsler energy tensor). Let $\mathcal{L}_g$ be a Finsler structure with energy form $\mathcal{L}_e = \frac{1}{2} \mathcal{L}_g^2$. Then the energy tensor $M_e = \partial_{\dot{x}\dot{x}}^2 \mathcal{L}_e$ is homogeneous of degree 0.

25 [0193] The above lemma means that M_e is dependent on velocity $\dot{x}$ only through its norm $\hat{\dot{x}}$, i.e. rescaling $\dot{x}$ does not affect the energy tensor.

energy $\mathcal{L}_e = \frac{1}{2}\mathcal{L}_g^2$. The equations of motion of $\mathcal{L}_e$ define a geometry generator whose geometric equation is given by the equations of motion of $\mathcal{L}_g$.

[0195] These results mean that Finsler structures $\mathcal{L}_g$ define nonlinear geometries of paths
5 whose generators define energy levels of the energy $\mathcal{L}_e$.

[0196] The expression in Equation 43 shows that energization can be viewed as a zero work modification to the energy equations of motion. When the original differential equation $\ddot{x} + h = 0$ is a geometry generator (i.e. h is homogeneous of degree 2) and $\mathcal{L}_e$ is Finsler, then the Finsler equations of motion, the original differential equation, and energized equation are
10 all geometry generators, and importantly the energized equation in 43 generates a geometry equivalent to the original equation's generated geometry. Therefore, view this zero work modification as bending the Finsler geometry to match the desired geometry without affecting the system energy. This result is summarized in the following proposition.

[0197] Corollary 5.7 (Bent Finsler Representation). Suppose $h_2(x, \dot{x})$ is homogeneous of
15 degree 2 so that $\ddot{x} + h_2(x, \dot{x}) = 0$ is a geometry generator, and let $\mathcal{L}_e$ be a Finsler structure (and therefore also a Finsler energy). Then the energized system $M_e\ddot{x} + f_e + P_e[M_e h_2 - f_e] = 0$ is a geometry generator whose geometry matches the original system's geometry. Since the Finsler system $M_e\ddot{x} + f_e = 0$ is a geometry generator as well, view the energized system as a zero work geometric modification to the Finsler geometry, referred to herein as a
20 bending of the geometric system.

[0198] Proof. The energized system takes the form $\ddot{x} + \tilde{h}_2(x, \dot{x}) = 0$ where

$$\tilde{h}_2 = M_e^{-1}f_e + R_e[M_e h_2 - f_e] \quad (69)$$

since $P_e = M_e R_{p_e}$. The energy $\mathcal{L}_e$ is Finsler, so f_e is homogeneous of degree 2 and M_e is homogeneous of degree 0, which means the first term in combination is homogeneous of
25 degree 2. Moreover, $R_{p_e} = M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}}$ is homogeneous of degree 0 since the numerator and denominator scalars would cancel in the second term when $\dot{x}$ is scaled. Therefore, the energized system in its entirety forms a geometry generator.

direction of motion $\dot{x}$ do not change the paths taken by the system. So $\ddot{x} + \tilde{h}_2(x, \dot{x}) = 0$ forms a generator whose geometry matches the original geometry defined by $\ddot{x} + h_2(x, \dot{x}) = 0$.

[0199] Corollary 5.7 shows that geometries are invariant under energization transforms per-
5 formed with respect to Finsler energies.

[0200] Note that for the energized system to be a generator, the original system is to be a generator, and also the energy should be Finsler. If the energy is not Finsler, the resulting energized system will still follow the same paths as the original geometry (since by definition it is formed by accelerating along the direction of motion), but it will not be itself a generator
10 (i.e. the paths will not be invariant to reparameterization).

[0201] Corollary 5.8 (Geometric fabrics). Suppose $h_2(x, \dot{x})$ is homogeneous of degree 2 so that $\ddot{x} + h_2(x, \dot{x}) = 0$ is a geometry generator, and suppose $\mathcal{L}_e$ is Finsler. Then the energized system is fabric defined by a generator whose geometry matches the original generator's geometry. Such a fabric is called a geometric fabric.

15 [0202] Proof. By Corollary 5.7 the energized system is a generator with matching geometry, and by Theorem 4.29 that energized system forms a fabric.

[0203] Each class of fabric described above is closed under the spec algebra. Specifically, specs forming a fabrics of a particular class remain fabrics of the same class under the spec algebra operations of combination and pullback. This result is summarized by the following
20 theorem.

[0204] Theorem 6.1. The following classes of fabrics are closed under the spec algebra: optimization fabrics, conservative fabrics, Finsler energized fabrics, geometric fabrics. A fabric of each of these types will remain a fabric of the same type under spec algebra operations in regions where the differentiable transforms remain finite.

25 [0205] Above in Theorem 6.1 shows that fabrics behave naturally under spec algebra operations in the sense that each of the above described classes is closed under these operations. Here it is shown additionally that the operation of energization outlined in Theorem 4.22 commutes with the pullback operator as long as the pullback is performed with respect to the metric defined by the energy used for energization.

defined, establish the Euler-Lagrange equation $M_e \ddot{x} + f_e = 0$ in χ and pull it back to Q to get $(J^T M_e J) \ddot{q} + J^T (f_e - \dot{J} \dot{q}) = 0$, or pull the Lagrangian back to Q to get $\tilde{\mathcal{L}}_e(q, \dot{q}) = \mathcal{L}_e(\phi(q), J\dot{q})$ first and apply the Euler-Lagrange equation direction to that pullback Lagrangian to get

5 $\tilde{M}_e \ddot{q} + \tilde{f}_e = 0$, and the resulting equations of motion will be the same with $\tilde{M}_e (J^T M_e J)$ and $\tilde{f}_e = J^T (f_e - \dot{J} \dot{q})$. This standard result shows that the operation of deriving the Euler-Lagrange equation from a Lagrangian commutes with the pullback transform. Thus, the system can either apply the Euler-Lagrange equation in the co-domain (ambient space) and pull back the resulting equations of motion, or pull back the Lagrangian to the domain and

10 directly apply the Euler-Lagrange equation there. The resulting equations will match.

[0207] The result presented in Theorem 7.1 shows that the same type of commutivity holds for the energization operation as well. This result is specific to the case where the differentiable map defines an embedding, and the intuition comes from understanding that case as well. An example is where the differentiable map is a full-rank map from a d -

15 dimensional space of generalized coordinates Q into a higher-dimensional ambient space χ of dimension $n > d$. The embedded manifold may be viewed as a constraint, and the coordinates Q define generalized coordinates for that constraint. The theorem states that if there is an energy Lagrangian defined on the ambient space along with some differential equation $\ddot{x} + h(x, \dot{x}) = 0$, the system can either energize the ambient space and pullback the

20 resulting equations or first pull back the differential equation with respect to the energy Lagrangian's metric and energize the equation there. The theorem shows that when the energization operation is used to define a fabric the most fundamental element is the energy metric. The energy defines how the pullback of the differential equation must be performed in order to remain consistent with the energization operation.

25 [0208] In practice the system does not need to explicitly energize. Instead, the system can simply pull the differential equation back to the root with respect to the energy metric and keep track of the energy itself to define a lower bound on the damping required to maintain stability while instead controlling an alternative execution energy. The energy Lagrangian's primary role in shaping the equations of motion is to define the metric. Theorem 7.2 shows

30 that this basic theorem allows us to view energization as inducing metric weighted averages of acceleration policies in different task spaces.

order differential equation with associated natural form $\text{spec}(M_e, f)$ under metric $M_e = \partial_{\dot{x}\dot{x}}^2 \mathcal{L}_e$ where $f = M_e h$. Suppose $x = \phi(q)$ is a differentiable map for which the pullback metric $J^T M_e J$ is full rank. Then

$$5 \quad \text{energize}_{\text{pull}_{\mathcal{L}_e}}(\text{pull}_{\phi}(M_e, f_2)) = \text{pull}_{\phi}(\text{energize}_{\mathcal{L}_e}(M_e, f_2)). \quad (70)$$

[0210] The energization operation commutes with the pullback transform.

[0211] Proof. The equivalence by calculation is shown. The energization of $\ddot{x} + h = 0$ in force form $M_e \ddot{x} + f = 0$ with $f = M_e h$ is $M_e \ddot{x} + f_e^h$ where:

$$f_e^h + f_e + M_e \left[M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] \quad (71)$$

10 where $f_e = \partial_{\dot{x}\dot{x}} \mathcal{L}_e \dot{x} - \partial_x \mathcal{L}_e$ so that $M_e \ddot{x} + f_e^h$ is the energy equation. Let $J = \partial_x \phi$. The pullback of the energized geometry generator is:

$$J^T M_e (J\ddot{q} + \dot{J}\dot{q}) + J^T f_e^h = 0 \quad (72)$$

$$\Rightarrow (J^T M_e J)\ddot{q} + J^T (f_e^h + M_e \dot{J}\dot{q}) = 0 \quad (73)$$

$$\Rightarrow (J^T M_e J)\ddot{q} + J^T f_e + J^T M_e \left[M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (f - f_e) + J^T M_e \dot{J}\dot{q} = 0 \quad (74)$$

$$15 \quad \Rightarrow \tilde{M}_e \ddot{q} + \tilde{f}_e + J^T M_e \left[M_e^{-1} - \frac{\dot{x}\dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (f - f_e), \quad (75)$$

Where $\tilde{M}_e = J^T M_e J$ and $\tilde{f}_e = J^T (f_e + M_e \dot{J}\dot{q})$ form the standard pullback of (M_e, f_e) .

[0212] Now calculate the geometry pullback with respect to the energy metric M_e by pulling back the metric weighted force form of the geometry $M_e \ddot{x} + f = 0$, where again $f = M_e h$. The pullback is:

$$20 \quad J^T M_e (J\ddot{q} + \dot{J}\dot{q}) + J^T f = 0 \quad (76)$$

$$\Rightarrow (J^T M_e J)\ddot{q} + J^T (f + M_e \dot{J}\dot{q}) \quad (77)$$

$$\Leftrightarrow \tilde{M}_e \ddot{q} + \tilde{f} = 0 \quad (78)$$

where $\tilde{M}_e = J^T M_e J$ as before and $\tilde{f} = J^T (f + M_e \dot{J}\dot{q})$.

Lagrange equation commutes with the pullback, so applying the Euler-Lagrange equation to this pullback energy $\tilde{\mathcal{L}}_e$ is equivalent to pulling back the Euler-Lagrange equation of $\mathcal{L}_e$.

Thus, calculate the Euler-Lagrange equation of $\mathcal{L}_e$ as:

$$5 \quad (J^T M_e J) \ddot{q} + J^T (f_e + M_e \dot{J} \dot{q}) = 0 \quad (79)$$

$$\Leftrightarrow \tilde{M}_e \ddot{q} + \tilde{f}_e = 0 \quad (80)$$

with $\tilde{M}_e = J^T M_e J$ and $\tilde{f}_e = J^T (f_e + M_e \dot{J} \dot{q})$ (both as previously defined). Therefore, energizing 78 with $\tilde{\mathcal{L}}_e$ gives

$$\tilde{M}_e \ddot{q} + \tilde{f}_e + \tilde{M}_e \left[\tilde{M}_e^{-1} - \frac{\dot{q} \dot{q}^T}{\dot{q}^T \tilde{M}_e \dot{q}} \right] (\tilde{f} - \tilde{f}_e) = 0 \quad (81)$$

$$10 \quad \Rightarrow \tilde{M}_e \ddot{q} + \tilde{f}_e + (J^T M_e J) \left[\tilde{M}_e^{-1} - \frac{\dot{q} \dot{q}^T}{\dot{q}^T J^T M_e J \dot{q}} \right] (J^T (f - M_e \dot{J} \dot{q}) - J^T (f_e + M_e \dot{J} \dot{q})) = 0 \quad (82)$$

$$\Rightarrow \tilde{M}_e \ddot{q} + \tilde{f}_e + (J^T M_e J) \left[(J^T M_e J)^{-1} - \frac{\dot{q} \dot{q}^T}{\dot{x}^T M_e \dot{x}} \right] J^T (f - f_e) = 0 \quad (83)$$

$$\Rightarrow \tilde{M}_e \ddot{q} + \tilde{f}_e + J^T M_e \left[J (J^T M_e J)^{-1} J^T - \frac{\dot{x} \dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (f - f_e) = 0 \quad (84)$$

Since

$$15 \quad J^T M_e J (J^T M_e J)^{-1} J^T = J^T = J^T M_e (M_e^{-1}), \quad (85)$$

the Equation 84 as can be expressed by

$$\tilde{M}_e \ddot{q} + \tilde{f}_e + J^T M_e \left[M_e^{-1} - \frac{\dot{x} \dot{x}^T}{\dot{x}^T M_e \dot{x}} \right] (f - f_e) = 0 \quad (86)$$

which matches the expression for the energized geometry pullback in Equation 75.

[0214] Theorem 7.1 shows that one concise way to compute the energized geometry in the root is to first energize the leaves and then perform standard pullbacks. However, it is equally valid to simply pullback the geometries with respect to the energy metrics and energize the result. The following proposition shows that the system can view such a pullback geometry as a metric weighted average of geometries.

[0215] Proposition 7.2 (Metric weighted average of geometries.). Let $x_i = \emptyset(q)$ for $i = 1, \dots, m$ denote the star-shaped reduction of any transform tree, and suppose the leaves are

$\partial_{\dot{x}\dot{x}}^2 \mathcal{L}_{e_i}$. Then the metric weighted pullback of the full leaf geometry is $\ddot{q} + \tilde{h}_2 = 0$, with

$$\tilde{h}_2 = \left(\sum_{i=1}^m \tilde{M}_i \right)^{-1} \sum_{i=1}^m \tilde{M}_i \tilde{h}_{2,i} \quad (87)$$

where $\tilde{M}_i = J^T M_i J$ and $\tilde{h}_{2,i} = \tilde{M}_i^\dagger J^T M_i (h_{2,i} - \dot{J}\dot{q})$ are the standard pullback components

5 written in acceleration form.

[0216] Proof. The standard algebra on $(M_i f_i)$ holds, where $f_i = M_i h_{2,i}$. Pulling back gives $(\tilde{M}_i \tilde{f}_i)$ and summing gives $\sum_i (\tilde{M}_i \tilde{f}_i) = (\sum_i \tilde{M}_i, \sum_i \tilde{f}_i)$. Expressing that result in canonical form gives

$$\left(\sum_i \tilde{M}_i \right) \left(\sum_i \tilde{M}_i \right)^{-1} \sum_i \tilde{M}_i \tilde{h}_{2,i} \quad (88)$$

where $\tilde{h}_{2,i}$ is the acceleration form of the individual pullbacks. Expanding gives the formula.

10 [0217] Since $\mathcal{L}_{e_i}$ are Finsler energies, the pullback metrics $\tilde{M}_i$ are homogeneous of degree 0 in velocity (i.e. they depend only on the normalized velocity $\hat{x}$). Therefore, $\tilde{h}_2$ is homogeneous of degree 2 and the pullback forms a geometry generator.

[0218] In general, once a geometry is energized with respect to some Finsler energy $\mathcal{L}_e$, it usually does not exhibit constant Euclidean speed $||\dot{x}||$ as it attempts to conserve $\mathcal{L}_e$. Since
 15 the underlying geometric fabric is defined by a speed agnostic velocity, doing so it relatively straightforward. In order to maintain constant Euclidean speed, the behavioral Finsler energy $\mathcal{L}_e$ would have to increase or decrease as needed. Intuitively, increasing that energy can be done by injecting energy into the system using the potential function, and decreasing the energy can be done using damping. These two operations give us latitude to regulate
 20 execution energies (such as the Euclidean energy) while still working within the framework outlined by the geometric fabric optimization theorems to guarantee convergence and stability. The primary tools that enable such execution energy regulation are provided by the following proposition.

energy with metric M_e , and h a forcing potential. Let $\alpha_{\mathcal{L}_e}$ be such that $\ddot{x} = -h_2 + \alpha_{\mathcal{L}_e}\dot{x}$ maintains constant $\mathcal{L}_e$ (the energization coefficient), α_{alt}^0 it be such that $\ddot{x} = -h_2 + \alpha_{alt}^0\dot{x}$ maintains constant execution energy $\mathcal{L}_e^{alt}$, and α_{alt}^ψ be such that $\ddot{x} = -h_2 - M_e^{-1}\partial_x\psi + \alpha_{alt}^\psi\dot{x}$ maintains constant execution energy. Let α_{alt} be an interpolation between α_{alt}^0 and α_{alt}^ψ . Then the system

$$\ddot{x} = -h_2 - M_e^{-1}\partial_x\psi + \alpha_{alt}\dot{x} - \beta\dot{x} \quad (89)$$

is optimizing when $\beta > \alpha_{alt} - \alpha_{\mathcal{L}_e}$.

[0220] Proof. By definition $\alpha_{\mathcal{L}_e}$ is an energization coefficient, so

$$\ddot{x} = -h_2 - M_e^{-1}\partial_x\psi + \alpha_{alt}\dot{x} - \tilde{\beta}\dot{x} \quad (90)$$

optimizes when $\beta > 0$. That means:

$$\ddot{x} = -h_2 - M_e^{-1}\partial_x\psi + (\alpha_{\mathcal{L}_e} + \alpha_{alt} - \alpha_{alt})\dot{x} - \tilde{\beta}\dot{x} \quad (91)$$

$$= -h_2 - M_e^{-1}\partial_x\psi + \alpha_{alt}\dot{x} - (\alpha_{alt} - \alpha_{\mathcal{L}_e} - \tilde{\beta})\dot{x} \quad (92)$$

[0221] Optimizes when $\tilde{\beta} > 0$. Using $\beta = \alpha_{alt} - \alpha_{\mathcal{L}_e} + \tilde{\beta}$ thus providing $\beta - (\alpha_{alt} - \alpha_{\mathcal{L}_e}) + \tilde{\beta} > 0$ which implies $\beta > \alpha_{alt} - \alpha_{\mathcal{L}_e}$.

[0222] The fundamental requirement from the analysis to maintain constant $\mathcal{L}_e$ is $\beta = \alpha_{alt} - \alpha_{\mathcal{L}_e}$; to optimize, the system must ensure that β is strictly larger than that (in particular, in order to converge to a local minimum). The system can use the restriction $\beta \geq 0$ to maintain the semantics of a damper, giving the following inequality $\beta \geq \max\{0, \alpha_{alt} - \alpha_{\mathcal{L}_e}\}$. To converge, that inequality must be strict.

[0223] Note that α_{alt} is the standard energy transform for the geometric term $-h_2$ alone, while α_{alt}^ψ ensures that the forcing term is included in the transform as well. Therefore, if $\beta = 0$ have the following is provided:

[0224] The system under α_{alt}^ψ

$$\ddot{x}_1 = -h_2 - M_e^{-1}\partial_x\psi + \alpha_{alt}^\psi\dot{x} \quad (93)$$

and zero potential $\psi = 0$, the system

$$\ddot{x}_2 = -h_2 + \alpha_{alt}^0 \dot{x} \quad (94)$$

will maintain constant $\mathcal{L}_e^{alt}$ while the forced system

$$5 \quad \ddot{x}_3 = -h_2 - M_e^{-1} \partial_x \psi + \alpha_{alt}^0 \dot{x} \quad (95)$$

will force the system while moving between energy levels as well. Therefore, the difference between Equations 95 and 93 must be the extra component of $-M_e^{-1} \partial_x h$ accelerating the system with respect to this execution energy. That component is

$$\ddot{x}_3 - \ddot{x}_1 = (-h_2 - M_e^{-1} \partial_x \psi + \alpha_{alt}^0 \dot{x}) - (-h_2 - M_e^{-1} \partial_x \psi + \alpha_{alt}^\psi \dot{x}) \quad (96)$$

$$10 \quad = (\alpha_{alt}^0 - \alpha_{alt}^\psi) \dot{x} \quad (97)$$

[0225] An interpolation between Equations 95 and 93 gives a scaling to this component:

Let $\eta \in [0, 1]$ then:

$$\eta \ddot{x}_3 + (1 - \eta) \ddot{x}_1 = \eta(-h_2 - M_e^{-1} \partial_x \psi + \alpha_{alt}^0 \dot{x}) - (1 - \eta)(-h_2 - M_e^{-1} \partial_x \psi + \alpha_{alt}^\psi \dot{x}) \quad (98)$$

$$15 \quad = -h_2 - M_e^{-1} \partial_x \psi + (\eta \alpha_{alt}^0 + (1 - \eta) \alpha_{alt}^\psi) \dot{x} \quad (99)$$

[0226] The added component is now:

$$(\eta \ddot{x}_3 + (1 - \eta) \ddot{x}_1) - \ddot{x}_1 = -h_2 - M_e^{-1} \partial_x \psi + (\eta \alpha_{alt}^0 + (1 - \eta) \alpha_{alt}^\psi) \dot{x} - (-h_2 - M_e^{-1} \partial_x \psi + \alpha_{alt}^\psi \dot{x}) \quad (100)$$

$$= [\eta \alpha_{alt}^0 + (1 - \eta) \alpha_{alt}^\psi - \alpha_{alt}^\psi] \dot{x} \quad (101)$$

$$20 \quad = \eta (\alpha_{alt}^0 - \alpha_{alt}^\psi) \dot{x} \quad (102)$$

when $\eta = 0$, take the entirety of system $\ddot{x}_1$, which projects the entirety of $-h_2 - M_e^{-1} \partial_x \psi$. In correspondence, $\eta (\alpha_{alt}^0 - \alpha_{alt}^\psi) \dot{x} = 0$ when $\eta = 0$. Similarly, when $\eta = 1$, take all of $\ddot{x}_3$ which leaves the full $M_e^{-1} \partial_x \psi$ in tact. Here, the entire component $\eta (\alpha_{alt}^0 - \alpha_{alt}^\psi) \dot{x}$ remains intact when $\eta = 1$.

theorem acts to modulate the component $\eta(\alpha_{alt}^0 - \alpha_{alt}^\psi)\dot{x}$ defining the amount of $-M_e^{-1}\partial_x\psi$ to let through to move the system between execution energy levels. A typical strategy for speed control could then be to:

- 5 1. Choose an execution energy $\mathcal{L}_e^{alt}$ to modulate.
2. At each cycle, calculate $\alpha_{alt}^0, \alpha_{alt}^\psi, \alpha_{\mathcal{L}_e}$.
3. Choose $\eta \in [0, 1]$ to increase the energy as needed, using the extremes of $\eta = 0$ to maintain execution energy and $\eta = 1$ to fully increase execution energy.
4. Choose damper under the constraint $\beta \geq \max \{0, \alpha_{alt} - \alpha_{\mathcal{L}_e}\}$ with $\alpha_{alt} = \eta_{alt}^0 +$
10 $(1 - \eta) \alpha_{alt}^\psi$. Use a strict inequality to remove energy from the system to ensure convergence. Note that even with $\eta = 1$ (fully active potential), the bound will adjust accordingly and ensure convergence under strict inequality.

[0228] The full system executed at each cycle is

$$\ddot{x} = v_\perp + \eta v_{//} - \beta \dot{x}, \quad (103)$$

- 15 (with η and β defined as above) where $v_\perp = P_{\mathcal{L}_e^{alt}}[-h_2 - M_e^{-1}\partial_x\psi] = -h_2 - M_e^{-1}\partial_x\psi + \alpha_{alt}^\psi \dot{x}$, is the component of the system preserving execution energy $\mathcal{L}_e^{alt}$, and $v_{//}$ is the remaining (execution energy changing) component such that $v_\perp + v_{//} = -h_2 - M_e^{-1}\partial_x\psi$ reconstructs the original system.

- [0229] When $\beta + \alpha_{alt} - \alpha_{\mathcal{L}_e}$, the underlying behavioral energy $\mathcal{L}_e$ is strictly maintained.
- 20 When the damping is strictly larger than that lower bound $\beta > \alpha_{alt} - \alpha_{\mathcal{L}_e}$, that system energy decreases. As long as that strict inequality is satisfied, convergence is guaranteed.

[0230] The projection equation for α is calculated in the same way as the energization alpha used to energize geometries:

$$\alpha = -(\dot{x}^T M_e \dot{x})^{-1} \dot{x}^T [M_e \ddot{x}_d - f_e], \quad (104)$$

- 25 where the spec (M_e, f_e) define the energy equation $M_e \ddot{x} + f_e = 0$ for the underlying behavioral energy $\mathcal{L}_e$ and $\ddot{x}_d$ in this case is either $\ddot{x}_d^0 = -h_2$ for α_{alt}^0 or $\ddot{x}_d^\psi = -h_2 - M_e^{-1}\partial_x\psi$ for α_{alt}^ψ .

$$\alpha_{alt} = -(\dot{x}^T \dot{x})^{-1} \dot{x}^T [\ddot{x}_d - 0] = \frac{-\dot{x}^T \ddot{x}_d}{\dot{x}^T \dot{x}}, \quad (105)$$

which gives:

$$\ddot{x} = \ddot{x}_d + \alpha_{alt} \dot{x}, \quad (106)$$

$$5 \quad = \ddot{x}_d - \frac{-\dot{x}^T \ddot{x}_d}{\dot{x}^T \dot{x}} \dot{x} = [I - \hat{x} \hat{x}^T] \ddot{x}_d, \quad (107)$$

$$= P_{\dot{x}}^\perp [\ddot{x}_d], \quad (108)$$

where $P_{\dot{x}}^\perp$ is the projection matrix projecting orthogonally to $\dot{x}$. The above analysis is therefore just a generalization of this form of orthogonal projection to arbitrary execution energies.

10 [0232] Although the theory provides that any potential function can be optimized over an arbitrary geometric fabric, the shape of the potential function can lead to more desirable system behavior and timely convergence. Specifically, an acceleration-based potential design is easy to use and tune and consists of a baseline potential $\psi_1(x)$ and energy tensor, M_ψ , from the Finsler energy, $\mathcal{L}_{e,\psi} = x^T G(x)x$. The gradient of $\psi_1(x)$ is prioritized with M_ψ ,
15 yielding a gradient of the total potential, $\psi(x)$, as

$$\partial_x \psi(x) = M_\psi \partial_x \psi_1(x) \quad (109)$$

[0233] Importantly, $\mathcal{L}_{e,\psi}$ should be added to a system's energy such that the forcing term in the system's acceleration, can be approximated by $\partial_x \psi_1(x)$ when M_ψ is large (high priority), thereby dominating the system's mass, M_e . That is,

$$20 \quad M_e^{-1} M_\psi \partial_x \psi_1(x) \approx \partial_x \psi_1(x), \quad (110)$$

[0234] Under this condition, the forced acceleration profile becomes

$$\dot{x} = -h_2 - M_e^{-1} M_\psi \partial_x \psi_1(x) = +\alpha \dot{x}, \quad (111)$$

$$\approx -h_2 - \partial_x \psi_1(x) + \alpha \dot{x}, \quad (112)$$

hence acceleration-based design. For $\partial_x \psi_1(x)$ to originate from a valid, scalar potential
25 function, $\psi_1(x)$ and M_e must be chosen such that $\partial_{xx}^2 \psi(x)$ is symmetric. This property is facilitated by designing $M_\psi = \omega(\|x\|^2)I$, where $\omega(\cdot) \in \mathbb{R}^+$ is a scalar function that makes

function. Symmetry of $\partial_{xx}^2 \psi(x)$ can then be shown as

$$\partial_{xx}^2 \psi(x) = \partial_x [(\omega(\| \dot{x} \|^2)I)(2I'(\| x \|^2)x)], \quad (113)$$

$$= \partial_x [r(\| x \|^2)x], \quad (114)$$

$$5 \quad = r(\| x \|^2)I + 2r', (\| x \|^2)xx^T \quad (115)$$

where $r = 2\omega(\| x \|^2)I'(\| x \|^2)$. The final expression is a sum of symmetric terms and therefore, $\partial_{xx}^2 \psi(x)$ is symmetric.

[0235] As indicated, policies or geometric fabrics for causing machines to perform one or more motions can be based on policy layers and fabric layers that cause the machines to perform the one or more motions using a layer by layer or fabric by fabric approach. These polices or geometric fabrics are a special type of fabric that expresses its unbiased nominal behavior as a generalized nonlinear geometry in the machine's configuration space; they constitute the most concrete incarnation of optimization fabric and capture many of intuitive properties that make RMPs so powerful, such as acceleration-based policy design and independent priority metric specification. Geometric fabrics can be conveniently constructed in parts distributed across a transform tree of relevant task spaces. But, importantly, they inherit key theoretical properties from the theory of fabrics, including stability and their unbiased behavior. Additionally, due to their construction as nonlinear geometries of paths, geometric fabrics exhibit a characteristic geometric consistency which allows for their construction in layer-wise to mitigate design complexity and each layer independently controls execution speed by accelerating the machine along a direction of motion without affecting the overall quality of the machine's motion behavior.

[0236] Furthermore, as described, geometric fabrics and polices build on the theory of spectral semi-sprays (specs), which generalize the idea of modular second-order differential equations derived and used as RMPs. That is, let $\mathcal{C}$ be the d -dimensional configuration space of the machine. A vector-notation describes elements of a space in coordinates. Mapped task spaces $x = \phi(q)$ are defined in coordinates denoting $q \in \mathcal{C} \subset \mathbb{R}^d$ and $x \in \mathcal{X} \subset \mathbb{R}^n$ with Jacobian matrix $J = \partial_x \phi$, used in the relations $\dot{x} = J\dot{q}$ and $\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$.

[0237] Natural-form specs $(M, f)_x$ represent equations of the form $M(x, \dot{x})\ddot{x} + f(x, \dot{x}) = 0$ and their algebra derives from how these equations sum and transform under $\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$

$h(x, \dot{x}) = 0$ where $h = M^{-1}\ddot{x}$. For robotics applications it is useful to additionally introduce a policy-form spec $[M, \pi]_x$ to denote the solved policy expression $\ddot{x} = -h(x, \dot{x}) = \pi(x, \dot{x})$ to emphasize $\pi = -h$ is an acceleration policy.

5 [0238] A transform tree can be constructed and used for task spaces where the specs reside. Each directed edge of the tree represents the differentiable map taking its parent (domain) to its child (co-domain). Specs populating a transform tree collectively represent a complete second-order differential equation in parts, linking a given spec to the root via the chain of differentiable maps encountered along the unique path to the root. Denoting that composed
10 map as $\dot{x} = \phi(q)$ as above, use the expressions $\dot{x} = J\dot{q}$ and $\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$ relating velocities and accelerations in the task space to velocities and accelerations in the root to derive a spec algebra that defines both how specs combine on a single space and how they transform
15 backward across edges from child to parent. The tree implicitly represents a complete differential equation at the root as a sum of the parts, computed by recursive application of the spec algebra.

[0239] The theory of generalized nonlinear geometry and Finsler energy are important for the derivation of geometric fabrics. For example, A generalized nonlinear geometry is an acceleration policy $\ddot{x} = \pi(x, \dot{x})$ for which π has a special homogeneity property, such as positively homogeneous of degree two, which means that for any $\lambda \geq 0$, $\pi(x, \lambda\dot{x}) =$
20 $\lambda^2\pi(x, \dot{x})$. It can be shown that the homogenous of degree two (HD2) property ensures the differential equation is more than just a collection of trajectories (its integral curves); it additionally has a path consistency property whereby every integral curve starting from a given position x_0 with velocity $\dot{x}_0 = \eta\hat{n}$ pointing in a given direction $\hat{n}$ (here $\eta > 0$) will follow the same path. In particular, any variant of the differential equation of the form $\ddot{x} =$
25 $\pi(x, \dot{x}) + \alpha(t, x, \dot{x})\dot{x}$, where $\alpha \in \mathbb{R}$, will have integral curves that trace out the same paths as π . That geometric consistency property turns π into a geometry of paths.

[0240] A Finsler energy $\mathcal{L}_e(x, \dot{x})$ is a generalization of classical kinetic energy from classical mechanics (the classical kinetic energy $\mathcal{K} = \frac{1}{2}x^T\dot{G}(x)\dot{x}$ is a form of Finsler energy). Analogous to the classical case, the Euler-Lagrange equation applied to a Finsler energy
30 defines an equation of motion $M_e(x, \dot{x})\ddot{x} = f_e(x, \dot{x}) = 0$ where $M_e = \partial_{\dot{x}\dot{x}}^2\mathcal{L}_e$ is the energy (or metric) tensor and $f_e = \partial_{\dot{x}\dot{x}}\mathcal{L}_e\dot{x} = \partial_x\mathcal{L}_e$ captures curvature terms (Coriolis and centripetal

motion when $\mathcal{L}_e = \mathcal{K}$, for which $M_e(x, \dot{x}) = G(x)$.

[0241] In geometric fabrics, the energy tensor defines the policy's priority metric and the curvature terms f_e are used for stability (see Section VI-C). Finsler energies are Lagrangians,

5 $\mathcal{L}_e(x, \dot{x})$, that satisfy:

- 1) Positivity: $\mathcal{L}_e(x, \dot{x}) \geq 0$, with equality only for $\dot{x} = 0$
- 2) Homogeneity: $\mathcal{L}_e(x, \dot{x})$ is positively homogenous of degree 2 in $\dot{x}$; i.e. for $\lambda \geq 0$ providing $\mathcal{L}_e(x, \lambda \dot{x}) = \lambda^2 \mathcal{L}_e(x, \dot{x})$
- 3) Energy tensor inevitability: $M_e = \partial_{\dot{x}\dot{x}}^2 \mathcal{L}_e$ is everywhere invertible.

10 [0242] The metric tensor $M_e(x, \dot{x})$ is in general a function of velocity as well as position, although the above homogeneity requirement enforces that M_e depends only on the directionality of the velocity ($M_e(x, \dot{x}) = M_e(x, \hat{\dot{x}})$ for $\dot{x} \neq 0$) and not the magnitude (i.e. it is homogeneous of degree 0). This dependence on directionality enables us to design directionally dependent priority matrices as metric tensors of Finsler energies.

15 [0243] Geometric fabrics are a form of optimization fabric which is a special type of differential equation designed to induce behavior by influencing the optimization path of a differential optimizer. This disclosure describes pragmatically how to effectively design the fabric to encode a desired behavior.

[0244] A forced geometric fabric is a collection of fabric terms defined as pairs $(\mathcal{L}_e, \pi)_x$ of a Finsler energy $\mathcal{L}_e(x, \dot{x})$ and an acceleration policy $\ddot{x} = \pi(x, \dot{x})$. Geometric terms define the fabric while forcing terms define the objective. A *geometric term* is a term $(\mathcal{L}_e, \pi_2)_x$ for which π_2 is an HD2 geometry. A *forcing term* is a term $(\mathcal{L}_e, -M_e^{-1} \partial_x \psi)_x$ which derives its policy from a potential function. Fabric terms can be added to spaces of a transform tree for the modular design of composite behaviors.

25 [0245] Each fabric term defines a triple $(M_e, f_e, \pi)_x$, where $M_e \ddot{x} + f_e = 0$ derives from the Euler-Lagrange equation applied to $\mathcal{L}_e$, which can be viewed as two specs, a policy spec $[M_e, \pi]_x$ and a natural energy spec $(M_e, f_e)_x$. Geometric fabric summation and pullback is, accordingly, defined in terms of the algebra of these two constituent specs.

[0246] Geometric fabrics are unbiased and thereby never prevent a system from reaching a
30 local minimum of the objective. The objective, therefore, encodes concrete task goals

consistent speed-invariant geometry of paths, which both simplifies the intuition on how they sum and enables behavior-invariant execution speed control.

[0247] The design of a geometric fabric follows the intuition of designing RMPs. Policy
 5 specs $[M_e, \pi_2]_x$ model both a desired behavior π_2 and a priority matrix on that behavior M_e defining how it combines with other policies as a metric- weighted average of parts. The spectrum of M_e can assign different weights to different directions in the space, and both $\pi_2(x, \dot{x})$ and $M_e(x, \dot{x})$ have the flexibility of depending on both position x and velocity $\dot{x}$. Since M_e is HD0, geometric terms remain geometric under summation and pullback (e.g. the policy
 10 resulting from a metric-weighted average of geometries is itself a geometry). The energy spec $(M_e, f_e)_x$ of each fabric term is used only to guarantee stability during execution.

Practitioners can, therefore, simply focus on designing the behavior policy specs $[M_e, \pi_2]_x$.

[0248] Once the forced geometric fabric is designed, it can be transform it by accelerating and decelerating along the direction of motion to maintain a given measured of *execution*
 15 *energy* (e.g. speed of the end-effector or joint speed through the configuration space). The geometric consistency of the fabric means the behavior remains consistent despite these speed modulations. Many numerical integrators are appropriate for integrating the final differential equation. For example, Euler (1ms time step) or fourth order Runge-Kutta (10ms time step) exhibit a good trade-off between speed and accuracy.

[0249] In an embodiment, a machine's series of motions can be designed in three parts
 20 (using a transform tree of task spaces): (1) design the underlying behavioral fabric, (2) add a driving potential to define task goals, (3) design an execution energy for speed control. In some embodiments, the transform tree can be designed/constructed through (1) Forward pass: Populate the nodes with the current state from the root to the leaves. (2) Backward pass:
 25 Evaluate the specs and pull them to root in separate channels, an energy channel for the geometric terms' energy specs, a policy channel for the geometric terms' policy specs, an execution energy channel for the execution energy specs, and a forcing policy channel for the forcing terms' policy specs. Add the forcing term's energy specs to the energy channel. (3) Use the four channels' root results to calculate the final desired acceleration using speed
 30 control.

[0250] Geometric fabrics follow acceleration-based design principles captured in the original canonical-form RMPs. A geometric fabric is a pair $(\mathcal{L}_e, \pi_2)_x$ characterizing two

while the geometry spec captures behavior. Behavioral design focuses on constructing the latter, using the class of HD2 geometries to model π_2 and deriving M_e as the energy tensor of a Finsler geometry $\mathcal{L}_e$.

- 5 [0251] When geometric fabrics are summed $\sum_i (\mathcal{L}_e^{(i)}, \pi_2^{(i)})_x$, the combined fabric's geometry spec (capturing its behavior) $\sum_i (M_e^{(i)}, \pi_2^{(i)})_x = (\tilde{M}_e, \tilde{\pi}_2)$ is a metric-weighted average of the contributing geometries $\tilde{\pi}_2 = \left(\sum_i M_e^{(i)} \right)^{-1} \sum_i M_e^{(i)} \pi_2^{(i)}$, prioritized by the total metric $M_e = \sum_i M_e^{(i)}$. When populating a transform tree, this intuitive combination rule is applied recursively at each node. Designers need only focus on intuitively creating modular acceleration policies (as HD2 geometries) in the different spaces and prioritizing them with metric tensors (from Finsler energies).

- 15 [0252] An HD2 geometry is a differential equation $\ddot{x} + h_2(x, \dot{x})$ where h_2 is HD2, which is usually denote in policy form $\ddot{x} = -h_2(x, \dot{x}) = \pi_2(x, \dot{x})$. Constructing an HD2 geometry is straightforward given the following rules of homogeneous functions: (1) a sum of HD2 functions is HD2; (2) multiplying homogeneous functions adds their degrees (denoting an HDk function as f_k , examples are $f_2 f_0 = f_2$, $f_1 f_1 = f_2$, etc.). For instance, a simple way to design an HD2 geometry is to choose an HD0 policy $\pi_0(x)$ that depends only on position and form $\pi_2(x, \dot{x}) = \|\dot{x}\|^2 \pi_0(x)$; by scaling it by $\|\dot{x}\|^2$. $\pi_0(x)$ can be chosen as the negative gradient of a potential $\pi_0(x) = -\partial_x \psi(x)$.

- 20 [0253] In at least one embodiment, it can be intuitive to design a geometric fabrics' forcing potential as a forcing spec $\mathcal{F} = [M_f, \pi_f]_x$ in policy form so it's treated intuitively as another acceleration policy averaged into the final metric weighted average. This policy π_f can, therefore, implicitly express a forcing potential $\psi_f(x)$ whose negative gradient is given by $-\partial_x \psi_f(x) = M_f \pi_f$. When designing $\mathcal{F}$, enforce that M_f and π_f remain theoretically compatible in that sense. Choose $\pi_f = \nabla_x \psi_{acc}(x)$ where ψ_{acc} as a potential function that is spherically symmetric around its global minimum expressing the acceleration policy directly as its negative gradient. Any metric $M_f(x)$ is theoretically compatible if it is also spherically symmetric around the same global minimum point. Note that position- only metrics are Riemannian and derive from Finsler energies of the form $\mathcal{L}_e = \frac{1}{2} \dot{x}^T M_f(x) \dot{x}$.

$-\mathbf{M}_e^{-1}\partial_{\mathbf{x}}\psi(\mathbf{x}) + \pi_0(\mathbf{x}, \dot{\mathbf{x}}) + \alpha_{\text{reg}}\dot{\mathbf{x}}$ with $\beta_{\text{reg}} = s_{\beta}(\mathbf{x})B + \underline{B} + \max\{0, \alpha_{\text{ex}}^{\eta} - \alpha_{\mathcal{L}_e}\}$. $\underline{B} > 0$ is a (small) baseline damping, the $B > \underline{B}$ is a larger damping coefficient for succinct convergence. The switch $s_{\beta}(\mathbf{x})$ turns on close to the target:

$$s_{\beta}(\mathbf{x}) = \frac{1}{2} \left(\tanh \left(-\alpha_{\beta} (\|\mathbf{x}\| - r) \right) + 1 \right)$$

5 where $\alpha_{\beta} \in \mathbb{R}^+$ is a gain defining the switching rate, and $r \in \mathbb{R}^+$ is the radius where the switch is half-way engaged. Denoting the desired execution energy as $\mathcal{L}_e^{\text{ex,d}}$, use the following policy for η .

$$\eta = \frac{1}{2} \left(\tanh \left(-\alpha_{\eta} (\mathcal{L}_e^{\text{ex}} - \mathcal{L}_e^{\text{ex,d}}) - \alpha_{\text{shift}} \right) + 1 \right)$$

10 where $\alpha_{\eta}, \alpha_{\text{shift}} \in \mathbb{R}^+$ adjust the rate and offset, respectively, of the switch as an affine function of the speed (execution energy) error. Finally, α_{boost} is modeled as $\alpha_{\text{boost}} = k_{\eta} \left(1 - s_{\beta}(\mathbf{x}) \right) \frac{1}{\|\dot{\mathbf{x}}\| + \epsilon}$, where $k \in \mathbb{R}^+$ is a gain that directly sets the desired level of acceleration, η (from above) sets $\alpha_{\text{boost}} = 0$ when the desired speed is achieved, $1 - s_{\beta}(\mathbf{x})$ sets $\alpha_{\text{boost}} = 0$ when the system is within the region of higher damping. The normalization by $\|\dot{\mathbf{x}}\| + \epsilon$ ensures that α_{boost} is directly applied along $\hat{\mathbf{x}}$ with a very small positive value for ϵ to ensure numerical stability. This overall design injects more energy into the system when $-\alpha_{\text{boost}} < \alpha_{\text{ex}}^{\eta} - \alpha_{\mathcal{L}_e}$. Since this injection occurs for finite time, the total system energy is still bounded. The additional switches ensure that the system is still subject to positive damping, guaranteeing convergence.

20 [0255] In at least one embodiment, the effectiveness of the layer-wise construction of a geometric fabric enables the robot to reach into and out of three sets of cubbies or box containers. In one example, there are six reachable cubbies or box containers in front and two on either side of a Franka arm. The length, width, and depth of each cubby/box are $0.3m$. The following discusses the fabric layers, where every new layer rests upon the previous, fixed layers, mitigating design and tuning complexity.

25 [0256] Each policy is defined as an HD2 geometry of the form $\ddot{\mathbf{x}} = -\|\dot{\mathbf{x}}\|^2 \partial_{\mathbf{x}}\psi(\mathbf{x})$. Policies are weighted by $\mathbf{M}_e(\mathbf{x}, \dot{\mathbf{x}})$ from the Finsler energy, $\mathcal{L}_e = \mathbf{x}^T \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) \dot{\mathbf{x}}$. $\psi(\mathbf{x})$ and $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ are defined as follows.

point-to-point end-effector navigation absent obstacles.

[0258] End-effector Attraction: Attraction towards a target uses the task map, $y =$

$\phi_{\text{att}}(x) = x_t - x$, where $x, x_t \in \mathbb{R}^3$ are the current and target end-effector position in

- 5 Euclidean space. The metric is simply an identity matrix scaled by $s(\|y\|)$, $G_{\text{att}} = s(\|y\|)I$ where $s(\|y\|) = 40$ if $\|y\| < 0.5$, and $s(\|y\|) = 1$, otherwise. The acceleration-based potential gradient, $\partial_q \psi_1(y) = M_{\text{att}}(y) \partial_q \psi_1(y)$, uses:

$$\psi_1(y) = k \left(\|y\| + \frac{1}{\alpha_\psi} \log(1 + e^{-2\alpha_\psi \|y\|}) \right)$$

where $k \in \mathbb{R}^+$ controls the overall gradient strength, $\alpha_\psi \in \mathbb{R}^+$ controls the transition rate of

- 10 $\psi_1(y)$ from a positive constant to 0, and here, $\alpha_\psi = 10$, $k = 10$.

[0259] Joint Limit Avoidance: This behavior uses two 1D task maps per joint, $x_j^u =$

$\phi_u(q_j) = \bar{q}_j - q_j$ and $x_j^l = \phi_l(q_j) = q_j - \underline{q}_j$, where q_j and $\underline{q}_j$ are the upper and lower limits

of the j^{th} joint. Denoting both generically as x , the metric G_l is defined as $G_l(x, \dot{x}) = s(\dot{x}) \frac{\lambda}{x^2}$,

where $s(\dot{x}) = 0$ if $\dot{x} > 0$ and $s(\dot{x}) = 1$ otherwise (1D normalization), and $\lambda = 10$.

- 15 Effectively, this removes the effect of the coordinate limit geometry once motion is orthogonal or away from the limit. The acceleration-based potential gradient, $\partial_q \psi(x) = M_l(x) \partial_q \psi_{1,1}(x)$, uses:

$$\psi_{1,l}(x) = \frac{\alpha_1}{x^2} + \alpha_2 \log(e^{-\alpha_3(x-\alpha_4)} + 1)$$

and $M_l(x)$ comes from Finsler energy $\mathcal{L}_e(x) = G_l(x)\dot{x}^2$, where $G_l(x)$ is $G_l(x, \dot{x})$ that drops

- 20 $s(\dot{x})$ term, $\alpha_1, \alpha_2 \in \mathbb{R}^+$ control the significance and mutual balance of the first and second terms. $\alpha_3 \in \mathbb{R}^+$ controls the sharpness of the smooth rectified linear unit (SmoothReLU) while $\alpha_4 \in \mathbb{R}^+$ offsets the SmoothReLU. Overall, $\psi_1(x) \rightarrow \infty$ as $x \rightarrow 0$ and $\psi_1(x) \rightarrow 0$ as $x \rightarrow \infty$, which impedes motion towards a limit. In this experiment, $\alpha_1 = 0.4$, $\alpha_2 = 0.5$, $\alpha_3 = 20$, and $\alpha_4 = \pi/6$.

- 25 [0260] Default Configuration: The task map for this behavior is $x = \phi_{\text{dc}}(q) = q_0 - q$, where q_0 is a default configuration. The metric G_{dc} is simply an identity matrix scaled by a constant λ_{dc} , $G_{\text{dc}} = \lambda_{\text{dc}}I$, where $\lambda_{\text{dc}} = 0.5$. The acceleration-based potential gradient is

configurations are created to cover different regions of the robot. This behavior controls robot posture and resolves manipulator redundancy.

[0261] The next layer enables the end-effector to extract and enter any cubby/box (ignoring collision). Cubby Extraction: Cubby extraction uses two task maps that are the distance between: 1) the end-effector and the front plane of the cubby, $y_1 = \phi_1(x) = \|x_f - x\|$ if the end-effector is inside of the cubby, $y_1 = \phi_1(x) = \|x_f - x\|$, otherwise, where $x, x_f \in \mathbb{R}^3$ are the end-effector position and its orthogonal projection onto the front plane, respectively; and 2) the end-effector and a line that is centered and orthogonal to the front plane of the target cubby, $y_2 = \phi_2(x) = \|x_c - x\|$, where $x, x_c \in \mathbb{R}^3$ are the end-effector position, and the closest point on the line to the end-effector, where the line is orthogonal and centered with the front plane. The priority metric is designed as

$$G_w(x) = s(y_1) \left((\bar{m} - \underline{m})s(y_2) + \underline{m} \right) l.$$

where $s(y_1) = 1$ if $y_1 < 0.1y_1$ and $s(y_1) = 0$, otherwise. $s(y_2) = 0.5(\tanh(\alpha_m(y_2 - r)) + 1)$, $\bar{m}, \underline{m} \in \mathbb{R}^+$ are the upper and lower isotropic masses, respectively, and $\alpha_m \in \mathbb{R}^+$ defines the rate of transition between 0 and 1, while $r \in \mathbb{R}^+$ offsets the transition. For this experiment, $\bar{m} = 5$, $\underline{m} = 0$, $\alpha_m = 100$, and $r = 0.15$. Overall, the priority vanishes if the end-effector is either more than 0.1m away from the front plane (outside of the cubby) or within 0.15m of the target cubby center line. The potential function is the same as defined in Eq. 4, where $\alpha_1 = 0$, $\alpha_2 = 15$, $\alpha_3 = 100$, $\alpha_4 = 0.05$.

[0262] Target Cubby Attraction: An additional target attraction policy is used to help pull the end-effector inside a target cubby. It is the same as the end-effector attraction defined in Layer 1, but with the priority metric defined as Eq. 5, where $s(y_1) = 1 \forall y_1$, $\bar{m} = 5$, $\underline{m} = 0$, $\alpha_m = -100$, and $r = 0.15$ and $\alpha_\psi = 10$, $k = 40$ are used in Eq. 3. Overall, the priority remains zero until it enters a cylindrical region aligned with the target cubby/box. The heightening priority funnels motion into the cubby/box.

[0263] Way-Point Attraction: This term guides the arm to the cubby/box opening by attracting to a point 0.15m ahead of the front plane. The term generally matches the end-effector attraction term above, but with a different target and a switching function on the metric disabling it once within the column of the target cubby/box. Specifically, make the

position; 2) define the switching function with the task map $y_2 = \emptyset_2(x) = ||x_2 - x||$ as described in the cubby/box extraction policy; 3) $k = 20$ is used. Note this term guides the arm but, as a geometric term, does not affect convergence to the target.

5 [0264] The next layer enables complete collision avoidance with the cubbies/boxes. Cubby Collision: The task map for this behavior is $y = \emptyset_2(x)$ which captures the minimum distance between a point on the arm and the cubby, where x in $\mathbb{R}^3$ is a designated collision point on the robot. Denoting the closest point on the cubby as x_c , use $y_2 = \emptyset_2(x) = ||x_c - x||$ (ignoring dependencies of x_c on x for simplicity). The metric is defined as a function of
10 position, $G_b(y) = \frac{k_b}{y^2}$, where $k_b \in \mathbb{R}^+$ is a barrier gain. The acceleration-based potential gradient, $\partial_q \psi_b(y) = M_b(y) \partial_q \psi_{1,b}(y)$, uses $\psi_{1,b}(y) = \frac{\alpha_b}{y^3}$, where $\alpha_b \in \mathbb{R}^+$ is the barrier gain. Here, $k_b = 1$ and $\alpha_b = 0.1$.

[0265] The optimization potential is an acceleration-based design, and it is designed for task space, $y = \emptyset(x) = x - x_t$, where $x, x_t \in \mathbb{R}^3$ are the current and target end-effector
15 position in Euclidean space. The acceleration-based potential gradient, $\partial_y \psi(y) = M_\psi(y) \partial_y \psi_1(y)$, uses $\psi_1(y)$ as defined in Eq. 3 with $\alpha_\psi = 10, k = 20$,

$$G_\psi(y) = \left((\overline{m} - \underline{m}) s(||y||) + \underline{m} \right) I,$$

where $s(||y||) = 0.5(\tanh(\alpha_m(||y|| - r)) + 1)$, $\overline{m}, \underline{m} \in \mathbb{R}^+$ are the upper and lower isotropic masses, respectively, and $\alpha_m \in \mathbb{R}^+$ defines the rate of transition between 0 and 1,
20 while $r \in \mathbb{R}^+$ offsets the transition. For this experiment, $\overline{m} = 40, \underline{m} = 0.1, \alpha_m = 100$, and $r = 0.15$. This design allows for small attraction potential when far away from the target while smoothly increasing priority as getting closer to the target location.

[0266] Damping values are $B = 17.5, \underline{B} = 0$. In Eq. 1, $\alpha_\beta = 50$ and $r = 0.15$. In Eq. 2, $\alpha_{eta} = 10, \alpha_{shift} = 2, \mathcal{L}_e^{ex} = y^T y$ and $\mathcal{L}_e^{ex,d} = 1$. The gain k in α_{boost} is defined as $k =$
25 $-5 | ||\dot{y}|| - \mathcal{L}_e^{ex,d} |$, where $||\dot{y}||$ is the current end-effector speed.

[0267] Using the foregoing layered approach to designing a policy, the robot intelligently navigates the cubbies/boxes. This global behavior was incrementally sequenced by adding layers of complexity to the underlying geometric fabric, a technique facilitated by geometric consistency and acceleration-based design. Specifically, in at least one embodiment, the first

entirely. The next layer improves cubby/box navigation. The next layer enables the robot for complete collision avoidance.

[0268] FIG. 7 illustrates an example of a process 700 to cause a computer-implemented action, in accordance with an embodiment. In at least one embodiment, one or more computer systems such as one or more of the computer systems described and illustrated in FIGS. 1-6, and 8A-41B, executes instructions stored in a computer-readable memory that cause the computer system to perform the process 700. In at least one embodiment, the computer system is a robotic system, such as one or more of the robotic systems described herein. The robotic system may incorporate the one or more of the computer systems described and illustrated herein.

[0269] At 702, a policy to cause a machine to execute at least one movement is identified. In at least one embodiment, the machine is a robot. For example, the machine can be the robot 206 illustrated in FIGS. 2-3, or the robot 402 illustrated in FIGS. 4-6. Alternatively, the machine can be the machine 106 illustrated in FIG. 1. In at least one embodiment, the policy can be associated with the computer system 100 illustrated in FIG. 1. Similarly, the policy can be associated with the computer system 200 illustrated in FIGS. 2-6. In at least one embodiment, the policy comprises at least one policy layer. In an example, the policy comprises a plurality of policy layers. The one or more policy layers can comprise at least one differential equation. In at least one embodiment, the at least one differential equation is a second-order differential equation. The second-order differential equation can be homogeneous of degree two. In at least one embodiment, the one or more policy layers are energized by one or more Finsler energies. The one or more Finsler energies can be homogeneous of degree two.

[0270] At 704, a first policy layer associated with the plurality policy layers is executed to cause the machine to perform a first motion that reaches an unbiased state. In at least one embodiment, the first motion caused by the first policy layer is limited by at least a first parameter associated with the machine and a second parameter associated with an area in which the machine operate. In at least one embodiment, the first parameter is a joint limit of the machine. Specifically, the joint limit can be associated with a wrist joint or other joint of a robot. In at least one embodiment, the second parameter comprises data indicating a target position to be reached by the machine. In at least one embodiment, the target position is

target position is a coordinate or coordinates within a Euclidean space in which the machinist operate.

[0271] At 706, a second policy layer associated with the plurality of policy layers is executed to cause the machine to perform a second motion. In at least one embodiment, the second motion performed by the machine builds on the first motion, caused by the first policy layer, performed by the machine. In at least one embodiment, the second motion caused by the second policy layer does not influence allowing the machine to reach the unbiased state based on the first policy layer. In at least one embodiment, one or more motion parameters of the second policy layer can be energized independent of the one or more motion parameters of the first policy layer. The one or more motion parameters can include trajectory motion parameters, acceleration motion parameters, velocity motion parameters, joint limitation parameters, redundancy parameters, directional and/or coordinate parameters, and so forth.

[0272] FIG. 8 illustrates an example of a process 800 to cause a computer-implemented action, in accordance with an embodiment. In at least one embodiment, one or more computer systems such as one or more of the computer systems described and illustrated in FIGS. 1-6, and 8A-41B, executes instructions stored in a computer-readable memory that cause the computer system to perform the process 800. In at least one embodiment, the computer system is a robotic system, such as one or more of the robotic systems described herein. The robotic system may incorporate the one or more of the computer systems described and illustrated herein.

[0273] At 802, a first policy layer associated with a plurality policy layers is generated. The first policy layer is to cause a machine to perform a first motion that reaches an unbiased state. In at least one embodiment, the first motion caused by the first policy layer is limited by at least a first parameter associated with the machine and a second parameter associated with an area in which the machine operate. In at least one embodiment, the first parameter is a joint limit of the machine. Specifically, the joint limit can be associated with a wrist joint or other joint of a robot. In at least one embodiment, the second parameter comprises data indicating a target position to be reached by the machine. In at least one embodiment, the target position is within a task space or area in which the machinist operate. In at least one embodiment, the target position is a coordinate or coordinates within a Euclidean space in which the machinist operate.

be the robot 206 illustrated in FIGS. 2-3, or the robot 402 illustrated in FIGS. 4-6.

Alternatively, the machine can be the machine 106 illustrated in FIG. 1. In at least one embodiment, the policy can be associated with the computer system 100 illustrated in FIG. 1.

5 Similarly, the policy can be associated with the computer system 200 illustrated in FIGS. 2-6. In at least one embodiment, the policy comprises at least one policy layer. In an example, the policy comprises a plurality of policy layers. The one or more policy layers can comprise at least one differential equation. In at least one embodiment, the at least one differential equation is a second-order differential equation. The second-order differential equation can
10 be homogeneous of degree two. In at least one embodiment, the one or more policy layers are energized by one or more Finsler energies. The one or more Finsler energies can be homogeneous of degree two.

[0275] At 804, a second policy layer associated with the plurality of policy layers is generated. The second policy layer is to cause the machine to perform a second motion. In at
15 least one embodiment, the second motion to be performed by the machine builds on the first motion, caused by the first policy layer, to be performed by the machine. In at least one embodiment, the second motion caused by the second policy layer does not influence allowing the machine to reach the unbiased state based on the first policy layer. In at least one embodiment, one or more motion parameters of the second policy layer can be energized
20 independent of the one or more motion parameters of the first policy layer. The one or more motion parameters can include trajectory motion parameters, acceleration motion parameters, velocity motion parameters, joint limitation parameters, redundancy parameters, directional and/or coordinate parameters, and so forth.

[0276] At 806, the first policy layer and the second policy layer are executed to cause
25 machine to execute the first motion and the second motion. In at least one embodiment, the first policy layer and the second policy layer are part of a geometric fabric. The second motion caused by the second policy layer can build on the first motion caused by the first policy layer.

INFERENCE AND TRAINING LOGIC

30 [0277] FIG. 9A illustrates inference and/or training logic 915 used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding

9B.

[0278] In at least one embodiment, inference and/or training logic 915 may include, without limitation, code and/or data storage 901 to store forward and/or output weight and/or input/output data, and/or other parameters to configure neurons or layers of a neural network trained and/or used for inferencing in aspects of one or more embodiments. In at least one embodiment, training logic 915 may include, or be coupled to code and/or data storage 901 to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs)). In at least one embodiment, code, such as graph code, loads weight or other parameter information into processor ALUs based on an architecture of a neural network to which such code corresponds. In at least one embodiment, code and/or data storage 901 stores weight parameters and/or input/output data of each layer of a neural network trained or used in conjunction with one or more embodiments during forward propagation of input/output data and/or weight parameters during training and/or inferencing using aspects of one or more embodiments. In at least one embodiment, any portion of code and/or data storage 901 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0279] In at least one embodiment, any portion of code and/or data storage 901 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or code and/or data storage 901 may be cache memory, dynamic randomly addressable memory ("DRAM"), static randomly addressable memory ("SRAM"), non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, a choice of whether code and/or code and/or data storage 901 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

[0280] In at least one embodiment, inference and/or training logic 915 may include, without limitation, a code and/or data storage 905 to store backward and/or output weight and/or input/output data corresponding to neurons or layers of a neural network trained and/or used for inferencing in aspects of one or more embodiments. In at least one

of each layer of a neural network trained or used in conjunction with one or more embodiments during backward propagation of input/output data and/or weight parameters during training and/or inferencing using aspects of one or more embodiments. In at least one embodiment, training logic 915 may include, or be coupled to code and/or data storage 905 to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs)).

[0281] In at least one embodiment, code, such as graph code, causes the loading of weight or other parameter information into processor ALUs based on an architecture of a neural network to which such code corresponds. In at least one embodiment, any portion of code and/or data storage 905 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory. In at least one embodiment, any portion of code and/or data storage 905 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or data storage 905 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, a choice of whether code and/or data storage 905 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash memory or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

[0282] In at least one embodiment, code and/or data storage 901 and code and/or data storage 905 may be separate storage structures. In at least one embodiment, code and/or data storage 901 and code and/or data storage 905 may be a combined storage structure. In at least one embodiment, code and/or data storage 901 and code and/or data storage 905 may be partially combined and partially separate. In at least one embodiment, any portion of code and/or data storage 901 and code and/or data storage 905 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0283] In at least one embodiment, inference and/or training logic 915 may include, without limitation, one or more arithmetic logic unit(s) ("ALU(s)") 910, including integer and/or floating point units, to perform logical and/or mathematical operations based, at least

may produce activations (e.g., output values from layers or neurons within a neural network) stored in an activation storage 920 that are functions of input/output and/or weight parameter data stored in code and/or data storage 901 and/or code and/or data storage 905. In at least
5 one embodiment, activations stored in activation storage 920 are generated according to linear algebraic and or matrix-based mathematics performed by ALU(s) 910 in response to performing instructions or other code, wherein weight values stored in code and/or data storage 905 and/or data storage 901 are used as operands along with other values, such as bias values, gradient information, momentum values, or other parameters or hyperparameters,
10 any or all of which may be stored in code and/or data storage 905 or code and/or data storage 901 or another storage on or off-chip.

[0284] In at least one embodiment, ALU(s) 910 are included within one or more processors or other hardware logic devices or circuits, whereas in another embodiment, ALU(s) 910 may be external to a processor or other hardware logic device or circuit that uses them (e.g., a co-
15 processor). In at least one embodiment, ALUs 910 may be included within a processor's execution units or otherwise within a bank of ALUs accessible by a processor's execution units either within same processor or distributed between different processors of different types (e.g., central processing units, graphics processing units, fixed function units, etc.). In at least one embodiment, code and/or data storage 901, code and/or data storage 905, and
20 activation storage 920 may share a processor or other hardware logic device or circuit, whereas in another embodiment, they may be in different processors or other hardware logic devices or circuits, or some combination of same and different processors or other hardware logic devices or circuits. In at least one embodiment, any portion of activation storage 920 may be included with other on-chip or off-chip data storage, including a processor's L1, L2,
25 or L3 cache or system memory. Furthermore, inferencing and/or training code may be stored with other code accessible to a processor or other hardware logic or circuit and fetched and/or processed using a processor's fetch, decode, scheduling, execution, retirement and/or other logical circuits.

[0285] In at least one embodiment, activation storage 920 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one
30 embodiment, activation storage 920 may be completely or partially within or external to one or more processors or other logical circuits. In at least one embodiment, a choice of whether activation storage 920 is internal or external to a processor, for example, or comprising

on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

5 [0286] In at least one embodiment, inference and/or training logic 915 illustrated in FIG. 9A may be used in conjunction with an application-specific integrated circuit (“ASIC”), such as a TensorFlow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, inference and/or training logic 915 illustrated in FIG. 9A may be used in
10 conjunction with central processing unit (“CPU”) hardware, graphics processing unit (“GPU”) hardware or other hardware, such as field programmable gate arrays (“FPGAs”).

[0287] FIG. 9B illustrates inference and/or training logic 915, according to at least one embodiment. In at least one embodiment, inference and/or training logic 915 may include, without limitation, hardware logic in which computational resources are dedicated or
15 otherwise exclusively used in conjunction with weight values or other information corresponding to one or more layers of neurons within a neural network. In at least one embodiment, inference and/or training logic 915 illustrated in FIG. 9B may be used in conjunction with an application-specific integrated circuit (ASIC), such as TensorFlow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a
20 Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, inference and/or training logic 915 illustrated in FIG. 9B may be used in conjunction with central processing unit (CPU) hardware, graphics processing unit (GPU) hardware or other hardware, such as field programmable gate arrays (FPGAs). In at least one embodiment, inference and/or training logic 915 includes, without limitation, code and/or data storage 901
25 and code and/or data storage 905, which may be used to store code (e.g., graph code), weight values and/or other information, including bias values, gradient information, momentum values, and/or other parameter or hyperparameter information. In at least one embodiment illustrated in FIG. 9B, each of code and/or data storage 901 and code and/or data storage 905 is associated with a dedicated computational resource, such as computational hardware 902
30 and computational hardware 906, respectively. In at least one embodiment, each of computational hardware 902 and computational hardware 906 comprises one or more ALUs that perform mathematical functions, such as linear algebraic functions, only on information

of which is stored in activation storage 920.

[0288] In at least one embodiment, each of code and/or data storage 901 and 905 and corresponding computational hardware 902 and 906, respectively, correspond to different layers of a neural network, such that resulting activation from one storage/computational pair 901/902 of code and/or data storage 901 and computational hardware 902 is provided as an input to a next storage/computational pair 905/906 of code and/or data storage 905 and computational hardware 906, in order to mirror a conceptual organization of a neural network. In at least one embodiment, each of storage/computational pairs 901/902 and 905/906 may correspond to more than one neural network layer. In at least one embodiment, additional storage/computation pairs (not shown) subsequent to or in parallel with storage/computation pairs 901/902 and 905/906 may be included in inference and/or training logic 915.

NEURAL NETWORK TRAINING AND DEPLOYMENT

[0289] FIG. 10 illustrates training and deployment of a deep neural network, according to at least one embodiment. In at least one embodiment, untrained neural network 1006 is trained using a training dataset 1002. In at least one embodiment, training framework 1004 is a PyTorch framework, whereas in other embodiments, training framework 1004 is a TensorFlow, Boost, Caffe, Microsoft Cognitive Toolkit/CNTK, MXNet, Chainer, Keras, Deeplearning4j, or other training framework. In at least one embodiment, training framework 1004 trains an untrained neural network 1006 and enables it to be trained using processing resources described herein to generate a trained neural network 1008. In at least one embodiment, weights may be chosen randomly or by pre-training using a deep belief network. In at least one embodiment, training may be performed in either a supervised, partially supervised, or unsupervised manner.

[0290] In at least one embodiment, untrained neural network 1006 is trained using supervised learning, wherein training dataset 1002 includes an input paired with a desired output for an input, or where training dataset 1002 includes input having a known output and an output of neural network 1006 is manually graded. In at least one embodiment, untrained neural network 1006 is trained in a supervised manner and processes inputs from training dataset 1002 and compares resulting outputs against a set of expected or desired outputs. In at least one embodiment, errors are then propagated back through untrained neural network

untrained neural network 1006. In at least one embodiment, training framework 1004 includes tools to monitor how well untrained neural network 1006 is converging towards a model, such as trained neural network 1008, suitable to generating correct answers, such as in result 1014, based on input data such as a new dataset 1012. In at least one embodiment, training framework 1004 trains untrained neural network 1006 repeatedly while adjust weights to refine an output of untrained neural network 1006 using a loss function and adjustment algorithm, such as stochastic gradient descent. In at least one embodiment, training framework 1004 trains untrained neural network 1006 until untrained neural network 1006 achieves a desired accuracy. In at least one embodiment, trained neural network 1008 can then be deployed to implement any number of machine learning operations.

[0291] In at least one embodiment, untrained neural network 1006 is trained using unsupervised learning, wherein untrained neural network 1006 attempts to train itself using unlabeled data. In at least one embodiment, unsupervised learning training dataset 1002 will include input data without any associated output data or “ground truth” data. In at least one embodiment, untrained neural network 1006 can learn groupings within training dataset 1002 and can determine how individual inputs are related to untrained dataset 1002. In at least one embodiment, unsupervised training can be used to generate a self-organizing map in trained neural network 1008 capable of performing operations useful in reducing dimensionality of new dataset 1012. In at least one embodiment, unsupervised training can also be used to perform anomaly detection, which allows identification of data points in new dataset 1012 that deviate from normal patterns of new dataset 1012.

[0292] In at least one embodiment, semi-supervised learning may be used, which is a technique in which in training dataset 1002 includes a mix of labeled and unlabeled data. In at least one embodiment, training framework 1004 may be used to perform incremental learning, such as through transferred learning techniques. In at least one embodiment, incremental learning enables trained neural network 1008 to adapt to new dataset 1012 without forgetting knowledge instilled within trained neural network 1008 during initial training.

DATA CENTER

[0293] FIG. 11 illustrates an example data center 1100, in which at least one embodiment may be used. In at least one embodiment, data center 1100 includes a data center

layer 1140.

[0294] In at least one embodiment, as shown in FIG. 11, data center infrastructure layer 1110 may include a resource orchestrator 1112, grouped computing resources 1114, and node computing resources (“node C.R.s”) 1116(1)-1116(N), where “N” represents a positive integer (which may be a different integer “N” than used in other figures). In at least one embodiment, node C.R.s 1116(1)-1116(N) may include, but are not limited to, any number of central processing units (“CPUs”) or other processors (including accelerators, field programmable gate arrays (FPGAs), graphics processors, etc.), memory storage devices 1118(1)-1118(N) (e.g., dynamic read-only memory, solid state storage or disk drives), network input/output (“NW I/O”) devices, network switches, virtual machines (“VMs”), power modules, and cooling modules, etc. In at least one embodiment, one or more node C.R.s from among node C.R.s 1116(1)-1116(N) may be a server having one or more of above-mentioned computing resources.

[0295] In at least one embodiment, grouped computing resources 1114 may include separate groupings of node C.R.s housed within one or more racks (not shown), or many racks housed in data centers at various geographical locations (also not shown). In at least one embodiment, separate groupings of node C.R.s within grouped computing resources 1114 may include grouped compute, network, memory or storage resources that may be configured or allocated to support one or more workloads. In at least one embodiment, several node C.R.s including CPUs or processors may grouped within one or more racks to provide compute resources to support one or more workloads. In at least one embodiment, one or more racks may also include any number of power modules, cooling modules, and network switches, in any combination.

[0296] In at least one embodiment, resource orchestrator 1112 may configure or otherwise control one or more node C.R.s 1116(1)-1116(N) and/or grouped computing resources 1114. In at least one embodiment, resource orchestrator 1112 may include a software design infrastructure (“SDI”) management entity for data center 1100. In at least one embodiment, resource orchestrator 912 may include hardware, software or some combination thereof.

[0297] In at least one embodiment, as shown in FIG. 11, framework layer 1120 includes a job scheduler 1122, a configuration manager 1124, a resource manager 1126 and a distributed file system 1128. In at least one embodiment, framework layer 1120 may include a

1142 of application layer 1140. In at least one embodiment, software 1132 or application(s) 1142 may respectively include web-based service software or applications, such as those provided by Amazon Web Services, Google Cloud and Microsoft Azure. In at least one
5 embodiment, framework layer 1120 may be, but is not limited to, a type of free and open-source software web application framework such as Apache Spark™ (hereinafter “Spark”) that may utilize distributed file system 1128 for large-scale data processing (e.g., “big data”). In at least one embodiment, job scheduler 1122 may include a Spark driver to facilitate scheduling of workloads supported by various layers of data center 1100. In at least one
10 embodiment, configuration manager 1124 may be capable of configuring different layers such as software layer 1130 and framework layer 1120 including Spark and distributed file system 1128 for supporting large-scale data processing. In at least one embodiment, resource manager 1126 may be capable of managing clustered or grouped computing resources mapped to or allocated for support of distributed file system 1128 and job scheduler 1122. In
15 at least one embodiment, clustered or grouped computing resources may include grouped computing resources 1114 at data center infrastructure layer 1110. In at least one embodiment, resource manager 1126 may coordinate with resource orchestrator 1112 to manage these mapped or allocated computing resources.

[0298] In at least one embodiment, software 1132 included in software layer 1130 may
20 include software used by at least portions of node C.R.s 1116(1)-1116(N), grouped computing resources 1114, and/or distributed file system 1128 of framework layer 1120. In at least one embodiment, one or more types of software may include, but are not limited to, Internet web page search software, e-mail virus scan software, database software, and streaming video content software.

[0299] In at least one embodiment, application(s) 1142 included in application layer 1140
25 may include one or more types of applications used by at least portions of node C.R.s 1116(1)-1116(N), grouped computing resources 1114, and/or distributed file system 1128 of framework layer 1120. In at least one embodiment, one or more types of applications may include, but are not limited to, any number of a genomics application, a cognitive compute,
30 application and a machine learning application, including training or inferencing software, machine learning framework software (e.g., PyTorch, TensorFlow, Caffe, etc.) or other machine learning applications used in conjunction with one or more embodiments.

1126, and resource orchestrator 1112 may implement any number and type of self-modifying actions based on any amount and type of data acquired in any technically feasible fashion. In at least one embodiment, self-modifying actions may relieve a data center operator of data center 1100 from making possibly bad configuration decisions and possibly avoiding underutilized and/or poor performing portions of a data center.

[0301] In at least one embodiment, data center 1100 may include tools, services, software or other resources to train one or more machine learning models or predict or infer information using one or more machine learning models according to one or more embodiments described herein. For example, in at least one embodiment, a machine learning model may be trained by calculating weight parameters according to a neural network architecture using software and computing resources described above with respect to data center 1100. In at least one embodiment, trained machine learning models corresponding to one or more neural networks may be used to infer or predict information using resources described above with respect to data center 1100 by using weight parameters calculated through one or more training techniques described herein.

[0302] In at least one embodiment, data center may use CPUs, application-specific integrated circuits (ASICs), GPUs, FPGAs, or other hardware to perform training and/or inferencing using above-described resources. Moreover, one or more software and/or hardware resources described above may be configured as a service to allow users to train or performing inferencing of information, such as image recognition, speech recognition, or other artificial intelligence services.

[0303] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 11 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

AUTONOMOUS VEHICLE

[0304] FIG. 12A illustrates an example of an autonomous vehicle 1200, according to at least one embodiment. In at least one embodiment, autonomous vehicle 1200 (alternatively

a car, a truck, a bus, and/or another type of vehicle that accommodates one or more passengers. In at least one embodiment, vehicle 1200 may be a semi-tractor-trailer truck used for hauling cargo. In at least one embodiment, vehicle 1200 may be an airplane, robotic vehicle, or other kind of vehicle.

[0305] Autonomous vehicles may be described in terms of automation levels, defined by National Highway Traffic Safety Administration (“NHTSA”), a division of US Department of Transportation, and Society of Automotive Engineers (“SAE”) “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles” (e.g., Standard No. J3016-201806, published on June 15, 2018, Standard No. J3016-201609, published on September 30, 2016, and previous and future versions of this standard). In at least one embodiment, vehicle 1200 may be capable of functionality in accordance with one or more of Level 1 through Level 5 of autonomous driving levels. For example, in at least one embodiment, vehicle 1200 may be capable of conditional automation (Level 3), high automation (Level 4), and/or full automation (Level 5), depending on embodiment.

[0306] In at least one embodiment, vehicle 1200 may include, without limitation, components such as a chassis, a vehicle body, wheels (e.g., 2, 4, 6, 8, 18, etc.), tires, axles, and other components of a vehicle. In at least one embodiment, vehicle 1200 may include, without limitation, a propulsion system 1250, such as an internal combustion engine, hybrid electric power plant, an all-electric engine, and/or another propulsion system type. In at least one embodiment, propulsion system 1250 may be connected to a drive train of vehicle 1200, which may include, without limitation, a transmission, to enable propulsion of vehicle 1200. In at least one embodiment, propulsion system 1250 may be controlled in response to receiving signals from a throttle/accelerator(s) 1252.

[0307] In at least one embodiment, a steering system 1254, which may include, without limitation, a steering wheel, is used to steer vehicle 1200 (e.g., along a desired path or route) when propulsion system 1250 is operating (e.g., when vehicle 1200 is in motion). In at least one embodiment, steering system 1254 may receive signals from steering actuator(s) 1256. In at least one embodiment, a steering wheel may be optional for full automation (Level 5) functionality. In at least one embodiment, a brake sensor system 1246 may be used to operate vehicle brakes in response to receiving signals from brake actuator(s) 1248 and/or brake sensors.

limitation, one or more system on chips (“SoCs”) (not shown in FIG. 12A) and/or graphics processing unit(s) (“GPU(s)”), provide signals (e.g., representative of commands) to one or more components and/or systems of vehicle 1200. For instance, in at least one embodiment, controller(s) 1236 may send signals to operate vehicle brakes via brake actuator(s) 1248, to operate steering system 1254 via steering actuator(s) 1256, to operate propulsion system 1250 via throttle/accelerator(s) 1252. In at least one embodiment, controller(s) 1236 may include one or more onboard (e.g., integrated) computing devices that process sensor signals, and output operation commands (e.g., signals representing commands) to enable autonomous driving and/or to assist a human driver in driving vehicle 1200. In at least one embodiment, controller(s) 1236 may include a first controller for autonomous driving functions, a second controller for functional safety functions, a third controller for artificial intelligence functionality (e.g., computer vision), a fourth controller for infotainment functionality, a fifth controller for redundancy in emergency conditions, and/or other controllers. In at least one embodiment, a single controller may handle two or more of above functionalities, two or more controllers may handle a single functionality, and/or any combination thereof.

[0309] In at least one embodiment, controller(s) 1236 provide signals for controlling one or more components and/or systems of vehicle 1200 in response to sensor data received from one or more sensors (e.g., sensor inputs). In at least one embodiment, sensor data may be received from, for example and without limitation, global navigation satellite systems (“GNSS”) sensor(s) 1258 (e.g., Global Positioning System sensor(s)), RADAR sensor(s) 1260, ultrasonic sensor(s) 1262, LIDAR sensor(s) 1264, inertial measurement unit (“IMU”) sensor(s) 1266 (e.g., accelerometer(s), gyroscope(s), a magnetic compass or magnetic compasses, magnetometer(s), etc.), microphone(s) 1296, stereo camera(s) 1268, wide-view camera(s) 1270 (e.g., fisheye cameras), infrared camera(s) 1272, surround camera(s) 1274 (e.g., 360 degree cameras), long-range cameras (not shown in Figure 12A), mid-range camera(s) (not shown in Figure 12A), speed sensor(s) 1244 (e.g., for measuring speed of vehicle 1200), vibration sensor(s) 1242, steering sensor(s) 1240, brake sensor(s) (e.g., as part of brake sensor system 1246), and/or other sensor types.

[0310] In at least one embodiment, one or more of controller(s) 1236 may receive inputs (e.g., represented by input data) from an instrument cluster 1232 of vehicle 1200 and provide outputs (e.g., represented by output data, display data, etc.) via a human-machine interface (“HMI”) display 1234, an audible annunciator, a loudspeaker, and/or via other components of

velocity, speed, time, map data (e.g., a High Definition map (not shown in FIG. 12A)), location data (e.g., vehicle's 1200 location, such as on a map), direction, location of other vehicles (e.g., an occupancy grid), information about objects and status of objects as perceived by controller(s) 1236, etc. For example, in at least one embodiment, HMI display 1234 may display information about presence of one or more objects (e.g., a street sign, caution sign, traffic light changing, etc.), and/or information about driving maneuvers vehicle has made, is making, or will make (e.g., changing lanes now, taking exit 34B in two miles, etc.).

[0311] In at least one embodiment, vehicle 1200 further includes a network interface 1224 which may use wireless antenna(s) 1226 and/or modem(s) to communicate over one or more networks. For example, in at least one embodiment, network interface 1224 may be capable of communication over Long-Term Evolution ("LTE"), Wideband Code Division Multiple Access ("WCDMA"), Universal Mobile Telecommunications System ("UMTS"), Global System for Mobile communication ("GSM"), IMT-CDMA Multi-Carrier ("CDMA2000") networks, etc. In at least one embodiment, wireless antenna(s) 1226 may also enable communication between objects in environment (e.g., vehicles, mobile devices, etc.), using local area network(s), such as Bluetooth, Bluetooth Low Energy ("LE"), Z-Wave, ZigBee, etc., and/or low power wide-area network(s) ("LPWANs"), such as LoRaWAN, SigFox, etc. protocols.

[0312] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 12A for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0313] FIG. 12B illustrates an example of camera locations and fields of view for autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, cameras and respective fields of view are one example embodiment and are not intended to be limiting. For instance, in at least one embodiment, additional and/or

vehicle 1200.

[0314] In at least one embodiment, camera types for cameras may include, but are not limited to, digital cameras that may be adapted for use with components and/or systems of vehicle 1200. In at least one embodiment, camera(s) may operate at automotive safety integrity level (“ASIL”) B and/or at another ASIL. In at least one embodiment, camera types may be capable of any image capture rate, such as 60 frames per second (fps), 1220 fps, 240 fps, etc., depending on embodiment. In at least one embodiment, cameras may be capable of using rolling shutters, global shutters, another type of shutter, or a combination thereof. In at least one embodiment, color filter array may include a red clear clear clear (“RCCC”) color filter array, a red clear clear blue (“RCCB”) color filter array, a red blue green clear (“RBGC”) color filter array, a Foveon X3 color filter array, a Bayer sensors (“RGGB”) color filter array, a monochrome sensor color filter array, and/or another type of color filter array. In at least one embodiment, clear pixel cameras, such as cameras with an RCCC, an RCCB, and/or an RBGC color filter array, may be used in an effort to increase light sensitivity.

[0315] In at least one embodiment, one or more of camera(s) may be used to perform advanced driver assistance systems (“ADAS”) functions (e.g., as part of a redundant or fail-safe design). For example, in at least one embodiment, a Multi-Function Mono Camera may be installed to provide functions including lane departure warning, traffic sign assist and intelligent headlamp control. In at least one embodiment, one or more of camera(s) (e.g., all cameras) may record and provide image data (e.g., video) simultaneously.

[0316] In at least one embodiment, one or more camera may be mounted in a mounting assembly, such as a custom designed (three-dimensional (“3D”) printed) assembly, in order to cut out stray light and reflections from within vehicle 1200 (e.g., reflections from dashboard reflected in windshield mirrors) which may interfere with camera image data capture abilities. With reference to wing-mirror mounting assemblies, in at least one embodiment, wing-mirror assemblies may be custom 3D printed so that a camera mounting plate matches a shape of a wing-mirror. In at least one embodiment, camera(s) may be integrated into wing-mirrors. In at least one embodiment, for side-view cameras, camera(s) may also be integrated within four pillars at each corner of a cabin.

[0317] In at least one embodiment, cameras with a field of view that include portions of an environment in front of vehicle 1200 (e.g., front-facing cameras) may be used for surround

or more of controller(s) 1236 and/or control SoCs, providing information critical to generating an occupancy grid and/or determining preferred vehicle paths. In at least one embodiment, front-facing cameras may be used to perform many similar ADAS functions as LIDAR, including, without limitation, emergency braking, pedestrian detection, and collision avoidance. In at least one embodiment, front-facing cameras may also be used for ADAS functions and systems including, without limitation, Lane Departure Warnings (“LDW”), Autonomous Cruise Control (“ACC”), and/or other functions such as traffic sign recognition.

[0318] In at least one embodiment, a variety of cameras may be used in a front-facing configuration, including, for example, a monocular camera platform that includes a CMOS (“complementary metal oxide semiconductor”) color imager. In at least one embodiment, a wide-view camera 1270 may be used to perceive objects coming into view from a periphery (e.g., pedestrians, crossing traffic or bicycles). Although only one wide-view camera 1270 is illustrated in FIG. 12B, in other embodiments, there may be any number (including zero) wide-view cameras on vehicle 1200. In at least one embodiment, any number of long-range camera(s) 1298 (e.g., a long-view stereo camera pair) may be used for depth-based object detection, especially for objects for which a neural network has not yet been trained. In at least one embodiment, long-range camera(s) 1298 may also be used for object detection and classification, as well as basic object tracking.

[0319] In at least one embodiment, any number of stereo camera(s) 1268 may also be included in a front-facing configuration. In at least one embodiment, one or more of stereo camera(s) 1268 may include an integrated control unit comprising a scalable processing unit, which may provide a programmable logic (“FPGA”) and a multi-core micro-processor with an integrated Controller Area Network (“CAN”) or Ethernet interface on a single chip. In at least one embodiment, such a unit may be used to generate a 3D map of an environment of vehicle 1200, including a distance estimate for all points in an image. In at least one embodiment, one or more of stereo camera(s) 1268 may include, without limitation, compact stereo vision sensor(s) that may include, without limitation, two camera lenses (one each on left and right) and an image processing chip that may measure distance from vehicle 1200 to target object and use generated information (e.g., metadata) to activate autonomous emergency braking and lane departure warning functions. In at least one embodiment, other types of stereo camera(s) 1268 may be used in addition to, or alternatively from, those described herein.

environment to sides of vehicle 1200 (e.g., side-view cameras) may be used for surround view, providing information used to create and update an occupancy grid, as well as to generate side impact collision warnings. For example, in at least one embodiment, surround camera(s) 1274 (e.g., four surround cameras as illustrated in FIG. 12B) could be positioned on vehicle 1200. In at least one embodiment, surround camera(s) 1274 may include, without limitation, any number and combination of wide-view cameras, fisheye camera(s), 360 degree camera(s), and/or similar cameras. For instance, in at least one embodiment, four fisheye cameras may be positioned on a front, a rear, and sides of vehicle 1200. In at least one embodiment, vehicle 1200 may use three surround camera(s) 1274 (e.g., left, right, and rear), and may leverage one or more other camera(s) (e.g., a forward-facing camera) as a fourth surround-view camera.

[0321] In at least one embodiment, cameras with a field of view that include portions of an environment behind vehicle 1200 (e.g., rear-view cameras) may be used for parking assistance, surround view, rear collision warnings, and creating and updating an occupancy grid. In at least one embodiment, a wide variety of cameras may be used including, but not limited to, cameras that are also suitable as a front-facing camera(s) (e.g., long-range cameras 1298 and/or mid-range camera(s) 1276, stereo camera(s) 1268), infrared camera(s) 1272, etc., as described herein.

[0322] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 12B for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0323] FIG. 12C is a block diagram illustrating an example system architecture for autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, each of components, features, and systems of vehicle 1200 in FIG. 12C is illustrated as being connected via a bus 1202. In at least one embodiment, bus 1202 may include, without limitation, a CAN data interface (alternatively referred to herein as a “CAN bus”). In at least one embodiment, a CAN may be a network inside vehicle 1200 used to aid

acceleration, braking, steering, windshield wipers, etc. In at least one embodiment, bus 1202 may be configured to have dozens or even hundreds of nodes, each with its own unique identifier (e.g., a CAN ID). In at least one embodiment, bus 1202 may be read to find steering wheel angle, ground speed, engine revolutions per minute (“RPMs”), button positions, and/or other vehicle status indicators. In at least one embodiment, bus 1202 may be a CAN bus that is ASIL B compliant.

[0324] In at least one embodiment, in addition to, or alternatively from CAN, FlexRay and/or Ethernet protocols may be used. In at least one embodiment, there may be any number of busses forming bus 1202, which may include, without limitation, zero or more CAN busses, zero or more FlexRay busses, zero or more Ethernet busses, and/or zero or more other types of busses using different protocols. In at least one embodiment, two or more busses may be used to perform different functions, and/or may be used for redundancy. For example, a first bus may be used for collision avoidance functionality and a second bus may be used for actuation control. In at least one embodiment, each bus of bus 1202 may communicate with any of components of vehicle 1200, and two or more busses of bus 1202 may communicate with corresponding components. In at least one embodiment, each of any number of system(s) on chip(s) (“SoC(s)”) 1204 (such as SoC 1204(A) and SoC 1204(B)), each of controller(s) 1236, and/or each computer within vehicle may have access to same input data (e.g., inputs from sensors of vehicle 1200), and may be connected to a common bus, such as CAN bus.

[0325] In at least one embodiment, vehicle 1200 may include one or more controller(s) 1236, such as those described herein with respect to FIG. 12A. In at least one embodiment, controller(s) 1236 may be used for a variety of functions. In at least one embodiment, controller(s) 1236 may be coupled to any of various other components and systems of vehicle 1200, and may be used for control of vehicle 1200, artificial intelligence of vehicle 1200, infotainment for vehicle 1200, and/or other functions.

[0326] In at least one embodiment, vehicle 1200 may include any number of SoCs 1204. In at least one embodiment, each of SoCs 1204 may include, without limitation, central processing units (“CPU(s)”) 1206, graphics processing units (“GPU(s)”) 1208, processor(s) 1210, cache(s) 1212, accelerator(s) 1214, data store(s) 1216, and/or other components and features not illustrated. In at least one embodiment, SoC(s) 1204 may be used to control

SoC(s) 1204 may be combined in a system (e.g., system of vehicle 1200) with a High Definition (“HD”) map 1222 which may obtain map refreshes and/or updates via network interface 1224 from one or more servers (not shown in Figure 12C).

5 [0327] In at least one embodiment, CPU(s) 1206 may include a CPU cluster or CPU complex (alternatively referred to herein as a “CCPLEX”). In at least one embodiment, CPU(s) 1206 may include multiple cores and/or level two (“L2”) caches. For instance, in at least one embodiment, CPU(s) 1206 may include eight cores in a coherent multi-processor configuration. In at least one embodiment, CPU(s) 1206 may include four dual-core clusters
10 where each cluster has a dedicated L2 cache (e.g., a 2 megabyte (MB) L2 cache). In at least one embodiment, CPU(s) 1206 (e.g., CCPLEX) may be configured to support simultaneous cluster operations enabling any combination of clusters of CPU(s) 1206 to be active at any given time.

[0328] In at least one embodiment, one or more of CPU(s) 1206 may implement power
15 management capabilities that include, without limitation, one or more of following features: individual hardware blocks may be clock-gated automatically when idle to save dynamic power; each core clock may be gated when such core is not actively executing instructions due to execution of Wait for Interrupt (“WFI”)/Wait for Event (“WFE”) instructions; each core may be independently power-gated; each core cluster may be independently clock-gated
20 when all cores are clock-gated or power-gated; and/or each core cluster may be independently power-gated when all cores are power-gated. In at least one embodiment, CPU(s) 1206 may further implement an enhanced algorithm for managing power states, where allowed power states and expected wakeup times are specified, and hardware/microcode determines which best power state to enter for core, cluster, and
25 CCPLEX. In at least one embodiment, processing cores may support simplified power state entry sequences in software with work offloaded to microcode.

[0329] In at least one embodiment, GPU(s) 1208 may include an integrated GPU (alternatively referred to herein as an “iGPU”). In at least one embodiment, GPU(s) 1208 may be programmable and may be efficient for parallel workloads. In at least one
30 embodiment, GPU(s) 1208 may use an enhanced tensor instruction set. In at least one embodiment, GPU(s) 1208 may include one or more streaming microprocessors, where each streaming microprocessor may include a level one (“L1”) cache (e.g., an L1 cache with at

cache (e.g., an L2 cache with a 512 KB storage capacity). In at least one embodiment, GPU(s) 1208 may include at least eight streaming microprocessors. In at least one embodiment, GPU(s) 1208 may use compute application programming interface(s) (API(s)).

- 5 In at least one embodiment, GPU(s) 1208 may use one or more parallel computing platforms and/or programming models (e.g., NVIDIA's CUDA model).

[0330] In at least one embodiment, one or more of GPU(s) 1208 may be power-optimized for best performance in automotive and embedded use cases. For example, in at least one embodiment, GPU(s) 1208 could be fabricated on Fin field-effect transistor ("FinFET")
10 circuitry. In at least one embodiment, each streaming microprocessor may incorporate a number of mixed-precision processing cores partitioned into multiple blocks. For example, and without limitation, 64 PF32 cores and 32 PF64 cores could be partitioned into four processing blocks. In at least one embodiment, each processing block could be allocated 16 FP32 cores, 8 FP64 cores, 16 INT32 cores, two mixed-precision NVIDIA Tensor cores for
15 deep learning matrix arithmetic, a level zero ("L0") instruction cache, a warp scheduler, a dispatch unit, and/or a 64 KB register file. In at least one embodiment, streaming microprocessors may include independent parallel integer and floating-point data paths to provide for efficient execution of workloads with a mix of computation and addressing calculations. In at least one embodiment, streaming microprocessors may include
20 independent thread scheduling capability to enable finer-grain synchronization and cooperation between parallel threads. In at least one embodiment, streaming microprocessors may include a combined L1 data cache and shared memory unit in order to improve performance while simplifying programming.

[0331] In at least one embodiment, one or more of GPU(s) 1208 may include a high
25 bandwidth memory ("HBM") and/or a 16 GB HBM2 memory subsystem to provide, in some examples, about 900 GB/second peak memory bandwidth. In at least one embodiment, in addition to, or alternatively from, HBM memory, a synchronous graphics random-access memory ("SGRAM") may be used, such as a graphics double data rate type five synchronous random-access memory ("GDDR5").

30 [0332] In at least one embodiment, GPU(s) 1208 may include unified memory technology. In at least one embodiment, address translation services ("ATS") support may be used to allow GPU(s) 1208 to access CPU(s) 1206 page tables directly. In at least one embodiment,

a miss, an address translation request may be transmitted to CPU(s) 1206. In response, 2 CPU of CPU(s) 1206 may look in its page tables for a virtual-to-physical mapping for an address and transmit translation back to GPU(s) 1208, in at least one embodiment. In at least
5 one embodiment, unified memory technology may allow a single unified virtual address space for memory of both CPU(s) 1206 and GPU(s) 1208, thereby simplifying GPU(s) 1208 programming and porting of applications to GPU(s) 1208.

[0333] In at least one embodiment, GPU(s) 1208 may include any number of access
counters that may keep track of frequency of access of GPU(s) 1208 to memory of other
10 processors. In at least one embodiment, access counter(s) may help ensure that memory pages are moved to physical memory of a processor that is accessing pages most frequently, thereby improving efficiency for memory ranges shared between processors.

[0334] In at least one embodiment, one or more of SoC(s) 1204 may include any number of
cache(s) 1212, including those described herein. For example, in at least one embodiment,
15 cache(s) 1212 could include a level three (“L3”) cache that is available to both CPU(s) 1206 and GPU(s) 1208 (e.g., that is connected to CPU(s) 1206 and GPU(s) 1208). In at least one embodiment, cache(s) 1212 may include a write-back cache that may keep track of states of lines, such as by using a cache coherence protocol (e.g., MEI, MESI, MSI, etc.). In at least one embodiment, a L3 cache may include 4 MB of memory or more, depending on
20 embodiment, although smaller cache sizes may be used.

[0335] In at least one embodiment, one or more of SoC(s) 1204 may include one or more
accelerator(s) 1214 (e.g., hardware accelerators, software accelerators, or a combination thereof). In at least one embodiment, SoC(s) 1204 may include a hardware acceleration cluster that may include optimized hardware accelerators and/or large on-chip memory. In at
25 least one embodiment, large on-chip memory (e.g., 4 MB of SRAM), may enable a hardware acceleration cluster to accelerate neural networks and other calculations. In at least one embodiment, a hardware acceleration cluster may be used to complement GPU(s) 1208 and to off-load some of tasks of GPU(s) 1208 (e.g., to free up more cycles of GPU(s) 1208 for performing other tasks). In at least one embodiment, accelerator(s) 1214 could be used for
30 targeted workloads (e.g., perception, convolutional neural networks (“CNNs”), recurrent neural networks (“RNNs”), etc.) that are stable enough to be amenable to acceleration. In at least one embodiment, a CNN may include a region-based or regional convolutional neural

CNN.

[0336] In at least one embodiment, accelerator(s) 1214 (e.g., hardware acceleration cluster) may include one or more deep learning accelerator (“DLA”). In at least one embodiment, DLA(s) may include, without limitation, one or more Tensor processing units (“TPUs”) that may be configured to provide an additional ten trillion operations per second for deep learning applications and inferencing. In at least one embodiment, TPUs may be accelerators configured to, and optimized for, performing image processing functions (e.g., for CNNs, RCNNs, etc.). In at least one embodiment, DLA(s) may further be optimized for a specific set of neural network types and floating point operations, as well as inferencing. In at least one embodiment, design of DLA(s) may provide more performance per millimeter than a typical general-purpose GPU, and typically vastly exceeds performance of a CPU. In at least one embodiment, TPU(s) may perform several functions, including a single-instance convolution function, supporting, for example, INT8, INT16, and FP16 data types for both features and weights, as well as post-processor functions. In at least one embodiment, DLA(s) may quickly and efficiently execute neural networks, especially CNNs, on processed or unprocessed data for any of a variety of functions, including, for example and without limitation: a CNN for object identification and detection using data from camera sensors; a CNN for distance estimation using data from camera sensors; a CNN for emergency vehicle detection and identification and detection using data from microphones; a CNN for facial recognition and vehicle owner identification using data from camera sensors; and/or a CNN for security and/or safety related events.

[0337] In at least one embodiment, DLA(s) may perform any function of GPU(s) 1208, and by using an inference accelerator, for example, a designer may target either DLA(s) or GPU(s) 1208 for any function. For example, in at least one embodiment, a designer may focus processing of CNNs and floating point operations on DLA(s) and leave other functions to GPU(s) 1208 and/or accelerator(s) 1214.

[0338] In at least one embodiment, accelerator(s) 1214 may include programmable vision accelerator (“PVA”), which may alternatively be referred to herein as a computer vision accelerator. In at least one embodiment, PVA may be designed and configured to accelerate computer vision algorithms for advanced driver assistance system (“ADAS”) 1238, autonomous driving, augmented reality (“AR”) applications, and/or virtual reality (“VR”)

and flexibility. For example, in at least one embodiment, each PVA may include, for example and without limitation, any number of reduced instruction set computer (“RISC”) cores, direct memory access (“DMA”), and/or any number of vector processors.

5 [0339] In at least one embodiment, RISC cores may interact with image sensors (e.g., image sensors of any cameras described herein), image signal processor(s), etc. In at least one embodiment, each RISC core may include any amount of memory. In at least one embodiment, RISC cores may use any of a number of protocols, depending on embodiment. In at least one embodiment, RISC cores may execute a real-time operating system (“RTOS”).
10 In at least one embodiment, RISC cores may be implemented using one or more integrated circuit devices, application specific integrated circuits (“ASICs”), and/or memory devices. For example, in at least one embodiment, RISC cores could include an instruction cache and/or a tightly coupled RAM.

[0340] In at least one embodiment, DMA may enable components of PVA to access system
15 memory independently of CPU(s) 1206. In at least one embodiment, DMA may support any number of features used to provide optimization to a PVA including, but not limited to, supporting multi-dimensional addressing and/or circular addressing. In at least one embodiment, DMA may support up to six or more dimensions of addressing, which may include, without limitation, block width, block height, block depth, horizontal block stepping,
20 vertical block stepping, and/or depth stepping.

[0341] In at least one embodiment, vector processors may be programmable processors that may be designed to efficiently and flexibly execute programming for computer vision algorithms and provide signal processing capabilities. In at least one embodiment, a PVA may include a PVA core and two vector processing subsystem partitions. In at least one
25 embodiment, a PVA core may include a processor subsystem, DMA engine(s) (e.g., two DMA engines), and/or other peripherals. In at least one embodiment, a vector processing subsystem may operate as a primary processing engine of a PVA, and may include a vector processing unit (“VPU”), an instruction cache, and/or vector memory (e.g., “VMEM”). In at least one embodiment, VPU core may include a digital signal processor such as, for example,
30 a single instruction, multiple data (“SIMD”), very long instruction word (“VLIW”) digital signal processor. In at least one embodiment, a combination of SIMD and VLIW may enhance throughput and speed.

cache and may be coupled to dedicated memory. As a result, in at least one embodiment, each of vector processors may be configured to execute independently of other vector processors. In at least one embodiment, vector processors that are included in a particular PVA may be configured to employ data parallelism. For instance, in at least one embodiment, plurality of vector processors included in a single PVA may execute a common computer vision algorithm, but on different regions of an image. In at least one embodiment, vector processors included in a particular PVA may simultaneously execute different computer vision algorithms, on one image, or even execute different algorithms on sequential images or portions of an image. In at least one embodiment, among other things, any number of PVAs may be included in hardware acceleration cluster and any number of vector processors may be included in each PVA. In at least one embodiment, PVA may include additional error correcting code (“ECC”) memory, to enhance overall system safety.

[0343] In at least one embodiment, accelerator(s) 1214 may include a computer vision network on-chip and static random-access memory (“SRAM”), for providing a high-bandwidth, low latency SRAM for accelerator(s) 1214. In at least one embodiment, on-chip memory may include at least 4 MB SRAM, comprising, for example and without limitation, eight field-configurable memory blocks, that may be accessible by both a PVA and a DLA. In at least one embodiment, each pair of memory blocks may include an advanced peripheral bus (“APB”) interface, configuration circuitry, a controller, and a multiplexer. In at least one embodiment, any type of memory may be used. In at least one embodiment, a PVA and a DLA may access memory via a backbone that provides a PVA and a DLA with high-speed access to memory. In at least one embodiment, a backbone may include a computer vision network on-chip that interconnects a PVA and a DLA to memory (e.g., using APB).

[0344] In at least one embodiment, a computer vision network on-chip may include an interface that determines, before transmission of any control signal/address/data, that both a PVA and a DLA provide ready and valid signals. In at least one embodiment, an interface may provide for separate phases and separate channels for transmitting control signals/addresses/data, as well as burst-type communications for continuous data transfer. In at least one embodiment, an interface may comply with International Organization for Standardization (“ISO”) 26262 or International Electrotechnical Commission (“IEC”) 61508 standards, although other standards and protocols may be used.

ray-tracing hardware accelerator. In at least one embodiment, real-time ray-tracing hardware accelerator may be used to quickly and efficiently determine positions and extents of objects (e.g., within a world model), to generate real-time visualization simulations, for RADAR
5 signal interpretation, for sound propagation synthesis and/or analysis, for simulation of SONAR systems, for general wave propagation simulation, for comparison to LIDAR data for purposes of localization and/or other functions, and/or for other uses.

[0346] In at least one embodiment, accelerator(s) 1214 can have a wide array of uses for autonomous driving. In at least one embodiment, a PVA may be used for key processing
10 stages in ADAS and autonomous vehicles. In at least one embodiment, a PVA's capabilities are a good match for algorithmic domains needing predictable processing, at low power and low latency. In other words, a PVA performs well on semi-dense or dense regular computation, even on small data sets, which might require predictable run-times with low latency and low power. In at least one embodiment, such as in vehicle 1200, PVAs might be
15 designed to run classic computer vision algorithms, as they can be efficient at object detection and operating on integer math.

[0347] For example, according to at least one embodiment of technology, a PVA is used to perform computer stereo vision. In at least one embodiment, a semi-global matching-based algorithm may be used in some examples, although this is not intended to be limiting. In at
20 least one embodiment, applications for Level 3-5 autonomous driving use motion estimation/stereo matching on-the-fly (e.g., structure from motion, pedestrian recognition, lane detection, etc.). In at least one embodiment, a PVA may perform computer stereo vision functions on inputs from two monocular cameras.

[0348] In at least one embodiment, a PVA may be used to perform dense optical flow. For
25 example, in at least one embodiment, a PVA could process raw RADAR data (e.g., using a 4D Fast Fourier Transform) to provide processed RADAR data. In at least one embodiment, a PVA is used for time of flight depth processing, by processing raw time of flight data to provide processed time of flight data, for example.

[0349] In at least one embodiment, a DLA may be used to run any type of network to
30 enhance control and driving safety, including for example and without limitation, a neural network that outputs a measure of confidence for each object detection. In at least one embodiment, confidence may be represented or interpreted as a probability, or as providing a

a confidence measure enables a system to make further decisions regarding which detections should be considered as true positive detections rather than false positive detections. In at least one embodiment, a system may set a threshold value for confidence and consider only

5 detections exceeding threshold value as true positive detections. In an embodiment in which an automatic emergency braking (“AEB”) system is used, false positive detections would cause vehicle to automatically perform emergency braking, which is obviously undesirable.

In at least one embodiment, highly confident detections may be considered as triggers for AEB. In at least one embodiment, a DLA may run a neural network for regressing

10 confidence value. In at least one embodiment, neural network may take as its input at least some subset of parameters, such as bounding box dimensions, ground plane estimate obtained (e.g., from another subsystem), output from IMU sensor(s) 1266 that correlates with vehicle 1200 orientation, distance, 3D location estimates of object obtained from neural network and/or other sensors (e.g., LIDAR sensor(s) 1264 or RADAR sensor(s) 1260), among others.

15 [0350] In at least one embodiment, one or more of SoC(s) 1204 may include data store(s) 1216 (e.g., memory). In at least one embodiment, data store(s) 1216 may be on-chip memory of SoC(s) 1204, which may store neural networks to be executed on GPU(s) 1208 and/or a DLA. In at least one embodiment, data store(s) 1216 may be large enough in capacity to store multiple instances of neural networks for redundancy and safety. In at least one

20 embodiment, data store(s) 1216 may comprise L2 or L3 cache(s).

[0351] In at least one embodiment, one or more of SoC(s) 1204 may include any number of processor(s) 1210 (e.g., embedded processors). In at least one embodiment, processor(s) 1210 may include a boot and power management processor that may be a dedicated processor and subsystem to handle boot power and management functions and related security

25 enforcement. In at least one embodiment, a boot and power management processor may be a part of a boot sequence of SoC(s) 1204 and may provide runtime power management services. In at least one embodiment, a boot power and management processor may provide clock and voltage programming, assistance in system low power state transitions, management of SoC(s) 1204 thermals and temperature sensors, and/or management of SoC(s)

30 1204 power states. In at least one embodiment, each temperature sensor may be implemented as a ring-oscillator whose output frequency is proportional to temperature, and SoC(s) 1204 may use ring-oscillators to detect temperatures of CPU(s) 1206, GPU(s) 1208, and/or accelerator(s) 1214. In at least one embodiment, if temperatures are determined to exceed a

routine and put SoC(s) 1204 into a lower power state and/or put vehicle 1200 into a chauffeur to safe stop mode (e.g., bring vehicle 1200 to a safe stop).

[0352] In at least one embodiment, processor(s) 1210 may further include a set of embedded processors that may serve as an audio processing engine which may be an audio subsystem that enables full hardware support for multi-channel audio over multiple interfaces, and a broad and flexible range of audio I/O interfaces. In at least one embodiment, an audio processing engine is a dedicated processor core with a digital signal processor with dedicated RAM.

[0353] In at least one embodiment, processor(s) 1210 may further include an always-on processor engine that may provide necessary hardware features to support low power sensor management and wake use cases. In at least one embodiment, an always-on processor engine may include, without limitation, a processor core, a tightly coupled RAM, supporting peripherals (e.g., timers and interrupt controllers), various I/O controller peripherals, and routing logic.

[0354] In at least one embodiment, processor(s) 1210 may further include a safety cluster engine that includes, without limitation, a dedicated processor subsystem to handle safety management for automotive applications. In at least one embodiment, a safety cluster engine may include, without limitation, two or more processor cores, a tightly coupled RAM, support peripherals (e.g., timers, an interrupt controller, etc.), and/or routing logic. In a safety mode, two or more cores may operate, in at least one embodiment, in a lockstep mode and function as a single core with comparison logic to detect any differences between their operations. In at least one embodiment, processor(s) 1210 may further include a real-time camera engine that may include, without limitation, a dedicated processor subsystem for handling real-time camera management. In at least one embodiment, processor(s) 1210 may further include a high-dynamic range signal processor that may include, without limitation, an image signal processor that is a hardware engine that is part of a camera processing pipeline.

[0355] In at least one embodiment, processor(s) 1210 may include a video image compositor that may be a processing block (e.g., implemented on a microprocessor) that implements video post-processing functions needed by a video playback application to produce a final image for a player window. In at least one embodiment, a video image

camera(s) 1274, and/or on in-cabin monitoring camera sensor(s). In at least one embodiment, in-cabin monitoring camera sensor(s) are preferably monitored by a neural network running on another instance of SoC 1204, configured to identify in cabin events and respond accordingly. In at least one embodiment, an in-cabin system may perform, without limitation, lip reading to activate cellular service and place a phone call, dictate emails, change a vehicle's destination, activate or change a vehicle's infotainment system and settings, or provide voice-activated web surfing. In at least one embodiment, certain functions are available to a driver when a vehicle is operating in an autonomous mode and are disabled otherwise.

[0356] In at least one embodiment, a video image compositor may include enhanced temporal noise reduction for both spatial and temporal noise reduction. For example, in at least one embodiment, where motion occurs in a video, noise reduction weights spatial information appropriately, decreasing weights of information provided by adjacent frames.

In at least one embodiment, where an image or portion of an image does not include motion, temporal noise reduction performed by video image compositor may use information from a previous image to reduce noise in a current image.

[0357] In at least one embodiment, a video image compositor may also be configured to perform stereo rectification on input stereo lens frames. In at least one embodiment, a video image compositor may further be used for user interface composition when an operating system desktop is in use, and GPU(s) 1208 are not required to continuously render new surfaces. In at least one embodiment, when GPU(s) 1208 are powered on and active doing 3D rendering, a video image compositor may be used to offload GPU(s) 1208 to improve performance and responsiveness.

[0358] In at least one embodiment, one or more SoC of SoC(s) 1204 may further include a mobile industry processor interface ("MIPI") camera serial interface for receiving video and input from cameras, a high-speed interface, and/or a video input block that may be used for a camera and related pixel input functions. In at least one embodiment, one or more of SoC(s) 1204 may further include an input/output controller(s) that may be controlled by software and may be used for receiving I/O signals that are uncommitted to a specific role.

[0359] In at least one embodiment, one or more Soc of SoC(s) 1204 may further include a broad range of peripheral interfaces to enable communication with peripherals, audio

embodiment, SoC(s) 1204 may be used to process data from cameras (e.g., connected over Gigabit Multimedia Serial Link and Ethernet channels), sensors (e.g., LIDAR sensor(s) 1264, RADAR sensor(s) 1260, etc. that may be connected over Ethernet channels), data from bus 1202 (e.g., speed of vehicle 1200, steering wheel position, etc.), data from GNSS sensor(s) 1258 (e.g., connected over a Ethernet bus or a CAN bus), etc. In at least one embodiment, one or more SoC of SoC(s) 1204 may further include dedicated high-performance mass storage controllers that may include their own DMA engines, and that may be used to free CPU(s) 1206 from routine data management tasks.

[0360] In at least one embodiment, SoC(s) 1204 may be an end-to-end platform with a flexible architecture that spans automation Levels 3-5, thereby providing a comprehensive functional safety architecture that leverages and makes efficient use of computer vision and ADAS techniques for diversity and redundancy, and provides a platform for a flexible, reliable driving software stack, along with deep learning tools. In at least one embodiment, SoC(s) 1204 may be faster, more reliable, and even more energy-efficient and space-efficient than conventional systems. For example, in at least one embodiment, accelerator(s) 1214, when combined with CPU(s) 1206, GPU(s) 1208, and data store(s) 1216, may provide for a fast, efficient platform for Level 3-5 autonomous vehicles.

[0361] In at least one embodiment, computer vision algorithms may be executed on CPUs, which may be configured using a high-level programming language, such as C, to execute a wide variety of processing algorithms across a wide variety of visual data. However, in at least one embodiment, CPUs are oftentimes unable to meet performance requirements of many computer vision applications, such as those related to execution time and power consumption, for example. In at least one embodiment, many CPUs are unable to execute complex object detection algorithms in real-time, which is used in in-vehicle ADAS applications and in practical Level 3-5 autonomous vehicles.

[0362] Embodiments described herein allow for multiple neural networks to be performed simultaneously and/or sequentially, and for results to be combined together to enable Level 3-5 autonomous driving functionality. For example, in at least one embodiment, a CNN executing on a DLA or a discrete GPU (e.g., GPU(s) 1220) may include text and word recognition, allowing reading and understanding of traffic signs, including signs for which a neural network has not been specifically trained. In at least one embodiment, a DLA may

understanding of a sign, and to pass that semantic understanding to path planning modules running on a CPU Complex.

[0363] In at least one embodiment, multiple neural networks may be run simultaneously, as for Level 3, 4, or 5 driving. For example, in at least one embodiment, a warning sign stating “Caution: flashing lights indicate icy conditions,” along with an electric light, may be independently or collectively interpreted by several neural networks. In at least one embodiment, such warning sign itself may be identified as a traffic sign by a first deployed neural network (e.g., a neural network that has been trained), text “flashing lights indicate icy conditions” may be interpreted by a second deployed neural network, which informs a vehicle’s path planning software (preferably executing on a CPU Complex) that when flashing lights are detected, icy conditions exist. In at least one embodiment, a flashing light may be identified by operating a third deployed neural network over multiple frames, informing a vehicle’s path-planning software of a presence (or an absence) of flashing lights. In at least one embodiment, all three neural networks may run simultaneously, such as within a DLA and/or on GPU(s) 1208.

[0364] In at least one embodiment, a CNN for facial recognition and vehicle owner identification may use data from camera sensors to identify presence of an authorized driver and/or owner of vehicle 1200. In at least one embodiment, an always-on sensor processing engine may be used to unlock a vehicle when an owner approaches a driver door and turns on lights, and, in a security mode, to disable such vehicle when an owner leaves such vehicle. In this way, SoC(s) 1204 provide for security against theft and/or carjacking.

[0365] In at least one embodiment, a CNN for emergency vehicle detection and identification may use data from microphones 1296 to detect and identify emergency vehicle sirens. In at least one embodiment, SoC(s) 1204 use a CNN for classifying environmental and urban sounds, as well as classifying visual data. In at least one embodiment, a CNN running on a DLA is trained to identify a relative closing speed of an emergency vehicle (e.g., by using a Doppler effect). In at least one embodiment, a CNN may also be trained to identify emergency vehicles specific to a local area in which a vehicle is operating, as identified by GNSS sensor(s) 1258. In at least one embodiment, when operating in Europe, a CNN will seek to detect European sirens, and when in North America, a CNN will seek to identify only North American sirens. In at least one embodiment, once an emergency vehicle

slowing a vehicle, pulling over to a side of a road, parking a vehicle, and/or idling a vehicle, with assistance of ultrasonic sensor(s) 1262, until emergency vehicles pass.

[0366] In at least one embodiment, vehicle 1200 may include CPU(s) 1218 (e.g., discrete CPU(s), or dCPU(s)), that may be coupled to SoC(s) 1204 via a high-speed interconnect (e.g., PCIe). In at least one embodiment, CPU(s) 1218 may include an X86 processor, for example. CPU(s) 1218 may be used to perform any of a variety of functions, including arbitrating potentially inconsistent results between ADAS sensors and SoC(s) 1204, and/or monitoring status and health of controller(s) 1236 and/or an infotainment system on a chip (“infotainment SoC”) 1230, for example.

[0367] In at least one embodiment, vehicle 1200 may include GPU(s) 1220 (e.g., discrete GPU(s), or dGPU(s)), that may be coupled to SoC(s) 1204 via a high-speed interconnect (e.g., NVIDIA’s NVLINK channel). In at least one embodiment, GPU(s) 1220 may provide additional artificial intelligence functionality, such as by executing redundant and/or different neural networks, and may be used to train and/or update neural networks based at least in part on input (e.g., sensor data) from sensors of a vehicle 1200.

[0368] In at least one embodiment, vehicle 1200 may further include network interface 1224 which may include, without limitation, wireless antenna(s) 1226 (e.g., one or more wireless antennas for different communication protocols, such as a cellular antenna, a Bluetooth antenna, etc.). In at least one embodiment, network interface 1224 may be used to enable wireless connectivity to Internet cloud services (e.g., with server(s) and/or other network devices), with other vehicles, and/or with computing devices (e.g., client devices of passengers). In at least one embodiment, to communicate with other vehicles, a direct link may be established between vehicle 120 and another vehicle and/or an indirect link may be established (e.g., across networks and over the Internet). In at least one embodiment, direct links may be provided using a vehicle-to-vehicle communication link. In at least one embodiment, a vehicle-to-vehicle communication link may provide vehicle 1200 information about vehicles in proximity to vehicle 1200 (e.g., vehicles in front of, on a side of, and/or behind vehicle 1200). In at least one embodiment, such aforementioned functionality may be part of a cooperative adaptive cruise control functionality of vehicle 1200.

[0369] In at least one embodiment, network interface 1224 may include an SoC that provides modulation and demodulation functionality and enables controller(s) 1236 to

may include a radio frequency front-end for up-conversion from baseband to radio frequency, and down conversion from radio frequency to baseband. In at least one embodiment, frequency conversions may be performed in any technically feasible fashion. For example, frequency conversions could be performed through well-known processes, and/or using super-heterodyne processes. In at least one embodiment, radio frequency front end functionality may be provided by a separate chip. In at least one embodiment, network interfaces may include wireless functionality for communicating over LTE, WCDMA, UMTS, GSM, CDMA2000, Bluetooth, Bluetooth LE, Wi-Fi, Z-Wave, ZigBee, LoRaWAN, and/or other wireless protocols.

[0370] In at least one embodiment, vehicle 1200 may further include data store(s) 1228 which may include, without limitation, off-chip (e.g., off SoC(s) 1204) storage. In at least one embodiment, data store(s) 1228 may include, without limitation, one or more storage elements including RAM, SRAM, dynamic random-access memory (“DRAM”), video random-access memory (“VRAM”), flash memory, hard disks, and/or other components and/or devices that may store at least one bit of data.

[0371] In at least one embodiment, vehicle 1200 may further include GNSS sensor(s) 1258 (e.g., GPS and/or assisted GPS sensors), to assist in mapping, perception, occupancy grid generation, and/or path planning functions. In at least one embodiment, any number of GNSS sensor(s) 1258 may be used, including, for example and without limitation, a GPS using a USB connector with an Ethernet-to-Serial (e.g., RS-232) bridge.

[0372] In at least one embodiment, vehicle 1200 may further include RADAR sensor(s) 1260. In at least one embodiment, RADAR sensor(s) 1260 may be used by vehicle 1200 for long-range vehicle detection, even in darkness and/or severe weather conditions. In at least one embodiment, RADAR functional safety levels may be ASIL B. In at least one embodiment, RADAR sensor(s) 1260 may use a CAN bus and/or bus 1202 (e.g., to transmit data generated by RADAR sensor(s) 1260) for control and to access object tracking data, with access to Ethernet channels to access raw data in some examples. In at least one embodiment, a wide variety of RADAR sensor types may be used. For example, and without limitation, RADAR sensor(s) 1260 may be suitable for front, rear, and side RADAR use. In at least one embodiment, one or more sensor of RADAR sensors(s) 1260 is a Pulse Doppler RADAR sensor.

configurations, such as long-range with narrow field of view, short-range with wide field of view, short-range side coverage, etc. In at least one embodiment, long-range RADAR may be used for adaptive cruise control functionality. In at least one embodiment, long-range

5 RADAR systems may provide a broad field of view realized by two or more independent scans, such as within a 250 m (meter) range. In at least one embodiment, RADAR sensor(s) 1260 may help in distinguishing between static and moving objects, and may be used by ADAS system 1238 for emergency brake assist and forward collision warning. In at least one embodiment, sensors 1260(s) included in a long-range RADAR system may include, without

10 limitation, monostatic multimodal RADAR with multiple (e.g., six or more) fixed RADAR antennae and a high-speed CAN and FlexRay interface. In at least one embodiment, with six antennae, a central four antennae may create a focused beam pattern, designed to record vehicle's 1200 surroundings at higher speeds with minimal interference from traffic in adjacent lanes. In at least one embodiment, another two antennae may expand field of view,

15 making it possible to quickly detect vehicles entering or leaving a lane of vehicle 1200.

[0374] In at least one embodiment, mid-range RADAR systems may include, as an example, a range of up to 160 m (front) or 80 m (rear), and a field of view of up to 42 degrees (front) or 150 degrees (rear). In at least one embodiment, short-range RADAR systems may include, without limitation, any number of RADAR sensor(s) 1260 designed to be installed at

20 both ends of a rear bumper. When installed at both ends of a rear bumper, in at least one embodiment, a RADAR sensor system may create two beams that constantly monitor blind spots in a rear direction and next to a vehicle. In at least one embodiment, short-range RADAR systems may be used in ADAS system 1238 for blind spot detection and/or lane change assist.

25 [0375] In at least one embodiment, vehicle 1200 may further include ultrasonic sensor(s) 1262. In at least one embodiment, ultrasonic sensor(s) 1262, which may be positioned at a front, a back, and/or side location of vehicle 1200, may be used for parking assist and/or to create and update an occupancy grid. In at least one embodiment, a wide variety of ultrasonic sensor(s) 1262 may be used, and different ultrasonic sensor(s) 1262 may be used for different

30 ranges of detection (e.g., 2.5 m, 4 m). In at least one embodiment, ultrasonic sensor(s) 1262 may operate at functional safety levels of ASIL B.

least one embodiment, LIDAR sensor(s) 1264 may be used for object and pedestrian detection, emergency braking, collision avoidance, and/or other functions. In at least one embodiment, LIDAR sensor(s) 1264 may operate at functional safety level ASIL B. In at least one embodiment, vehicle 1200 may include multiple LIDAR sensors 1264 (e.g., two, four, six, etc.) that may use an Ethernet channel (e.g., to provide data to a Gigabit Ethernet switch).

[0377] In at least one embodiment, LIDAR sensor(s) 1264 may be capable of providing a list of objects and their distances for a 360-degree field of view. In at least one embodiment, commercially available LIDAR sensor(s) 1264 may have an advertised range of approximately 100 m, with an accuracy of 2 cm to 3 cm, and with support for a 100 Mbps Ethernet connection, for example. In at least one embodiment, one or more non-protruding LIDAR sensors may be used. In such an embodiment, LIDAR sensor(s) 1264 may include a small device that may be embedded into a front, a rear, a side, and/or a corner location of vehicle 1200. In at least one embodiment, LIDAR sensor(s) 1264, in such an embodiment, may provide up to a 120-degree horizontal and 35-degree vertical field-of-view, with a 200 m range even for low-reflectivity objects. In at least one embodiment, front-mounted LIDAR sensor(s) 1264 may be configured for a horizontal field of view between 45 degrees and 135 degrees.

[0378] In at least one embodiment, LIDAR technologies, such as 3D flash LIDAR, may also be used. In at least one embodiment, 3D flash LIDAR uses a flash of a laser as a transmission source, to illuminate surroundings of vehicle 1200 up to approximately 200 m. In at least one embodiment, a flash LIDAR unit includes, without limitation, a receptor, which records laser pulse transit time and reflected light on each pixel, which in turn corresponds to a range from vehicle 1200 to objects. In at least one embodiment, flash LIDAR may allow for highly accurate and distortion-free images of surroundings to be generated with every laser flash. In at least one embodiment, four flash LIDAR sensors may be deployed, one at each side of vehicle 1200. In at least one embodiment, 3D flash LIDAR systems include, without limitation, a solid-state 3D staring array LIDAR camera with no moving parts other than a fan (e.g., a non-scanning LIDAR device). In at least one embodiment, flash LIDAR device may use a 5 nanosecond class I (eye-safe) laser pulse per frame and may capture reflected laser light as a 3D range point cloud and co-registered intensity data.

In at least one embodiment, IMU sensor(s) 1266 may be located at a center of a rear axle of vehicle 1200. In at least one embodiment, IMU sensor(s) 1266 may include, for example and without limitation, accelerometer(s), magnetometer(s), gyroscope(s), a magnetic compass, magnetic compasses, and/or other sensor types. In at least one embodiment, such as in six-axis applications, IMU sensor(s) 1266 may include, without limitation, accelerometers and gyroscopes. In at least one embodiment, such as in nine-axis applications, IMU sensor(s) 1266 may include, without limitation, accelerometers, gyroscopes, and magnetometers.

[0380] In at least one embodiment, IMU sensor(s) 1266 may be implemented as a miniature, high performance GPS-Aided Inertial Navigation System (“GPS/INS”) that combines micro-electro-mechanical systems (“MEMS”) inertial sensors, a high-sensitivity GPS receiver, and advanced Kalman filtering algorithms to provide estimates of position, velocity, and attitude. In at least one embodiment, IMU sensor(s) 1266 may enable vehicle 1200 to estimate its heading without requiring input from a magnetic sensor by directly observing and correlating changes in velocity from a GPS to IMU sensor(s) 1266. In at least one embodiment, IMU sensor(s) 1266 and GNSS sensor(s) 1258 may be combined in a single integrated unit.

[0381] In at least one embodiment, vehicle 1200 may include microphone(s) 1296 placed in and/or around vehicle 1200. In at least one embodiment, microphone(s) 1296 may be used for emergency vehicle detection and identification, among other things.

[0382] In at least one embodiment, vehicle 1200 may further include any number of camera types, including stereo camera(s) 1268, wide-view camera(s) 1270, infrared camera(s) 1272, surround camera(s) 1274, long-range camera(s) 1298, mid-range camera(s) 1276, and/or other camera types. In at least one embodiment, cameras may be used to capture image data around an entire periphery of vehicle 1200. In at least one embodiment, which types of cameras used depends on vehicle 1200. In at least one embodiment, any combination of camera types may be used to provide necessary coverage around vehicle 1200. In at least one embodiment, a number of cameras deployed may differ depending on embodiment. For example, in at least one embodiment, vehicle 1200 could include six cameras, seven cameras, ten cameras, twelve cameras, or another number of cameras. In at least one embodiment, cameras may support, as an example and without limitation, Gigabit Multimedia Serial Link (“GMSL”) and/or Gigabit Ethernet communications. In at least one embodiment, each

and FIG. 12B.

[0383] In at least one embodiment, vehicle 1200 may further include vibration sensor(s) 1242. In at least one embodiment, vibration sensor(s) 1242 may measure vibrations of components of vehicle 1200, such as axle(s). For example, in at least one embodiment, changes in vibrations may indicate a change in road surfaces. In at least one embodiment, when two or more vibration sensors 1242 are used, differences between vibrations may be used to determine friction or slippage of road surface (e.g., when a difference in vibration is between a power-driven axle and a freely rotating axle).

[0384] In at least one embodiment, vehicle 1200 may include ADAS system 1238. In at least one embodiment, ADAS system 1238 may include, without limitation, an SoC, in some examples. In at least one embodiment, ADAS system 1238 may include, without limitation, any number and combination of an autonomous/adaptive/automatic cruise control (“ACC”) system, a cooperative adaptive cruise control (“CACC”) system, a forward crash warning (“FCW”) system, an automatic emergency braking (“AEB”) system, a lane departure warning (“LDW”) system, a lane keep assist (“LKA”) system, a blind spot warning (“BSW”) system, a rear cross-traffic warning (“RCTW”) system, a collision warning (“CW”) system, a lane centering (“LC”) system, and/or other systems, features, and/or functionality.

[0385] In at least one embodiment, ACC system may use RADAR sensor(s) 1260, LIDAR sensor(s) 1264, and/or any number of camera(s). In at least one embodiment, ACC system may include a longitudinal ACC system and/or a lateral ACC system. In at least one embodiment, a longitudinal ACC system monitors and controls distance to another vehicle immediately ahead of vehicle 1200 and automatically adjusts speed of vehicle 1200 to maintain a safe distance from vehicles ahead. In at least one embodiment, a lateral ACC system performs distance keeping, and advises vehicle 1200 to change lanes when necessary. In at least one embodiment, a lateral ACC is related to other ADAS applications, such as LC and CW.

[0386] In at least one embodiment, a CACC system uses information from other vehicles that may be received via network interface 1224 and/or wireless antenna(s) 1226 from other vehicles via a wireless link, or indirectly, over a network connection (e.g., over the Internet). In at least one embodiment, direct links may be provided by a vehicle-to-vehicle (“V2V”) communication link, while indirect links may be provided by an infrastructure-to-vehicle

immediately preceding vehicles (e.g., vehicles immediately ahead of and in same lane as vehicle 1200), while I2V communication provides information about traffic further ahead. In at least one embodiment, a CACC system may include either or both I2V and V2V

5 information sources. In at least one embodiment, given information of vehicles ahead of vehicle 1200, a CACC system may be more reliable and it has potential to improve traffic flow smoothness and reduce congestion on road.

[0387] In at least one embodiment, an FCW system is designed to alert a driver to a hazard, so that such driver may take corrective action. In at least one embodiment, an FCW system
10 uses a front-facing camera and/or RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, an FCW system may provide a warning, such as in form of a sound, visual warning, vibration and/or a quick brake pulse.

15 [0388] In at least one embodiment, an AEB system detects an impending forward collision with another vehicle or other object, and may automatically apply brakes if a driver does not take corrective action within a specified time or distance parameter. In at least one embodiment, AEB system may use front-facing camera(s) and/or RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC. In at least one embodiment,
20 when an AEB system detects a hazard, it will typically first alert a driver to take corrective action to avoid collision and, if that driver does not take corrective action, that AEB system may automatically apply brakes in an effort to prevent, or at least mitigate, an impact of a predicted collision. In at least one embodiment, an AEB system may include techniques such as dynamic brake support and/or crash imminent braking.

25 [0389] In at least one embodiment, an LDW system provides visual, audible, and/or tactile warnings, such as steering wheel or seat vibrations, to alert driver when vehicle 1200 crosses lane markings. In at least one embodiment, an LDW system does not activate when a driver indicates an intentional lane departure, such as by activating a turn signal. In at least one embodiment, an LDW system may use front-side facing cameras, coupled to a dedicated
30 processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, an LKA system is a variation of an LDW system. In at least one embodiment, an LKA system

lane.

[0390] In at least one embodiment, a BSW system detects and warns a driver of vehicles in an automobile's blind spot. In at least one embodiment, a BSW system may provide a visual, audible, and/or tactile alert to indicate that merging or changing lanes is unsafe. In at least one embodiment, a BSW system may provide an additional warning when a driver uses a turn signal. In at least one embodiment, a BSW system may use rear-side facing camera(s) and/or RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component.

[0391] In at least one embodiment, an RCTW system may provide visual, audible, and/or tactile notification when an object is detected outside a rear-camera range when vehicle 1200 is backing up. In at least one embodiment, an RCTW system includes an AEB system to ensure that vehicle brakes are applied to avoid a crash. In at least one embodiment, an RCTW system may use one or more rear-facing RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component.

[0392] In at least one embodiment, conventional ADAS systems may be prone to false positive results which may be annoying and distracting to a driver, but typically are not catastrophic, because conventional ADAS systems alert a driver and allow that driver to decide whether a safety condition truly exists and act accordingly. In at least one embodiment, vehicle 1200 itself decides, in case of conflicting results, whether to heed result from a primary computer or a secondary computer (e.g., a first controller or a second controller of controllers 1236). For example, in at least one embodiment, ADAS system 1238 may be a backup and/or secondary computer for providing perception information to a backup computer rationality module. In at least one embodiment, a backup computer rationality monitor may run redundant diverse software on hardware components to detect faults in perception and dynamic driving tasks. In at least one embodiment, outputs from ADAS system 1238 may be provided to a supervisory MCU. In at least one embodiment, if outputs from a primary computer and outputs from a secondary computer conflict, a supervisory MCU determines how to reconcile conflict to ensure safe operation.

supervisory MCU with a confidence score, indicating that primary computer's confidence in a chosen result. In at least one embodiment, if that confidence score exceeds a threshold, that supervisory MCU may follow that primary computer's direction, regardless of whether that
5 secondary computer provides a conflicting or inconsistent result. In at least one embodiment, where a confidence score does not meet a threshold, and where primary and secondary computers indicate different results (e.g., a conflict), a supervisory MCU may arbitrate between computers to determine an appropriate outcome.

[0394] In at least one embodiment, a supervisory MCU may be configured to run a neural
10 network(s) that is trained and configured to determine, based at least in part on outputs from a primary computer and outputs from a secondary computer, conditions under which that secondary computer provides false alarms. In at least one embodiment, neural network(s) in a supervisory MCU may learn when a secondary computer's output may be trusted, and when it cannot. For example, in at least one embodiment, when that secondary computer is a
15 RADAR-based FCW system, a neural network(s) in that supervisory MCU may learn when an FCW system is identifying metallic objects that are not, in fact, hazards, such as a drainage grate or manhole cover that triggers an alarm. In at least one embodiment, when a secondary computer is a camera-based LDW system, a neural network in a supervisory MCU may learn to override LDW when bicyclists or pedestrians are present and a lane departure is,
20 in fact, a safest maneuver. In at least one embodiment, a supervisory MCU may include at least one of a DLA or a GPU suitable for running neural network(s) with associated memory. In at least one embodiment, a supervisory MCU may comprise and/or be included as a component of SoC(s) 1204.

[0395] In at least one embodiment, ADAS system 1238 may include a secondary computer
25 that performs ADAS functionality using traditional rules of computer vision. In at least one embodiment, that secondary computer may use classic computer vision rules (if-then), and presence of a neural network(s) in a supervisory MCU may improve reliability, safety and performance. For example, in at least one embodiment, diverse implementation and intentional non-identity makes an overall system more fault-tolerant, especially to faults
30 caused by software (or software-hardware interface) functionality. For example, in at least one embodiment, if there is a software bug or error in software running on a primary computer, and non-identical software code running on a secondary computer provides a consistent overall result, then a supervisory MCU may have greater confidence that an overall

material error.

[0396] In at least one embodiment, an output of ADAS system 1238 may be fed into a primary computer's perception block and/or a primary computer's dynamic driving task block. For example, in at least one embodiment, if ADAS system 1238 indicates a forward crash warning due to an object immediately ahead, a perception block may use this information when identifying objects. In at least one embodiment, a secondary computer may have its own neural network that is trained and thus reduces a risk of false positives, as described herein.

[0397] In at least one embodiment, vehicle 1200 may further include infotainment SoC 1230 (e.g., an in-vehicle infotainment system (IVI)). Although illustrated and described as an SoC, infotainment system SoC 1230, in at least one embodiment, may not be an SoC, and may include, without limitation, two or more discrete components. In at least one embodiment, infotainment SoC 1230 may include, without limitation, a combination of hardware and software that may be used to provide audio (e.g., music, a personal digital assistant, navigational instructions, news, radio, etc.), video (e.g., TV, movies, streaming, etc.), phone (e.g., hands-free calling), network connectivity (e.g., LTE, WiFi, etc.), and/or information services (e.g., navigation systems, rear-parking assistance, a radio data system, vehicle related information such as fuel level, total distance covered, brake fuel level, oil level, door open/close, air filter information, etc.) to vehicle 1200. For example, infotainment SoC 1230 could include radios, disk players, navigation systems, video players, USB and Bluetooth connectivity, carputers, in-car entertainment, WiFi, steering wheel audio controls, hands free voice control, a heads-up display ("HUD"), HMI display 1234, a telematics device, a control panel (e.g., for controlling and/or interacting with various components, features, and/or systems), and/or other components. In at least one embodiment, infotainment SoC 1230 may further be used to provide information (e.g., visual and/or audible) to user(s) of vehicle 1200, such as information from ADAS system 1238, autonomous driving information such as planned vehicle maneuvers, trajectories, surrounding environment information (e.g., intersection information, vehicle information, road information, etc.), and/or other information.

[0398] In at least one embodiment, infotainment SoC 1230 may include any amount and type of GPU functionality. In at least one embodiment, infotainment SoC 1230 may

In at least one embodiment, infotainment SoC 1230 may be coupled to a supervisory MCU such that a GPU of an infotainment system may perform some self-driving functions in event that primary controller(s) 1236 (e.g., primary and/or backup computers of vehicle 1200) fail.

5 In at least one embodiment, infotainment SoC 1230 may put vehicle 1200 into a chauffeur to safe stop mode, as described herein.

[0399] In at least one embodiment, vehicle 1200 may further include instrument cluster 1232 (e.g., a digital dash, an electronic instrument cluster, a digital instrument panel, etc.). In at least one embodiment, instrument cluster 1232 may include, without limitation, a
10 controller and/or supercomputer (e.g., a discrete controller or supercomputer). In at least one embodiment, instrument cluster 1232 may include, without limitation, any number and combination of a set of instrumentation such as a speedometer, fuel level, oil pressure, tachometer, odometer, turn indicators, gearshift position indicator, seat belt warning light(s), parking-brake warning light(s), engine-malfunction light(s), supplemental restraint system
15 (e.g., airbag) information, lighting controls, safety system controls, navigation information, etc. In some examples, information may be displayed and/or shared among infotainment SoC 1230 and instrument cluster 1232. In at least one embodiment, instrument cluster 1232 may be included as part of infotainment SoC 1230, or vice versa.

[0400] Inference and/or training logic 915 are used to perform inferencing and/or training
20 operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 12C for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or
25 neural network use cases described herein.

[0401] FIG. 12D is a diagram of a system for communication between cloud-based server(s) and autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, system may include, without limitation, server(s) 1278, network(s) 1290, and any number and type of vehicles, including vehicle 1200. In at least
30 one embodiment, server(s) 1278 may include, without limitation, a plurality of GPUs 1284(A)-1284(H) (collectively referred to herein as GPUs 1284), PCIe switches 1282(A)-1282(D) (collectively referred to herein as PCIe switches 1282), and/or CPUs 1280(A)-

1284, CPUs 1280, and PCIe switches 1282 may be interconnected with high-speed interconnects such as, for example and without limitation, NVLink interfaces 1288 developed by NVIDIA and/or PCIe connections 1286. In at least one embodiment, GPUs 1284 are
5 connected via an NVLink and/or NVSwitch SoC and GPUs 1284 and PCIe switches 1282 are connected via PCIe interconnects. Although eight GPUs 1284, two CPUs 1280, and four PCIe switches 1282 are illustrated, this is not intended to be limiting. In at least one embodiment, each of server(s) 1278 may include, without limitation, any number of GPUs 1284, CPUs 1280, and/or PCIe switches 1282, in any combination. For example, in at least
10 one embodiment, server(s) 1278 could each include eight, sixteen, thirty-two, and/or more GPUs 1284.

[0402] In at least one embodiment, server(s) 1278 may receive, over network(s) 1290 and from vehicles, image data representative of images showing unexpected or changed road conditions, such as recently commenced road-work. In at least one embodiment, server(s)
15 1278 may transmit, over network(s) 1290 and to vehicles, neural networks 1292, updated or otherwise, and/or map information 1294, including, without limitation, information regarding traffic and road conditions. In at least one embodiment, updates to map information 1294 may include, without limitation, updates for HD map 1222, such as information regarding construction sites, potholes, detours, flooding, and/or other obstructions. In at least one
20 embodiment, neural networks 1292, and/or map information 1294 may have resulted from new training and/or experiences represented in data received from any number of vehicles in an environment, and/or based at least in part on training performed at a data center (e.g., using server(s) 1278 and/or other servers).

[0403] In at least one embodiment, server(s) 1278 may be used to train machine learning
25 models (e.g., neural networks) based at least in part on training data. In at least one embodiment, training data may be generated by vehicles, and/or may be generated in a simulation (e.g., using a game engine). In at least one embodiment, any amount of training data is tagged (e.g., where associated neural network benefits from supervised learning) and/or undergoes other pre-processing. In at least one embodiment, any amount of training
30 data is not tagged and/or pre-processed (e.g., where associated neural network does not require supervised learning). In at least one embodiment, once machine learning models are trained, machine learning models may be used by vehicles (e.g., transmitted to vehicles over

monitor vehicles.

[0404] In at least one embodiment, server(s) 1278 may receive data from vehicles and apply data to up-to-date real-time neural networks for real-time intelligent inferencing. In at least one embodiment, server(s) 1278 may include deep-learning supercomputers and/or dedicated AI computers powered by GPU(s) 1284, such as a DGX and DGX Station machines developed by NVIDIA. However, in at least one embodiment, server(s) 1278 may include deep learning infrastructure that uses CPU-powered data centers.

[0405] In at least one embodiment, deep-learning infrastructure of server(s) 1278 may be capable of fast, real-time inferencing, and may use that capability to evaluate and verify health of processors, software, and/or associated hardware in vehicle 1200. For example, in at least one embodiment, deep-learning infrastructure may receive periodic updates from vehicle 1200, such as a sequence of images and/or objects that vehicle 1200 has located in that sequence of images (e.g., via computer vision and/or other machine learning object classification techniques). In at least one embodiment, deep-learning infrastructure may run its own neural network to identify objects and compare them with objects identified by vehicle 1200 and, if results do not match and deep-learning infrastructure concludes that AI in vehicle 1200 is malfunctioning, then server(s) 1278 may transmit a signal to vehicle 1200 instructing a fail-safe computer of vehicle 1200 to assume control, notify passengers, and complete a safe parking maneuver.

[0406] In at least one embodiment, server(s) 1278 may include GPU(s) 1284 and one or more programmable inference accelerators (e.g., NVIDIA's TensorRT 3 devices). In at least one embodiment, a combination of GPU-powered servers and inference acceleration may make real-time responsiveness possible. In at least one embodiment, such as where performance is less critical, servers powered by CPUs, FPGAs, and other processors may be used for inferencing. In at least one embodiment, hardware structure(s) 915 are used to perform one or more embodiments. Details regarding hardware structure(x) 915 are provided herein in conjunction with FIGS. 9A and/or 9B.

COMPUTER SYSTEMS

[0407] FIG. 13 is a block diagram illustrating an exemplary computer system, which may be a system with interconnected devices and components, a system-on-a-chip (SOC) or some combination thereof formed with a processor that may include execution units to execute an

system 1300 may include, without limitation, a component, such as a processor 1302 to employ execution units including logic to perform algorithms for process data, in accordance with present disclosure, such as in embodiment described herein. In at least one embodiment, computer system 1300 may include processors, such as PENTIUM® Processor family, Xeon™, Itanium®, XScale™ and/or StrongARM™, Intel® Core™, or Intel® Nervana™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and like) may also be used. In at least one embodiment, computer system 1300 may execute a version of WINDOWS operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems (UNIX and Linux, for example), embedded software, and/or graphical user interfaces, may also be used.

[0408] Embodiments may be used in other devices such as handheld devices and embedded applications. Some examples of handheld devices include cellular phones, Internet Protocol devices, digital cameras, personal digital assistants (“PDAs”), and handheld PCs. In at least one embodiment, embedded applications may include a microcontroller, a digital signal processor (“DSP”), system on a chip, network computers (“NetPCs”), set-top boxes, network hubs, wide area network (“WAN”) switches, or any other system that may perform one or more instructions in accordance with at least one embodiment.

[0409] In at least one embodiment, computer system 1300 may include, without limitation, processor 1302 that may include, without limitation, one or more execution units 1308 to perform machine learning model training and/or inferencing according to techniques described herein. In at least one embodiment, computer system 1300 is a single processor desktop or server system, but in another embodiment, computer system 1300 may be a multiprocessor system. In at least one embodiment, processor 1302 may include, without limitation, a complex instruction set computer (“CISC”) microprocessor, a reduced instruction set computing (“RISC”) microprocessor, a very long instruction word (“VLIW”) microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such as a digital signal processor, for example. In at least one embodiment, processor 1302 may be coupled to a processor bus 1310 that may transmit data signals between processor 1302 and other components in computer system 1300.

1 (“L1”) internal cache memory (“cache”) 1304. In at least one embodiment, processor 1302 may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor 1302. Other embodiments may also include a combination of both internal and external caches depending on particular implementation and needs. In at least one embodiment, a register file 1306 may store different types of data in various registers including, without limitation, integer registers, floating point registers, status registers, and an instruction pointer register.

[0411] In at least one embodiment, execution unit 1308, including, without limitation, logic to perform integer and floating point operations, also resides in processor 1302. In at least one embodiment, processor 1302 may also include a microcode (“ucode”) read only memory (“ROM”) that stores microcode for certain macro instructions. In at least one embodiment, execution unit 1308 may include logic to handle a packed instruction set 1309. In at least one embodiment, by including packed instruction set 1309 in an instruction set of a general-purpose processor, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using packed data in processor 1302. In at least one embodiment, many multimedia applications may be accelerated and executed more efficiently by using a full width of a processor’s data bus for performing operations on packed data, which may eliminate a need to transfer smaller units of data across that processor’s data bus to perform one or more operations one data element at a time.

[0412] In at least one embodiment, execution unit 1308 may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. In at least one embodiment, computer system 1300 may include, without limitation, a memory 1320. In at least one embodiment, memory 1320 may be a Dynamic Random Access Memory (“DRAM”) device, a Static Random Access Memory (“SRAM”) device, a flash memory device, or another memory device. In at least one embodiment, memory 1320 may store instruction(s) 1319 and/or data 1321 represented by data signals that may be executed by processor 1302.

[0413] In at least one embodiment, a system logic chip may be coupled to processor bus 1310 and memory 1320. In at least one embodiment, a system logic chip may include, without limitation, a memory controller hub (“MCH”) 1316, and processor 1302 may communicate with MCH 1316 via processor bus 1310. In at least one embodiment, MCH

data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH 1316 may direct data signals between processor 1302, memory 1320, and other components in computer system 1300 and to bridge data signals between processor bus 1310, memory 1320, and a system I/O interface 1322. In at least one embodiment, a system logic chip may provide a graphics port for coupling to a graphics controller. In at least one embodiment, MCH 1316 may be coupled to memory 1320 through high bandwidth memory path 1318 and a graphics/video card 1312 may be coupled to MCH 1316 through an Accelerated Graphics Port (“AGP”) interconnect 1314.

[0414] In at least one embodiment, computer system 1300 may use system I/O interface 1322 as a proprietary hub interface bus to couple MCH 1316 to an I/O controller hub (“ICH”) 1330. In at least one embodiment, ICH 1330 may provide direct connections to some I/O devices via a local I/O bus. In at least one embodiment, a local I/O bus may include, without limitation, a high-speed I/O bus for connecting peripherals to memory 1320, a chipset, and processor 1302. Examples may include, without limitation, an audio controller 1329, a firmware hub (“flash BIOS”) 1328, a wireless transceiver 1326, a data storage 1324, a legacy I/O controller 1323 containing user input and keyboard interfaces 1325, a serial expansion port 1327, such as a Universal Serial Bus (“USB”) port, and a network controller 1334. In at least one embodiment, data storage 1324 may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

[0415] In at least one embodiment, FIG. 13 illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. 13 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 13 may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of computer system 1300 are interconnected using compute express link (CXL) interconnects.

[0416] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 13 for inferencing or predicting operations based, at least in part, on weight parameters calculated

neural network use cases described herein.

[0417] FIG. 14 is a block diagram illustrating an electronic device 1400 for utilizing a processor 1410, according to at least one embodiment. In at least one embodiment, electronic device 1400 may be, for example and without limitation, a notebook, a tower server, a rack server, a blade server, a laptop, a desktop, a tablet, a mobile device, a phone, an embedded computer, or any other suitable electronic device.

[0418] In at least one embodiment, electronic device 1400 may include, without limitation, processor 1410 communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor 1410 is coupled using a bus or interface, such as a I²C bus, a System Management Bus (“SMBus”), a Low Pin Count (LPC) bus, a Serial Peripheral Interface (“SPI”), a High Definition Audio (“HDA”) bus, a Serial Advance Technology Attachment (“SATA”) bus, a Universal Serial Bus (“USB”) (versions 1, 2, 3, etc.), or a Universal Asynchronous Receiver/Transmitter (“UART”) bus. In at least one embodiment, FIG. 14 illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. 14 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 14 may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of FIG. 14 are interconnected using compute express link (CXL) interconnects.

[0419] In at least one embodiment, FIG 14 may include a display 1424, a touch screen 1425, a touch pad 1430, a Near Field Communications unit (“NFC”) 1445, a sensor hub 1440, a thermal sensor 1446, an Express Chipset (“EC”) 1435, a Trusted Platform Module (“TPM”) 1438, BIOS/firmware/flash memory (“BIOS, FW Flash”) 1422, a DSP 1460, a drive 1420 such as a Solid State Disk (“SSD”) or a Hard Disk Drive (“HDD”), a wireless local area network unit (“WLAN”) 1450, a Bluetooth unit 1452, a Wireless Wide Area Network unit (“WWAN”) 1456, a Global Positioning System (GPS) unit 1455, a camera (“USB 3.0 camera”) 1454 such as a USB 3.0 camera, and/or a Low Power Double Data Rate (“LPDDR”) memory unit (“LPDDR3”) 1415 implemented in, for example, an LPDDR3 standard. These components may each be implemented in any suitable manner.

[0420] In at least one embodiment, other components may be communicatively coupled to processor 1410 through components described herein. In at least one embodiment, an

1444 may be communicatively coupled to sensor hub 1440. In at least one embodiment, a thermal sensor 1439, a fan 1437, a keyboard 1436, and touch pad 1430 may be communicatively coupled to EC 1435. In at least one embodiment, speakers 1463, headphones 1464, and a microphone (“mic”) 1465 may be communicatively coupled to an audio unit (“audio codec and class D amp”) 1462, which may in turn be communicatively coupled to DSP 1460. In at least one embodiment, audio unit 1462 may include, for example and without limitation, an audio coder/decoder (“codec”) and a class D amplifier. In at least one embodiment, a SIM card (“SIM”) 1457 may be communicatively coupled to WWAN unit 1456. In at least one embodiment, components such as WLAN unit 1450 and Bluetooth unit 1452, as well as WWAN unit 1456 may be implemented in a Next Generation Form Factor (“NGFF”).

[0421] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 14 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0422] FIG. 15 illustrates a computer system 1500, according to at least one embodiment. In at least one embodiment, computer system 1500 is configured to implement various processes and methods described throughout this disclosure.

[0423] In at least one embodiment, computer system 1500 comprises, without limitation, at least one central processing unit (“CPU”) 1502 that is connected to a communication bus 1510 implemented using any suitable protocol, such as PCI (“Peripheral Component Interconnect”), peripheral component interconnect express (“PCI-Express”), AGP (“Accelerated Graphics Port”), HyperTransport, or any other bus or point-to-point communication protocol(s). In at least one embodiment, computer system 1500 includes, without limitation, a main memory 1504 and control logic (e.g., implemented as hardware, software, or a combination thereof) and data are stored in main memory 1504, which may take form of random access memory (“RAM”). In at least one embodiment, a network interface subsystem (“network interface”) 1522 provides an interface to other computing

computer system 1500.

[0424] In at least one embodiment, computer system 1500, in at least one embodiment, includes, without limitation, input devices 1508, a parallel processing system 1512, and display devices 1506 that can be implemented using a conventional cathode ray tube (“CRT”), a liquid crystal display (“LCD”), a light emitting diode (“LED”) display, a plasma display, or other suitable display technologies. In at least one embodiment, user input is received from input devices 1508 such as keyboard, mouse, touchpad, microphone, etc. In at least one embodiment, each module described herein can be situated on a single semiconductor platform to form a processing system.

[0425] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 15 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0426] FIG. 16 illustrates a computer system 1600, according to at least one embodiment. In at least one embodiment, computer system 1600 includes, without limitation, a computer 1610 and a USB stick 1620. In at least one embodiment, computer 1610 may include, without limitation, any number and type of processor(s) (not shown) and a memory (not shown). In at least one embodiment, computer 1610 includes, without limitation, a server, a cloud instance, a laptop, and a desktop computer.

[0427] In at least one embodiment, USB stick 1620 includes, without limitation, a processing unit 1630, a USB interface 1640, and USB interface logic 1650. In at least one embodiment, processing unit 1630 may be any instruction execution system, apparatus, or device capable of executing instructions. In at least one embodiment, processing unit 1630 may include, without limitation, any number and type of processing cores (not shown). In at least one embodiment, processing unit 1630 comprises an application specific integrated circuit (“ASIC”) that is optimized to perform any amount and type of operations associated with machine learning. For instance, in at least one embodiment, processing unit 1630 is a tensor processing unit (“TPC”) that is optimized to perform machine learning inference

(“VPU”) that is optimized to perform machine vision and machine learning inference operations.

[0428] In at least one embodiment, USB interface 1640 may be any type of USB connector or USB socket. For instance, in at least one embodiment, USB interface 1640 is a USB 3.0 Type-C socket for data and power. In at least one embodiment, USB interface 1640 is a USB 3.0 Type-A connector. In at least one embodiment, USB interface logic 1650 may include any amount and type of logic that enables processing unit 1630 to interface with devices (e.g., computer 1610) via USB connector 1640.

[0429] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 16 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0430] FIG. 17A illustrates an exemplary architecture in which a plurality of GPUs 1710(1)-1710(N) is communicatively coupled to a plurality of multi-core processors 1705(1)-1705(M) over high-speed links 1740(1)-1740(N) (e.g., buses, point-to-point interconnects, etc.). In at least one embodiment, high-speed links 1740(1)-1740(N) support a communication throughput of 4 GB/s, 30 GB/s, 80 GB/s or higher. In at least one embodiment, various interconnect protocols may be used including, but not limited to, PCIe 4.0 or 5.0 and NVLink 2.0. In various figures, “N” and “M” represent positive integers, values of which may be different from figure to figure.

[0431] In addition, and in at least one embodiment, two or more of GPUs 1710 are interconnected over high-speed links 1729(1)-1729(2), which may be implemented using similar or different protocols/links than those used for high-speed links 1740(1)-1740(N). Similarly, two or more of multi-core processors 1705 may be connected over a high-speed link 1728 which may be symmetric multi-processor (SMP) buses operating at 20 GB/s, 30 GB/s, 120 GB/s or higher. Alternatively, all communication between various system components shown in FIG. 17A may be accomplished using similar protocols/links (e.g., over a common interconnection fabric).

coupled to a processor memory 1701(1)-1701(M), via memory interconnects 1726(1)-1726(M), respectively, and each GPU 1710(1)-1710(N) is communicatively coupled to GPU memory 1720(1)-1720(N) over GPU memory interconnects 1750(1)-1750(N), respectively.

5 In at least one embodiment, memory interconnects 1726 and 1750 may utilize similar or different memory access technologies. By way of example, and not limitation, processor memories 1701(1)-1701(M) and GPU memories 1720 may be volatile memories such as dynamic random access memories (DRAMs) (including stacked DRAMs), Graphics DDR SDRAM (GDDR) (e.g., GDDR5, GDDR6), or High Bandwidth Memory (HBM) and/or may
10 be non-volatile memories such as 3D XPoint or Nano-Ram. In at least one embodiment, some portion of processor memories 1701 may be volatile memory and another portion may be non-volatile memory (e.g., using a two-level memory (2LM) hierarchy).

[0433] As described herein, although various multi-core processors 1705 and GPUs 1710 may be physically coupled to a particular memory 1701, 1720, respectively, and/or a unified
15 memory architecture may be implemented in which a virtual system address space (also referred to as “effective address” space) is distributed among various physical memories. For example, processor memories 1701(1)-1701(M) may each comprise 64 GB of system memory address space and GPU memories 1720(1)-1720(N) may each comprise 32 GB of system memory address space resulting in a total of 256 GB addressable memory when M=2
20 and N=4. Other values for N and M are possible.

[0434] FIG. 17B illustrates additional details for an interconnection between a multi-core processor 1707 and a graphics acceleration module 1746 in accordance with one exemplary embodiment. In at least one embodiment, graphics acceleration module 1746 may include one or more GPU chips integrated on a line card which is coupled to processor 1707 via high-
25 speed link 1740 (e.g., a PCIe bus, NVLink, etc.). In at least one embodiment, graphics acceleration module 1746 may alternatively be integrated on a package or chip with processor 1707.

[0435] In at least one embodiment, processor 1707 includes a plurality of cores 1760A-1760D, each with a translation lookaside buffer (“TLB”) 1761A-1761D and one or more
30 caches 1762A-1762D. In at least one embodiment, cores 1760A-1760D may include various other components for executing instructions and processing data that are not illustrated. In at least one embodiment, caches 1762A-1762D may comprise Level 1 (L1) and Level 2 (L2)

1762D and shared by sets of cores 1760A-1760D. For example, one embodiment of processor 1707 includes 24 cores, each with its own L1 cache, twelve shared L2 caches, and twelve shared L3 caches. In this embodiment, one or more L2 and L3 caches are shared by two adjacent cores. In at least one embodiment, processor 1707 and graphics acceleration module 1746 connect with system memory 1714, which may include processor memories 1701(1)-1701(M) of FIG. 17A.

[0436] In at least one embodiment, coherency is maintained for data and instructions stored in various caches 1762A-1762D, 1756 and system memory 1714 via inter-core communication over a coherence bus 1764. In at least one embodiment, for example, each cache may have cache coherency logic/circuitry associated therewith to communicate to over coherence bus 1764 in response to detected reads or writes to particular cache lines. In at least one embodiment, a cache snooping protocol is implemented over coherence bus 1764 to snoop cache accesses.

[0437] In at least one embodiment, a proxy circuit 1725 communicatively couples graphics acceleration module 1746 to coherence bus 1764, allowing graphics acceleration module 1746 to participate in a cache coherence protocol as a peer of cores 1760A-1760D. In particular, in at least one embodiment, an interface 1735 provides connectivity to proxy circuit 1725 over high-speed link 1740 and an interface 1737 connects graphics acceleration module 1746 to high-speed link 1740.

[0438] In at least one embodiment, an accelerator integration circuit 1736 provides cache management, memory access, context management, and interrupt management services on behalf of a plurality of graphics processing engines 1731(1)-1731(N) of graphics acceleration module 1746. In at least one embodiment, graphics processing engines 1731(1)-1731(N) may each comprise a separate graphics processing unit (GPU). In at least one embodiment, graphics processing engines 1731(1)-1731(N) alternatively may comprise different types of graphics processing engines within a GPU, such as graphics execution units, media processing engines (e.g., video encoders/decoders), samplers, and blit engines. In at least one embodiment, graphics acceleration module 1746 may be a GPU with a plurality of graphics processing engines 1731(1)-1731(N) or graphics processing engines 1731(1)-1731(N) may be individual GPUs integrated on a common package, line card, or chip.

management unit (MMU) 1739 for performing various memory management functions such as virtual-to-physical memory translations (also referred to as effective-to-real memory translations) and memory access protocols for accessing system memory 1714. In at least one embodiment, MMU 1739 may also include a translation lookaside buffer (TLB) (not shown) for caching virtual/effective to physical/real address translations. In at least one embodiment, a cache 1738 can store commands and data for efficient access by graphics processing engines 1731(1)-1731(N). In at least one embodiment, data stored in cache 1738 and graphics memories 1733(1)-1733(M) is kept coherent with core caches 1762A-1762D, 1756 and system memory 1714, possibly using a fetch unit 1744. As mentioned, this may be accomplished via proxy circuit 1725 on behalf of cache 1738 and memories 1733(1)-1733(M) (e.g., sending updates to cache 1738 related to modifications/accesses of cache lines on processor caches 1762A-1762D, 1756 and receiving updates from cache 1738).

[0440] In at least one embodiment, a set of registers 1745 store context data for threads executed by graphics processing engines 1731(1)-1731(N) and a context management circuit 1748 manages thread contexts. For example, context management circuit 1748 may perform save and restore operations to save and restore contexts of various threads during contexts switches (e.g., where a first thread is saved and a second thread is stored so that a second thread can be executed by a graphics processing engine). For example, on a context switch, context management circuit 1748 may store current register values to a designated region in memory (e.g., identified by a context pointer). It may then restore register values when returning to a context. In at least one embodiment, an interrupt management circuit 1747 receives and processes interrupts received from system devices.

[0441] In at least one embodiment, virtual/effective addresses from a graphics processing engine 1731 are translated to real/physical addresses in system memory 1714 by MMU 1739. In at least one embodiment, accelerator integration circuit 1736 supports multiple (e.g., 4, 8, 16) graphics accelerator modules 1746 and/or other accelerator devices. In at least one embodiment, graphics accelerator module 1746 may be dedicated to a single application executed on processor 1707 or may be shared between multiple applications. In at least one embodiment, a virtualized graphics execution environment is presented in which resources of graphics processing engines 1731(1)-1731(N) are shared with multiple applications or virtual machines (VMs). In at least one embodiment, resources may be subdivided into “slices”

and priorities associated with VMs and/or applications.

[0442] In at least one embodiment, accelerator integration circuit 1736 performs as a bridge to a system for graphics acceleration module 1746 and provides address translation and system memory cache services. In addition, in at least one embodiment, accelerator integration circuit 1736 may provide virtualization facilities for a host processor to manage virtualization of graphics processing engines 1731(1)-1731(N), interrupts, and memory management.

[0443] In at least one embodiment, because hardware resources of graphics processing engines 1731(1)-1731(N) are mapped explicitly to a real address space seen by host processor 1707, any host processor can address these resources directly using an effective address value. In at least one embodiment, one function of accelerator integration circuit 1736 is physical separation of graphics processing engines 1731(1)-1731(N) so that they appear to a system as independent units.

[0444] In at least one embodiment, one or more graphics memories 1733(1)-1733(M) are coupled to each of graphics processing engines 1731(1)-1731(N), respectively and $N=M$. In at least one embodiment, graphics memories 1733(1)-1733(M) store instructions and data being processed by each of graphics processing engines 1731(1)-1731(N). In at least one embodiment, graphics memories 1733(1)-1733(M) may be volatile memories such as DRAMs (including stacked DRAMs), GDDR memory (e.g., GDDR5, GDDR6), or HBM, and/or may be non-volatile memories such as 3D XPoint or Nano-Ram.

[0445] In at least one embodiment, to reduce data traffic over high-speed link 1740, biasing techniques can be used to ensure that data stored in graphics memories 1733(1)-1733(M) is data that will be used most frequently by graphics processing engines 1731(1)-1731(N) and preferably not used by cores 1760A-1760D (at least not frequently). Similarly, in at least one embodiment, a biasing mechanism attempts to keep data needed by cores (and preferably not graphics processing engines 1731(1)-1731(N)) within caches 1762A-1762D, 1756 and system memory 1714.

[0446] FIG. 17C illustrates another exemplary embodiment in which accelerator integration circuit 1736 is integrated within processor 1707. In this embodiment, graphics processing engines 1731(1)-1731(N) communicate directly over high-speed link 1740 to accelerator integration circuit 1736 via interface 1737 and interface 1735 (which, again, may be any form

may perform similar operations as those described with respect to FIG. 17B, but potentially at a higher throughput given its close proximity to coherence bus 1764 and caches 1762A-1762D, 1756. In at least one embodiment, an accelerator integration circuit supports different programming models including a dedicated-process programming model (no graphics acceleration module virtualization) and shared programming models (with virtualization), which may include programming models which are controlled by accelerator integration circuit 1736 and programming models which are controlled by graphics acceleration module 1746.

[0447] In at least one embodiment, graphics processing engines 1731(1)-1731(N) are dedicated to a single application or process under a single operating system. In at least one embodiment, a single application can funnel other application requests to graphics processing engines 1731(1)-1731(N), providing virtualization within a VM/partition.

[0448] In at least one embodiment, graphics processing engines 1731(1)-1731(N), may be shared by multiple VM/application partitions. In at least one embodiment, shared models may use a system hypervisor to virtualize graphics processing engines 1731(1)-1731(N) to allow access by each operating system. In at least one embodiment, for single-partition systems without a hypervisor, graphics processing engines 1731(1)-1731(N) are owned by an operating system. In at least one embodiment, an operating system can virtualize graphics processing engines 1731(1)-1731(N) to provide access to each process or application.

[0449] In at least one embodiment, graphics acceleration module 1746 or an individual graphics processing engine 1731(1)-1731(N) selects a process element using a process handle. In at least one embodiment, process elements are stored in system memory 1714 and are addressable using an effective address to real address translation technique described herein. In at least one embodiment, a process handle may be an implementation-specific value provided to a host process when registering its context with graphics processing engine 1731(1)-1731(N) (that is, calling system software to add a process element to a process element linked list). In at least one embodiment, a lower 16-bits of a process handle may be an offset of a process element within a process element linked list.

[0450] FIG. 17D illustrates an exemplary accelerator integration slice 1790. In at least one embodiment, a “slice” comprises a specified portion of processing resources of accelerator integration circuit 1736. In at least one embodiment, an application is effective address space

process elements 1783 are stored in response to GPU invocations 1781 from applications 1780 executed on processor 1707. In at least one embodiment, a process element 1783 contains process state for corresponding application 1780. In at least one embodiment, a work descriptor (WD) 1784 contained in process element 1783 can be a single job requested by an application or may contain a pointer to a queue of jobs. In at least one embodiment, WD 1784 is a pointer to a job request queue in an application's effective address space 1782.

[0451] In at least one embodiment, graphics acceleration module 1746 and/or individual graphics processing engines 1731(1)-1731(N) can be shared by all or a subset of processes in a system. In at least one embodiment, an infrastructure for setting up process states and sending a WD 1784 to a graphics acceleration module 1746 to start a job in a virtualized environment may be included.

[0452] In at least one embodiment, a dedicated-process programming model is implementation-specific. In at least one embodiment, in this model, a single process owns graphics acceleration module 1746 or an individual graphics processing engine 1731. In at least one embodiment, when graphics acceleration module 1746 is owned by a single process, a hypervisor initializes accelerator integration circuit 1736 for an owning partition and an operating system initializes accelerator integration circuit 1736 for an owning process when graphics acceleration module 1746 is assigned.

[0453] In at least one embodiment, in operation, a WD fetch unit 1791 in accelerator integration slice 1790 fetches next WD 1784, which includes an indication of work to be done by one or more graphics processing engines of graphics acceleration module 1746. In at least one embodiment, data from WD 1784 may be stored in registers 1745 and used by MMU 1739, interrupt management circuit 1747 and/or context management circuit 1748 as illustrated. For example, one embodiment of MMU 1739 includes segment/page walk circuitry for accessing segment/page tables 1786 within an OS virtual address space 1785. In at least one embodiment, interrupt management circuit 1747 may process interrupt events 1792 received from graphics acceleration module 1746. In at least one embodiment, when performing graphics operations, an effective address 1793 generated by a graphics processing engine 1731(1)-1731(N) is translated to a real address by MMU 1739.

[0454] In at least one embodiment, registers 1745 are duplicated for each graphics processing engine 1731(1)-1731(N) and/or graphics acceleration module 1746 and may be

duplicated registers may be included in an accelerator integration slice 1790. Exemplary registers that may be initialized by a hypervisor are shown in Table 1.

Table 1 – Hypervisor Initialized Registers

<u>Register #</u>	<u>Description</u>
1	Slice Control Register
2	Real Address (RA) Scheduled Processes Area Pointer
3	Authority Mask Override Register
4	Interrupt Vector Table Entry Offset
5	Interrupt Vector Table Entry Limit
6	State Register
7	Logical Partition ID
8	Real address (RA) Hypervisor Accelerator Utilization Record Pointer
9	Storage Description Register

[0455] Exemplary registers that may be initialized by an operating system are shown in

5 Table 2.

Table 2 – Operating System Initialized Registers

<u>Register #</u>	<u>Description</u>
1	Process and Thread Identification
2	Effective Address (EA) Context Save/Restore Pointer
3	Virtual Address (VA) Accelerator Utilization Record Pointer
4	Virtual Address (VA) Storage Segment Table Pointer
5	Authority Mask
6	Work descriptor

[0456] In at least one embodiment, each WD 1784 is specific to a particular graphics acceleration module 1746 and/or graphics processing engines 1731(1)-1731(N). In at least one embodiment, it contains all information required by a graphics processing engine 1731(1)-1731(N) to do work, or it can be a pointer to a memory location where an application

10 has set up a command queue of work to be completed.

model. This embodiment includes a hypervisor real address space 1798 in which a process element list 1799 is stored. In at least one embodiment, hypervisor real address space 1798 is accessible via a hypervisor 1796 which virtualizes graphics acceleration module engines for operating system 1795.

[0458] In at least one embodiment, shared programming models allow for all or a subset of processes from all or a subset of partitions in a system to use a graphics acceleration module 1746. In at least one embodiment, there are two programming models where graphics acceleration module 1746 is shared by multiple processes and partitions, namely time-sliced shared and graphics directed shared.

[0459] In at least one embodiment, in this model, system hypervisor 1796 owns graphics acceleration module 1746 and makes its function available to all operating systems 1795. In at least one embodiment, for a graphics acceleration module 1746 to support virtualization by system hypervisor 1796, graphics acceleration module 1746 may adhere to certain requirements, such as (1) an application's job request must be autonomous (that is, state does not need to be maintained between jobs), or graphics acceleration module 1746 must provide a context save and restore mechanism, (2) an application's job request is guaranteed by graphics acceleration module 1746 to complete in a specified amount of time, including any translation faults, or graphics acceleration module 1746 provides an ability to preempt processing of a job, and (3) graphics acceleration module 1746 must be guaranteed fairness between processes when operating in a directed shared programming model.

[0460] In at least one embodiment, application 1780 is required to make an operating system 1795 system call with a graphics acceleration module type, a work descriptor (WD), an authority mask register (AMR) value, and a context save/restore area pointer (CSR). In at least one embodiment, graphics acceleration module type describes a targeted acceleration function for a system call. In at least one embodiment, graphics acceleration module type may be a system-specific value. In at least one embodiment, WD is formatted specifically for graphics acceleration module 1746 and can be in a form of a graphics acceleration module 1746 command, an effective address pointer to a user-defined structure, an effective address pointer to a queue of commands, or any other data structure to describe work to be done by graphics acceleration module 1746.

process. In at least one embodiment, a value passed to an operating system is similar to an application setting an AMR. In at least one embodiment, if accelerator integration circuit 1736 (not shown) and graphics acceleration module 1746 implementations do not support a User Authority Mask Override Register (UAMOR), an operating system may apply a current UAMOR value to an AMR value before passing an AMR in a hypervisor call. In at least one embodiment, hypervisor 1796 may optionally apply a current Authority Mask Override Register (AMOR) value before placing an AMR into process element 1783. In at least one embodiment, CSRP is one of registers 1745 containing an effective address of an area in an application's effective address space 1782 for graphics acceleration module 1746 to save and restore context state. In at least one embodiment, this pointer is optional if no state is required to be saved between jobs or when a job is preempted. In at least one embodiment, context save/restore area may be pinned system memory.

[0462] Upon receiving a system call, operating system 1795 may verify that application 1780 has registered and been given authority to use graphics acceleration module 1746. In at least one embodiment, operating system 1795 then calls hypervisor 1796 with information shown in Table 3.

Table 3 – OS to Hypervisor Call Parameters

<u>Parameter #</u>	<u>Description</u>
1	A work descriptor (WD)
2	An Authority Mask Register (AMR) value (potentially masked)
3	An effective address (EA) Context Save/Restore Area Pointer (CSRP)
4	A process ID (PID) and optional thread ID (TID)
5	A virtual address (VA) accelerator utilization record pointer (AURP)
6	Virtual address of storage segment table pointer (SSTP)
7	A logical interrupt service number (LISN)

[0463] In at least one embodiment, upon receiving a hypervisor call, hypervisor 1796 verifies that operating system 1795 has registered and been given authority to use graphics acceleration module 1746. In at least one embodiment, hypervisor 1796 then puts process element 1783 into a process element linked list for a corresponding graphics acceleration module 1746 type. In at least one embodiment, a process element may include information shown in Table 4.

<u>Element #</u>	<u>Description</u>
1	A work descriptor (WD)
2	An Authority Mask Register (AMR) value (potentially masked).
3	An effective address (EA) Context Save/Restore Area Pointer (CSRP)
4	A process ID (PID) and optional thread ID (TID)
5	A virtual address (VA) accelerator utilization record pointer (AURP)
6	Virtual address of storage segment table pointer (SSTP)
7	A logical interrupt service number (LISN)
8	Interrupt vector table, derived from hypervisor call parameters
9	A state register (SR) value
10	A logical partition ID (LPID)
11	A real address (RA) hypervisor accelerator utilization record pointer
12	Storage Descriptor Register (SDR)

[0464] In at least one embodiment, hypervisor initializes a plurality of accelerator integration slice 1790 registers 1745.

[0465] As illustrated in FIG. 17F, in at least one embodiment, a unified memory is used, addressable via a common virtual memory address space used to access physical processor memories 1701(1)-1701(N) and GPU memories 1720(1)-1720(N). In this implementation, operations executed on GPUs 1710(1)-1710(N) utilize a same virtual/effective memory address space to access processor memories 1701(1)-1701(M) and vice versa, thereby simplifying programmability. In at least one embodiment, a first portion of a virtual/effective address space is allocated to processor memory 1701(1), a second portion to second processor memory 1701(N), a third portion to GPU memory 1720(1), and so on. In at least one embodiment, an entire virtual/effective memory space (sometimes referred to as an effective address space) is thereby distributed across each of processor memories 1701 and GPU memories 1720, allowing any processor or GPU to access any physical memory with a virtual address mapped to that memory.

[0466] In at least one embodiment, bias/coherence management circuitry 1794A-1794E within one or more of MMUs 1739A-1739E ensures cache coherence between caches of one or more host processors (e.g., 1705) and GPUs 1710 and implements biasing techniques indicating physical memories in which certain types of data should be stored. In at least one

are illustrated in FIG. 17F, bias/coherence circuitry may be implemented within an MMU of one or more host processors 1705 and/or within accelerator integration circuit 1736.

[0467] One embodiment allows GPU memories 1720 to be mapped as part of system memory, and accessed using shared virtual memory (SVM) technology, but without suffering performance drawbacks associated with full system cache coherence. In at least one embodiment, an ability for GPU memories 1720 to be accessed as system memory without onerous cache coherence overhead provides a beneficial operating environment for GPU offload. In at least one embodiment, this arrangement allows software of host processor 1705 to setup operands and access computation results, without overhead of traditional I/O DMA data copies. In at least one embodiment, such traditional copies involve driver calls, interrupts and memory mapped I/O (MMIO) accesses that are all inefficient relative to simple memory accesses. In at least one embodiment, an ability to access GPU memories 1720 without cache coherence overheads can be critical to execution time of an offloaded computation. In at least one embodiment, in cases with substantial streaming write memory traffic, for example, cache coherence overhead can significantly reduce an effective write bandwidth seen by a GPU 1710. In at least one embodiment, efficiency of operand setup, efficiency of results access, and efficiency of GPU computation may play a role in determining effectiveness of a GPU offload.

[0468] In at least one embodiment, selection of GPU bias and host processor bias is driven by a bias tracker data structure. In at least one embodiment, a bias table may be used, for example, which may be a page-granular structure (e.g., controlled at a granularity of a memory page) that includes 1 or 2 bits per GPU-attached memory page. In at least one embodiment, a bias table may be implemented in a stolen memory range of one or more GPU memories 1720, with or without a bias cache in a GPU 1710 (e.g., to cache frequently/recently used entries of a bias table). Alternatively, in at least one embodiment, an entire bias table may be maintained within a GPU.

[0469] In at least one embodiment, a bias table entry associated with each access to a GPU attached memory 1720 is accessed prior to actual access to a GPU memory, causing following operations. In at least one embodiment, local requests from a GPU 1710 that find their page in GPU bias are forwarded directly to a corresponding GPU memory 1720. In at least one embodiment, local requests from a GPU that find their page in host bias are

embodiment, requests from processor 1705 that find a requested page in host processor bias complete a request like a normal memory read. Alternatively, requests directed to a GPU-biased page may be forwarded to a GPU 1710. In at least one embodiment, a GPU may then
5 transition a page to a host processor bias if it is not currently using a page. In at least one embodiment, a bias state of a page can be changed either by a software-based mechanism, a hardware-assisted software-based mechanism, or, for a limited set of cases, a purely hardware-based mechanism.

[0470] In at least one embodiment, one mechanism for changing bias state employs an API
10 call (e.g., OpenCL), which, in turn, calls a GPU's device driver which, in turn, sends a message (or enqueues a command descriptor) to a GPU directing it to change a bias state and, for some transitions, perform a cache flushing operation in a host. In at least one embodiment, a cache flushing operation is used for a transition from host processor 1705 bias to GPU bias, but is not for an opposite transition.

[0471] In at least one embodiment, cache coherency is maintained by temporarily rendering GPU-biased pages uncacheable by host processor 1705. In at least one embodiment, to
15 access these pages, processor 1705 may request access from GPU 1710, which may or may not grant access right away. In at least one embodiment, thus, to reduce communication between processor 1705 and GPU 1710 it is beneficial to ensure that GPU-biased pages are
20 those which are required by a GPU but not host processor 1705 and vice versa.

[0472] Hardware structure(s) 915 are used to perform one or more embodiments. Details regarding a hardware structure(s) 915 may be provided herein in conjunction with FIGS. 9A and/or 9B.

[0473] FIG. 18 illustrates exemplary integrated circuits and associated graphics processors
25 that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included in at least one embodiment, including additional graphics processors/cores, peripheral interface controllers, or general-purpose processor cores.

[0474] FIG. 18 is a block diagram illustrating an exemplary system on a chip integrated
30 circuit 1800 that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, integrated circuit 1800 includes one or more application processor(s) 1805 (e.g., CPUs), at least one graphics processor 1810, and may

may be a modular IP core. In at least one embodiment, integrated circuit 1800 includes peripheral or bus logic including a USB controller 1825, a UART controller 1830, an SPI/SDIO controller 1835, and an I²S/I²C controller 1840. In at least one embodiment, integrated circuit 1800 can include a display device 1845 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1850 and a mobile industry processor interface (MIPI) display interface 1855. In at least one embodiment, storage may be provided by a flash memory subsystem 1860 including flash memory and a flash memory controller. In at least one embodiment, a memory interface may be provided via a memory controller 1865 for access to SDRAM or SRAM memory devices. In at least one embodiment, some integrated circuits additionally include an embedded security engine 1870.

[0475] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in integrated circuit 1800 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0476] FIGS. 19A-19B illustrate exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included in at least one embodiment, including additional graphics processors/cores, peripheral interface controllers, or general-purpose processor cores.

[0477] FIGS. 19A-19B are block diagrams illustrating exemplary graphics processors for use within an SoC, according to embodiments described herein. FIG. 19A illustrates an exemplary graphics processor 1910 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. FIG. 19B illustrates an additional exemplary graphics processor 1940 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, graphics processor 1910 of FIG. 19A is a low power graphics processor core. In at least one embodiment, graphics processor 1940 of FIG.

graphics processors 1910, 1940 can be variants of graphics processor 1810 of FIG. 18.

[0478] In at least one embodiment, graphics processor 1910 includes a vertex processor 1905 and one or more fragment processor(s) 1915A-1915N (e.g., 1915A, 1915B, 1915C, 1915D, through 1915N-1, and 1915N). In at least one embodiment, graphics processor 1910 can execute different shader programs via separate logic, such that vertex processor 1905 is optimized to execute operations for vertex shader programs, while one or more fragment processor(s) 1915A-1915N execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. In at least one embodiment, vertex processor 1905 performs a vertex processing stage of a 3D graphics pipeline and generates primitives and vertex data. In at least one embodiment, fragment processor(s) 1915A-1915N use primitive and vertex data generated by vertex processor 1905 to produce a framebuffer that is displayed on a display device. In at least one embodiment, fragment processor(s) 1915A-1915N are optimized to execute fragment shader programs as provided for in an OpenGL API, which may be used to perform similar operations as a pixel shader program as provided for in a Direct 3D API.

[0479] In at least one embodiment, graphics processor 1910 additionally includes one or more memory management units (MMUs) 1920A-1920B, cache(s) 1925A-1925B, and circuit interconnect(s) 1930A-1930B. In at least one embodiment, one or more MMU(s) 1920A-1920B provide for virtual to physical address mapping for graphics processor 1910, including for vertex processor 1905 and/or fragment processor(s) 1915A-1915N, which may reference vertex or image/texture data stored in memory, in addition to vertex or image/texture data stored in one or more cache(s) 1925A-1925B. In at least one embodiment, one or more MMU(s) 1920A-1920B may be synchronized with other MMUs within a system, including one or more MMUs associated with one or more application processor(s) 1805, image processors 1815, and/or video processors 1820 of FIG. 18, such that each processor 1805-1820 can participate in a shared or unified virtual memory system. In at least one embodiment, one or more circuit interconnect(s) 1930A-1930B enable graphics processor 1910 to interface with other IP cores within SoC, either via an internal bus of SoC or via a direct connection.

[0480] In at least one embodiment, graphics processor 1940 includes one or more shader core(s) 1955A-1955N (e.g., 1955A, 1955B, 1955C, 1955D, 1955E, 1955F, through 1955N-1, and 1955N) as shown in FIG. 19B, which provides for a unified shader core architecture in

including shader program code to implement vertex shaders, fragment shaders, and/or compute shaders. In at least one embodiment, a number of shader cores can vary. In at least one embodiment, graphics processor 1940 includes an inter-core task manager 1945, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores 1955A-1955N and a tiling unit 1958 to accelerate tiling operations for tile-based rendering, in which rendering operations for a scene are subdivided in image space, for example to exploit local spatial coherence within a scene or to optimize use of internal caches.

[0481] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in integrated circuit 19A and/or 19B for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0482] FIGS. 20A-20B illustrate additional exemplary graphics processor logic according to embodiments described herein. FIG. 20A illustrates a graphics core 2000 that may be included within graphics processor 1810 of FIG. 18, in at least one embodiment, and may be a unified shader core 1955A-1955N as in FIG. 19B in at least one embodiment. FIG. 20B illustrates a highly-parallel general-purpose graphics processing unit (“GPGPU”) 2030 suitable for deployment on a multi-chip module in at least one embodiment.

[0483] In at least one embodiment, graphics core 2000 includes a shared instruction cache 2002, a texture unit 2018, and a cache/shared memory 2020 that are common to execution resources within graphics core 2000. In at least one embodiment, graphics core 2000 can include multiple slices 2001A-2001N or a partition for each core, and a graphics processor can include multiple instances of graphics core 2000. In at least one embodiment, slices 2001A-2001N can include support logic including a local instruction cache 2004A-2004N, a thread scheduler 2006A-2006N, a thread dispatcher 2008A-2008N, and a set of registers 2010A-2010N. In at least one embodiment, slices 2001A-2001N can include a set of additional function units (AFUs 2012A-2012N), floating-point units (FPUs 2014A-2014N), integer arithmetic logic units (ALUs 2016A-2016N), address computational units (ACUs

processing units (MPUs 2017A-2017N).

[0484] In at least one embodiment, FPUs 2014A-2014N can perform single-precision (32-bit) and half-precision (16-bit) floating point operations, while DPFPU 2015A-2015N

5 perform double precision (64-bit) floating point operations. In at least one embodiment, ALUs 2016A-2016N can perform variable precision integer operations at 8-bit, 16-bit, and 32-bit precision, and can be configured for mixed precision operations. In at least one embodiment, MPUs 2017A-2017N can also be configured for mixed precision matrix operations, including half-precision floating point and 8-bit integer operations. In at least one
10 embodiment, MPUs 2017-2017N can perform a variety of matrix operations to accelerate machine learning application frameworks, including enabling support for accelerated general matrix to matrix multiplication (GEMM). In at least one embodiment, AFUs 2012A-2012N can perform additional logic operations not supported by floating-point or integer units, including trigonometric operations (e.g., sine, cosine, etc.).

15 [0485] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in graphics core 2000 for inferencing or predicting operations based, at least in part, on weight parameters calculated
20 using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0486] FIG. 20B illustrates a general-purpose processing unit (GPGPU) 2030 that can be configured to enable highly-parallel compute operations to be performed by an array of graphics processing units, in at least one embodiment. In at least one embodiment, GPGPU
25 2030 can be linked directly to other instances of GPGPU 2030 to create a multi-GPU cluster to improve training speed for deep neural networks. In at least one embodiment, GPGPU 2030 includes a host interface 2032 to enable a connection with a host processor. In at least one embodiment, host interface 2032 is a PCI Express interface. In at least one embodiment, host interface 2032 can be a vendor-specific communications interface or communications
30 fabric. In at least one embodiment, GPGPU 2030 receives commands from a host processor and uses a global scheduler 2034 to distribute execution threads associated with those commands to a set of compute clusters 2036A-2036H. In at least one embodiment, compute

memory 2038 can serve as a higher-level cache for cache memories within compute clusters 2036A-2036H.

[0487] In at least one embodiment, GPGPU 2030 includes memory 2044A-2044B coupled with compute clusters 2036A-2036H via a set of memory controllers 2042A-2042B. In at least one embodiment, memory 2044A-2044B can include various types of memory devices including dynamic random access memory (DRAM) or graphics random access memory, such as synchronous graphics random access memory (SGRAM), including graphics double data rate (GDDR) memory.

[0488] In at least one embodiment, compute clusters 2036A-2036H each include a set of graphics cores, such as graphics core 2000 of FIG. 20A, which can include multiple types of integer and floating point logic units that can perform computational operations at a range of precisions including suited for machine learning computations. For example, in at least one embodiment, at least a subset of floating point units in each of compute clusters 2036A-2036H can be configured to perform 16-bit or 32-bit floating point operations, while a different subset of floating point units can be configured to perform 64-bit floating point operations.

[0489] In at least one embodiment, multiple instances of GPGPU 2030 can be configured to operate as a compute cluster. In at least one embodiment, communication used by compute clusters 2036A-2036H for synchronization and data exchange varies across embodiments. In at least one embodiment, multiple instances of GPGPU 2030 communicate over host interface 2032. In at least one embodiment, GPGPU 2030 includes an I/O hub 2039 that couples GPGPU 2030 with a GPU link 2040 that enables a direct connection to other instances of GPGPU 2030. In at least one embodiment, GPU link 2040 is coupled to a dedicated GPU-to-GPU bridge that enables communication and synchronization between multiple instances of GPGPU 2030. In at least one embodiment, GPU link 2040 couples with a high-speed interconnect to transmit and receive data to other GPGPUs or parallel processors. In at least one embodiment, multiple instances of GPGPU 2030 are located in separate data processing systems and communicate via a network device that is accessible via host interface 2032. In at least one embodiment GPU link 2040 can be configured to enable a connection to a host processor in addition to or as an alternative to host interface 2032.

networks. In at least one embodiment, GPGPU 2030 can be used within an inferencing platform. In at least one embodiment, in which GPGPU 2030 is used for inferencing, GPGPU 2030 may include fewer compute clusters 2036A-2036H relative to when GPGPU 2030 is used for training a neural network. In at least one embodiment, memory technology associated with memory 2044A-2044B may differ between inferencing and training configurations, with higher bandwidth memory technologies devoted to training configurations. In at least one embodiment, an inferencing configuration of GPGPU 2030 can support inferencing specific instructions. For example, in at least one embodiment, an inferencing configuration can provide support for one or more 8-bit integer dot product instructions, which may be used during inferencing operations for deployed neural networks.

[0491] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in GPGPU 2030 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0492] FIG. 21 is a block diagram illustrating a computing system 2100 according to at least one embodiment. In at least one embodiment, computing system 2100 includes a processing subsystem 2101 having one or more processor(s) 2102 and a system memory 2104 communicating via an interconnection path that may include a memory hub 2105. In at least one embodiment, memory hub 2105 may be a separate component within a chipset component or may be integrated within one or more processor(s) 2102. In at least one embodiment, memory hub 2105 couples with an I/O subsystem 2111 via a communication link 2106. In at least one embodiment, I/O subsystem 2111 includes an I/O hub 2107 that can enable computing system 2100 to receive input from one or more input device(s) 2108. In at least one embodiment, I/O hub 2107 can enable a display controller, which may be included in one or more processor(s) 2102, to provide outputs to one or more display device(s) 2110A. In at least one embodiment, one or more display device(s) 2110A coupled with I/O hub 2107 can include a local, internal, or embedded display device.

parallel processor(s) 2112 coupled to memory hub 2105 via a bus or other communication link 2113. In at least one embodiment, communication link 2113 may use one of any number of standards based communication link technologies or protocols, such as, but not limited to
5 PCI Express, or may be a vendor-specific communications interface or communications fabric. In at least one embodiment, one or more parallel processor(s) 2112 form a computationally focused parallel or vector processing system that can include a large number of processing cores and/or processing clusters, such as a many-integrated core (MIC) processor. In at least one embodiment, some or all of parallel processor(s) 2112 form a
10 graphics processing subsystem that can output pixels to one of one or more display device(s) 2110A coupled via I/O Hub 2107. In at least one embodiment, parallel processor(s) 2112 can also include a display controller and display interface (not shown) to enable a direct connection to one or more display device(s) 2110B.

[0494] In at least one embodiment, a system storage unit 2114 can connect to I/O hub 2107
15 to provide a storage mechanism for computing system 2100. In at least one embodiment, an I/O switch 2116 can be used to provide an interface mechanism to enable connections between I/O hub 2107 and other components, such as a network adapter 2118 and/or a wireless network adapter 2119 that may be integrated into platform, and various other devices that can be added via one or more add-in device(s) 2120. In at least one embodiment,
20 network adapter 2118 can be an Ethernet adapter or another wired network adapter. In at least one embodiment, wireless network adapter 2119 can include one or more of a Wi-Fi, Bluetooth, near field communication (NFC), or other network device that includes one or more wireless radios.

[0495] In at least one embodiment, computing system 2100 can include other components
25 not explicitly shown, including USB or other port connections, optical storage drives, video capture devices, and like, may also be connected to I/O hub 2107. In at least one embodiment, communication paths interconnecting various components in FIG. 21 may be implemented using any suitable protocols, such as PCI (Peripheral Component Interconnect) based protocols (e.g., PCI-Express), or other bus or point-to-point communication interfaces
30 and/or protocol(s), such as NV-Link high-speed interconnect, or interconnect protocols.

[0496] In at least one embodiment, parallel processor(s) 2112 incorporate circuitry optimized for graphics and video processing, including, for example, video output circuitry,

processor(s) 2112 incorporate circuitry optimized for general purpose processing. In at least embodiment, components of computing system 2100 may be integrated with one or more other system elements on a single integrated circuit. For example, in at least one
5 embodiment, parallel processor(s) 2112, memory hub 2105, processor(s) 2102, and I/O hub 2107 can be integrated into a system on chip (SoC) integrated circuit. In at least one embodiment, components of computing system 2100 can be integrated into a single package to form a system in package (SIP) configuration. In at least one embodiment, at least a portion of components of computing system 2100 can be integrated into a multi-chip module
10 (MCM), which can be interconnected with other multi-chip modules into a modular computing system.

[0497] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least
15 one embodiment, inference and/or training logic 915 may be used in system FIG. 2100 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

PROCESSORS

[0498] FIG. 22A illustrates a parallel processor 2200 according to at least one embodiment. In at least one embodiment, various components of parallel processor 2200 may be implemented using one or more integrated circuit devices, such as programmable processors, application specific integrated circuits (ASICs), or field programmable gate arrays (FPGA). In at least one embodiment, illustrated parallel processor 2200 is a variant of one or more
25 parallel processor(s) 2112 shown in FIG. 21 according to an exemplary embodiment.

[0499] In at least one embodiment, parallel processor 2200 includes a parallel processing unit 2202. In at least one embodiment, parallel processing unit 2202 includes an I/O unit 2204 that enables communication with other devices, including other instances of parallel processing unit 2202. In at least one embodiment, I/O unit 2204 may be directly connected to
30 other devices. In at least one embodiment, I/O unit 2204 connects with other devices via use of a hub or switch interface, such as a memory hub 2205. In at least one embodiment, connections between memory hub 2205 and I/O unit 2204 form a communication link 2213.

crossbar 2216, where host interface 2206 receives commands directed to performing processing operations and memory crossbar 2216 receives commands directed to performing memory operations.

- 5 [0500] In at least one embodiment, when host interface 2206 receives a command buffer via I/O unit 2204, host interface 2206 can direct work operations to perform those commands to a front end 2208. In at least one embodiment, front end 2208 couples with a scheduler 2210, which is configured to distribute commands or other work items to a processing cluster array 2212. In at least one embodiment, scheduler 2210 ensures that processing cluster array
- 10 2212 is properly configured and in a valid state before tasks are distributed to a cluster of processing cluster array 2212. In at least one embodiment, scheduler 2210 is implemented via firmware logic executing on a microcontroller. In at least one embodiment, microcontroller implemented scheduler 2210 is configurable to perform complex scheduling and work distribution operations at coarse and fine granularity, enabling rapid preemption
- 15 and context switching of threads executing on processing array 2212. In at least one embodiment, host software can provide workloads for scheduling on processing cluster array 2212 via one of multiple graphics processing paths. In at least one embodiment, workloads can then be automatically distributed across processing array cluster 2212 by scheduler 2210 logic within a microcontroller including scheduler 2210.
- 20 [0501] In at least one embodiment, processing cluster array 2212 can include up to “N” processing clusters (e.g., cluster 2214A, cluster 2214B, through cluster 2214N), where “N” represents a positive integer (which may be a different integer “N” than used in other figures). In at least one embodiment, each cluster 2214A-2214N of processing cluster array 2212 can execute a large number of concurrent threads. In at least one embodiment,
- 25 scheduler 2210 can allocate work to clusters 2214A-2214N of processing cluster array 2212 using various scheduling and/or work distribution algorithms, which may vary depending on workload arising for each type of program or computation. In at least one embodiment, scheduling can be handled dynamically by scheduler 2210, or can be assisted in part by compiler logic during compilation of program logic configured for execution by processing
- 30 cluster array 2212. In at least one embodiment, different clusters 2214A-2214N of processing cluster array 2212 can be allocated for processing different types of programs or for performing different types of computations.

perform various types of parallel processing operations. In at least one embodiment, processing cluster array 2212 is configured to perform general-purpose parallel compute operations. For example, in at least one embodiment, processing cluster array 2212 can
5 include logic to execute processing tasks including filtering of video and/or audio data, performing modeling operations, including physics operations, and performing data transformations.

[0503] In at least one embodiment, processing cluster array 2212 is configured to perform parallel graphics processing operations. In at least one embodiment, processing cluster array
10 2212 can include additional logic to support execution of such graphics processing operations, including but not limited to, texture sampling logic to perform texture operations, as well as tessellation logic and other vertex processing logic. In at least one embodiment, processing cluster array 2212 can be configured to execute graphics processing related shader programs such as, but not limited to, vertex shaders, tessellation shaders, geometry shaders,
15 and pixel shaders. In at least one embodiment, parallel processing unit 2202 can transfer data from system memory via I/O unit 2204 for processing. In at least one embodiment, during processing, transferred data can be stored to on-chip memory (e.g., parallel processor memory 2222) during processing, then written back to system memory.

[0504] In at least one embodiment, when parallel processing unit 2202 is used to perform
20 graphics processing, scheduler 2210 can be configured to divide a processing workload into approximately equal sized tasks, to better enable distribution of graphics processing operations to multiple clusters 2214A-2214N of processing cluster array 2212. In at least one embodiment, portions of processing cluster array 2212 can be configured to perform different types of processing. For example, in at least one embodiment, a first portion may be
25 configured to perform vertex shading and topology generation, a second portion may be configured to perform tessellation and geometry shading, and a third portion may be configured to perform pixel shading or other screen space operations, to produce a rendered image for display. In at least one embodiment, intermediate data produced by one or more of clusters 2214A-2214N may be stored in buffers to allow intermediate data to be transmitted
30 between clusters 2214A-2214N for further processing.

[0505] In at least one embodiment, processing cluster array 2212 can receive processing tasks to be executed via scheduler 2210, which receives commands defining processing tasks

to be processed, e.g., surface (patch) data, primitive data, vertex data, and/or pixel data, as well as state parameters and commands defining how data is to be processed (e.g., what program is to be executed). In at least one embodiment, scheduler 2210 may be configured to
5 fetch indices corresponding to tasks or may receive indices from front end 2208. In at least one embodiment, front end 2208 can be configured to ensure processing cluster array 2212 is configured to a valid state before a workload specified by incoming command buffers (e.g., batch-buffers, push buffers, etc.) is initiated.

[0506] In at least one embodiment, each of one or more instances of parallel processing
10 unit 2202 can couple with a parallel processor memory 2222. In at least one embodiment, parallel processor memory 2222 can be accessed via memory crossbar 2216, which can receive memory requests from processing cluster array 2212 as well as I/O unit 2204. In at least one embodiment, memory crossbar 2216 can access parallel processor memory 2222 via a memory interface 2218. In at least one embodiment, memory interface 2218 can include
15 multiple partition units (e.g., partition unit 2220A, partition unit 2220B, through partition unit 2220N) that can each couple to a portion (e.g., memory unit) of parallel processor memory 2222. In at least one embodiment, a number of partition units 2220A-2220N is configured to be equal to a number of memory units, such that a first partition unit 2220A has a
corresponding first memory unit 2224A, a second partition unit 2220B has a corresponding
20 memory unit 2224B, and an N-th partition unit 2220N has a corresponding N-th memory unit 2224N. In at least one embodiment, a number of partition units 2220A-2220N may not be equal to a number of memory units.

[0507] In at least one embodiment, memory units 2224A-2224N can include various types of memory devices, including dynamic random access memory (DRAM) or graphics random
25 access memory, such as synchronous graphics random access memory (SGRAM), including graphics double data rate (GDDR) memory. In at least one embodiment, memory units 2224A-2224N may also include 3D stacked memory, including but not limited to high bandwidth memory (HBM). In at least one embodiment, render targets, such as frame buffers or texture maps may be stored across memory units 2224A-2224N, allowing partition
30 units 2220A-2220N to write portions of each render target in parallel to efficiently use available bandwidth of parallel processor memory 2222. In at least one embodiment, a local instance of parallel processor memory 2222 may be excluded in favor of a unified memory design that utilizes system memory in conjunction with local cache memory.

array 2212 can process data that will be written to any of memory units 2224A-2224N within parallel processor memory 2222. In at least one embodiment, memory crossbar 2216 can be configured to transfer an output of each cluster 2214A-2214N to any partition unit 2220A-2220N or to another cluster 2214A-2214N, which can perform additional processing operations on an output. In at least one embodiment, each cluster 2214A-2214N can communicate with memory interface 2218 through memory crossbar 2216 to read from or write to various external memory devices. In at least one embodiment, memory crossbar 2216 has a connection to memory interface 2218 to communicate with I/O unit 2204, as well as a connection to a local instance of parallel processor memory 2222, enabling processing units within different processing clusters 2214A-2214N to communicate with system memory or other memory that is not local to parallel processing unit 2202. In at least one embodiment, memory crossbar 2216 can use virtual channels to separate traffic streams between clusters 2214A-2214N and partition units 2220A-2220N.

[0509] In at least one embodiment, multiple instances of parallel processing unit 2202 can be provided on a single add-in card, or multiple add-in cards can be interconnected. In at least one embodiment, different instances of parallel processing unit 2202 can be configured to interoperate even if different instances have different numbers of processing cores, different amounts of local parallel processor memory, and/or other configuration differences. For example, in at least one embodiment, some instances of parallel processing unit 2202 can include higher precision floating point units relative to other instances. In at least one embodiment, systems incorporating one or more instances of parallel processing unit 2202 or parallel processor 2200 can be implemented in a variety of configurations and form factors, including but not limited to desktop, laptop, or handheld personal computers, servers, workstations, game consoles, and/or embedded systems.

[0510] FIG. 22B is a block diagram of a partition unit 2220 according to at least one embodiment. In at least one embodiment, partition unit 2220 is an instance of one of partition units 2220A-2220N of FIG. 22A. In at least one embodiment, partition unit 2220 includes an L2 cache 2221, a frame buffer interface 2225, and a ROP 2226 (raster operations unit). In at least one embodiment, L2 cache 2221 is a read/write cache that is configured to perform load and store operations received from memory crossbar 2216 and ROP 2226. In at least one embodiment, read misses and urgent write-back requests are output by L2 cache 2221 to frame buffer interface 2225 for processing. In at least one embodiment, updates can

embodiment, frame buffer interface 2225 interfaces with one of memory units in parallel processor memory, such as memory units 2224A-2224N of FIG. 22 (e.g., within parallel processor memory 2222).

5 [0511] In at least one embodiment, ROP 2226 is a processing unit that performs raster operations such as stencil, z test, blending, etc. In at least one embodiment, ROP 2226 then outputs processed graphics data that is stored in graphics memory. In at least one embodiment, ROP 2226 includes compression logic to compress depth or color data that is written to memory and decompress depth or color data that is read from memory. In at least one embodiment, compression logic can be lossless compression logic that makes use of one or more of multiple compression algorithms. In at least one embodiment, a type of compression that is performed by ROP 2226 can vary based on statistical characteristics of data to be compressed. For example, in at least one embodiment, delta color compression is performed on depth and color data on a per-tile basis.

15 [0512] In at least one embodiment, ROP 2226 is included within each processing cluster (e.g., cluster 2214A-2214N of FIG. 22A) instead of within partition unit 2220. In at least one embodiment, read and write requests for pixel data are transmitted over memory crossbar 2216 instead of pixel fragment data. In at least one embodiment, processed graphics data may be displayed on a display device, such as one of one or more display device(s) 2110 of FIG. 21, routed for further processing by processor(s) 2102, or routed for further processing by one of processing entities within parallel processor 2200 of FIG. 22A.

[0513] FIG. 22C is a block diagram of a processing cluster 2214 within a parallel processing unit according to at least one embodiment. In at least one embodiment, a processing cluster is an instance of one of processing clusters 2214A-2214N of FIG. 22A. In at least one embodiment, processing cluster 2214 can be configured to execute many threads in parallel, where “thread” refers to an instance of a particular program executing on a particular set of input data. In at least one embodiment, single-instruction, multiple-data (SIMD) instruction issue techniques are used to support parallel execution of a large number of threads without providing multiple independent instruction units. In at least one embodiment, single-instruction, multiple-thread (SIMT) techniques are used to support parallel execution of a large number of generally synchronized threads, using a common

of processing clusters.

[0514] In at least one embodiment, operation of processing cluster 2214 can be controlled via a pipeline manager 2232 that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager 2232 receives instructions from scheduler 2210 of FIG. 22A and manages execution of those instructions via a graphics multiprocessor 2234 and/or a texture unit 2236. In at least one embodiment, graphics multiprocessor 2234 is an exemplary instance of a SIMT parallel processor. However, in at least one embodiment, various types of SIMT parallel processors of differing architectures may be included within processing cluster 2214. In at least one embodiment, one or more instances of graphics multiprocessor 2234 can be included within a processing cluster 2214. In at least one embodiment, graphics multiprocessor 2234 can process data and a data crossbar 2240 can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager 2232 can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data crossbar 2240.

[0515] In at least one embodiment, each graphics multiprocessor 2234 within processing cluster 2214 can include an identical set of functional execution logic (e.g., arithmetic logic units, load-store units, etc.). In at least one embodiment, functional execution logic can be configured in a pipelined manner in which new instructions can be issued before previous instructions are complete. In at least one embodiment, functional execution logic supports a variety of operations including integer and floating point arithmetic, comparison operations, Boolean operations, bit-shifting, and computation of various algebraic functions. In at least one embodiment, same functional-unit hardware can be leveraged to perform different operations and any combination of functional units may be present.

[0516] In at least one embodiment, instructions transmitted to processing cluster 2214 constitute a thread. In at least one embodiment, a set of threads executing across a set of parallel processing engines is a thread group. In at least one embodiment, a thread group executes a common program on different input data. In at least one embodiment, each thread within a thread group can be assigned to a different processing engine within a graphics multiprocessor 2234. In at least one embodiment, a thread group may include fewer threads than a number of processing engines within graphics multiprocessor 2234. In at least one

engines, one or more of processing engines may be idle during cycles in which that thread group is being processed. In at least one embodiment, a thread group may also include more threads than a number of processing engines within graphics multiprocessor 2234. In at least one embodiment, when a thread group includes more threads than number of processing engines within graphics multiprocessor 2234, processing can be performed over consecutive clock cycles. In at least one embodiment, multiple thread groups can be executed concurrently on a graphics multiprocessor 2234.

[0517] In at least one embodiment, graphics multiprocessor 2234 includes an internal cache memory to perform load and store operations. In at least one embodiment, graphics multiprocessor 2234 can forego an internal cache and use a cache memory (e.g., L1 cache 2248) within processing cluster 2214. In at least one embodiment, each graphics multiprocessor 2234 also has access to L2 caches within partition units (e.g., partition units 2220A-2220N of FIG. 22A) that are shared among all processing clusters 2214 and may be used to transfer data between threads. In at least one embodiment, graphics multiprocessor 2234 may also access off-chip global memory, which can include one or more of local parallel processor memory and/or system memory. In at least one embodiment, any memory external to parallel processing unit 2202 may be used as global memory. In at least one embodiment, processing cluster 2214 includes multiple instances of graphics multiprocessor 2234 and can share common instructions and data, which may be stored in L1 cache 2248.

[0518] In at least one embodiment, each processing cluster 2214 may include an MMU 2245 (memory management unit) that is configured to map virtual addresses into physical addresses. In at least one embodiment, one or more instances of MMU 2245 may reside within memory interface 2218 of FIG. 22A. In at least one embodiment, MMU 2245 includes a set of page table entries (PTEs) used to map a virtual address to a physical address of a tile and optionally a cache line index. In at least one embodiment, MMU 2245 may include address translation lookaside buffers (TLB) or caches that may reside within graphics multiprocessor 2234 or L1 2248 cache or processing cluster 2214. In at least one embodiment, a physical address is processed to distribute surface data access locally to allow for efficient request interleaving among partition units. In at least one embodiment, a cache line index may be used to determine whether a request for a cache line is a hit or miss.

each graphics multiprocessor 2234 is coupled to a texture unit 2236 for performing texture mapping operations, e.g., determining texture sample positions, reading texture data, and filtering texture data. In at least one embodiment, texture data is read from an internal texture
5 L1 cache (not shown) or from an L1 cache within graphics multiprocessor 2234 and is fetched from an L2 cache, local parallel processor memory, or system memory, as needed. In at least one embodiment, each graphics multiprocessor 2234 outputs processed tasks to data crossbar 2240 to provide processed task to another processing cluster 2214 for further processing or to store processed task in an L2 cache, local parallel processor memory, or
10 system memory via memory crossbar 2216. In at least one embodiment, a preROP 2242 (pre-raster operations unit) is configured to receive data from graphics multiprocessor 2234, and direct data to ROP units, which may be located with partition units as described herein (e.g., partition units 2220A-2220N of FIG. 22A). In at least one embodiment, preROP 2242 unit can perform optimizations for color blending, organizing pixel color data, and
15 performing address translations.

[0520] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in graphics processing
20 cluster 2214 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0521] FIG. 22D shows a graphics multiprocessor 2234 according to at least one embodiment. In at least one embodiment, graphics multiprocessor 2234 couples with
25 pipeline manager 2232 of processing cluster 2214. In at least one embodiment, graphics multiprocessor 2234 has an execution pipeline including but not limited to an instruction cache 2252, an instruction unit 2254, an address mapping unit 2256, a register file 2258, one or more general purpose graphics processing unit (GPGPU) cores 2262, and one or more load/store units 2266. In at least one embodiment, GPGPU cores 2262 and load/store units
30 2266 are coupled with cache memory 2272 and shared memory 2270 via a memory and cache interconnect 2268.

to execute from pipeline manager 2232. In at least one embodiment, instructions are cached in instruction cache 2252 and dispatched for execution by an instruction unit 2254. In at least one embodiment, instruction unit 2254 can dispatch instructions as thread groups (e.g., warps), with each thread of thread group assigned to a different execution unit within GPGPU cores 2262. In at least one embodiment, an instruction can access any of a local, shared, or global address space by specifying an address within a unified address space. In at least one embodiment, address mapping unit 2256 can be used to translate addresses in a unified address space into a distinct memory address that can be accessed by load/store units 2266.

[0523] In at least one embodiment, register file 2258 provides a set of registers for functional units of graphics multiprocessor 2234. In at least one embodiment, register file 2258 provides temporary storage for operands connected to data paths of functional units (e.g., GPGPU cores 2262, load/store units 2266) of graphics multiprocessor 2234. In at least one embodiment, register file 2258 is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file 2258. In at least one embodiment, register file 2258 is divided between different warps being executed by graphics multiprocessor 2234.

[0524] In at least one embodiment, GPGPU cores 2262 can each include floating point units (FPUs) and/or integer arithmetic logic units (ALUs) that are used to execute instructions of graphics multiprocessor 2234. In at least one embodiment, GPGPU cores 2262 can be similar in architecture or can differ in architecture. In at least one embodiment, a first portion of GPGPU cores 2262 include a single precision FPU and an integer ALU while a second portion of GPGPU cores include a double precision FPU. In at least one embodiment, FPUs can implement IEEE 754-2008 standard floating point arithmetic or enable variable precision floating point arithmetic. In at least one embodiment, graphics multiprocessor 2234 can additionally include one or more fixed function or special function units to perform specific functions such as copy rectangle or pixel blending operations. In at least one embodiment, one or more of GPGPU cores 2262 can also include fixed or special function logic.

[0525] In at least one embodiment, GPGPU cores 2262 include SIMD logic capable of performing a single instruction on multiple sets of data. In at least one embodiment, GPGPU cores 2262 can physically execute SIMD4, SIMD8, and SIMD16 instructions and logically

instructions for GPGPU cores can be generated at compile time by a shader compiler or automatically generated when executing programs written and compiled for single program multiple data (SPMD) or SIMT architectures. In at least one embodiment, multiple threads of
5 a program configured for an SIMT execution model can be executed via a single SIMD instruction. For example, in at least one embodiment, eight SIMT threads that perform same or similar operations can be executed in parallel via a single SIMD8 logic unit.

[0526] In at least one embodiment, memory and cache interconnect 2268 is an interconnect network that connects each functional unit of graphics multiprocessor 2234 to register file
10 2258 and to shared memory 2270. In at least one embodiment, memory and cache interconnect 2268 is a crossbar interconnect that allows load/store unit 2266 to implement load and store operations between shared memory 2270 and register file 2258. In at least one embodiment, register file 2258 can operate at a same frequency as GPGPU cores 2262, thus data transfer between GPGPU cores 2262 and register file 2258 can have very low latency.

15 In at least one embodiment, shared memory 2270 can be used to enable communication between threads that execute on functional units within graphics multiprocessor 2234. In at least one embodiment, cache memory 2272 can be used as a data cache for example, to cache texture data communicated between functional units and texture unit 2236. In at least one embodiment, shared memory 2270 can also be used as a program managed cache. In at least
20 one embodiment, threads executing on GPGPU cores 2262 can programmatically store data within shared memory in addition to automatically cached data that is stored within cache memory 2272.

[0527] In at least one embodiment, a parallel processor or GPGPU as described herein is communicatively coupled to host/processor cores to accelerate graphics operations, machine-
25 learning operations, pattern analysis operations, and various general purpose GPU (GPGPU) functions. In at least one embodiment, a GPU may be communicatively coupled to host processor/cores over a bus or other interconnect (e.g., a high-speed interconnect such as PCIe or NVLink). In at least one embodiment, a GPU may be integrated on a package or chip as cores and communicatively coupled to cores over an internal processor bus/interconnect
30 internal to a package or chip. In at least one embodiment, regardless a manner in which a GPU is connected, processor cores may allocate work to such GPU in a form of sequences of commands/instructions contained in a work descriptor. In at least one embodiment, that GPU then uses dedicated circuitry/logic for efficiently processing these commands/instructions.

operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in graphics multiprocessor 2234 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0529] FIG. 23 illustrates a multi-GPU computing system 2300, according to at least one embodiment. In at least one embodiment, multi-GPU computing system 2300 can include a processor 2302 coupled to multiple general purpose graphics processing units (GPGPUs) 2306A-D via a host interface switch 2304. In at least one embodiment, host interface switch 2304 is a PCI express switch device that couples processor 2302 to a PCI express bus over which processor 2302 can communicate with GPGPUs 2306A-D. In at least one embodiment, GPGPUs 2306A-D can interconnect via a set of high-speed point-to-point GPU-to-GPU links 2316. In at least one embodiment, GPU-to-GPU links 2316 connect to each of GPGPUs 2306A-D via a dedicated GPU link. In at least one embodiment, P2P GPU links 2316 enable direct communication between each of GPGPUs 2306A-D without requiring communication over host interface bus 2304 to which processor 2302 is connected. In at least one embodiment, with GPU-to-GPU traffic directed to P2P GPU links 2316, host interface bus 2304 remains available for system memory access or to communicate with other instances of multi-GPU computing system 2300, for example, via one or more network devices. While in at least one embodiment GPGPUs 2306A-D connect to processor 2302 via host interface switch 2304, in at least one embodiment processor 2302 includes direct support for P2P GPU links 2316 and can connect directly to GPGPUs 2306A-D.

[0530] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in multi-GPU computing system 2300 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

embodiment. In at least one embodiment, graphics processor 2400 includes a ring interconnect 2402, a pipeline front-end 2404, a media engine 2437, and graphics cores 2480A-2480N. In at least one embodiment, ring interconnect 2402 couples graphics processor 2400 to other processing units, including other graphics processors or one or more general-purpose processor cores. In at least one embodiment, graphics processor 2400 is one of many processors integrated within a multi-core processing system.

[0532] In at least one embodiment, graphics processor 2400 receives batches of commands via ring interconnect 2402. In at least one embodiment, incoming commands are interpreted by a command streamer 2403 in pipeline front-end 2404. In at least one embodiment, graphics processor 2400 includes scalable execution logic to perform 3D geometry processing and media processing via graphics core(s) 2480A-2480N. In at least one embodiment, for 3D geometry processing commands, command streamer 2403 supplies commands to geometry pipeline 2436. In at least one embodiment, for at least some media processing commands, command streamer 2403 supplies commands to a video front end 2434, which couples with media engine 2437. In at least one embodiment, media engine 2437 includes a Video Quality Engine (VQE) 2430 for video and image post-processing and a multi-format encode/decode (MFX) 2433 engine to provide hardware-accelerated media data encoding and decoding. In at least one embodiment, geometry pipeline 2436 and media engine 2437 each generate execution threads for thread execution resources provided by at least one graphics core 2480.

[0533] In at least one embodiment, graphics processor 2400 includes scalable thread execution resources featuring graphics cores 2480A-2480N (which can be modular and are sometimes referred to as core slices), each having multiple sub-cores 2450A-50N, 2460A-2460N (sometimes referred to as core sub-slices). In at least one embodiment, graphics processor 2400 can have any number of graphics cores 2480A. In at least one embodiment, graphics processor 2400 includes a graphics core 2480A having at least a first sub-core 2450A and a second sub-core 2460A. In at least one embodiment, graphics processor 2400 is a low power processor with a single sub-core (e.g., 2450A). In at least one embodiment, graphics processor 2400 includes multiple graphics cores 2480A-2480N, each including a set of first sub-cores 2450A-2450N and a set of second sub-cores 2460A-2460N. In at least one embodiment, each sub-core in first sub-cores 2450A-2450N includes at least a first set of execution units 2452A-2452N and media/texture samplers 2454A-2454N. In at least one

of execution units 2462A-2462N and samplers 2464A-2464N. In at least one embodiment, each sub-core 2450A-2450N, 2460A-2460N shares a set of shared resources 2470A-2470N. In at least one embodiment, shared resources include shared cache memory and pixel
5 operation logic.

[0534] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in graphics processor 2400
10 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0535] FIG. 25 is a block diagram illustrating micro-architecture for a processor 2500 that may include logic circuits to perform instructions, according to at least one embodiment. In
15 at least one embodiment, processor 2500 may perform instructions, including x86 instructions, ARM instructions, specialized instructions for application-specific integrated circuits (ASICs), etc. In at least one embodiment, processor 2500 may include registers to store packed data, such as 64-bit wide MMX™ registers in microprocessors enabled with MMX technology from Intel Corporation of Santa Clara, Calif. In at least one embodiment,
20 MMX registers, available in both integer and floating point forms, may operate with packed data elements that accompany single instruction, multiple data (“SIMD”) and streaming SIMD extensions (“SSE”) instructions. In at least one embodiment, 128-bit wide XMM registers relating to SSE2, SSE3, SSE4, AVX, or beyond (referred to generically as “SSEx”) technology may hold such packed data operands. In at least one embodiment, processor 2500
25 may perform instructions to accelerate machine learning or deep learning algorithms, training, or inferencing.

[0536] In at least one embodiment, processor 2500 includes an in-order front end (“front end”) 2501 to fetch instructions to be executed and prepare instructions to be used later in a processor pipeline. In at least one embodiment, front end 2501 may include several units. In
30 at least one embodiment, an instruction prefetcher 2526 fetches instructions from memory and feeds instructions to an instruction decoder 2528 which in turn decodes or interprets instructions. For example, in at least one embodiment, instruction decoder 2528 decodes a

operations” (also called “micro ops” or “uops”) that a machine may execute. In at least one embodiment, instruction decoder 2528 parses an instruction into an opcode and corresponding data and control fields that may be used by micro-architecture to perform operations in accordance with at least one embodiment. In at least one embodiment, a trace cache 2530 may assemble decoded uops into program ordered sequences or traces in a uop queue 2534 for execution. In at least one embodiment, when trace cache 2530 encounters a complex instruction, a microcode ROM 2532 provides uops needed to complete an operation.

[0537] In at least one embodiment, some instructions may be converted into a single micro-op, whereas others need several micro-ops to complete full operation. In at least one embodiment, if more than four micro-ops are needed to complete an instruction, instruction decoder 2528 may access microcode ROM 2532 to perform that instruction. In at least one embodiment, an instruction may be decoded into a small number of micro-ops for processing at instruction decoder 2528. In at least one embodiment, an instruction may be stored within microcode ROM 2532 should a number of micro-ops be needed to accomplish such operation. In at least one embodiment, trace cache 2530 refers to an entry point programmable logic array (“PLA”) to determine a correct micro-instruction pointer for reading microcode sequences to complete one or more instructions from microcode ROM 2532 in accordance with at least one embodiment. In at least one embodiment, after microcode ROM 2532 finishes sequencing micro-ops for an instruction, front end 2501 of a machine may resume fetching micro-ops from trace cache 2530.

[0538] In at least one embodiment, out-of-order execution engine (“out of order engine”) 2503 may prepare instructions for execution. In at least one embodiment, out-of-order execution logic has a number of buffers to smooth out and re-order flow of instructions to optimize performance as they go down a pipeline and get scheduled for execution. In at least one embodiment, out-of-order execution engine 2503 includes, without limitation, an allocator/register renamer 2540, a memory uop queue 2542, an integer/floating point uop queue 2544, a memory scheduler 2546, a fast scheduler 2502, a slow/general floating point scheduler (“slow/general FP scheduler”) 2504, and a simple floating point scheduler (“simple FP scheduler”) 2506. In at least one embodiment, fast scheduler 2502, slow/general floating point scheduler 2504, and simple floating point scheduler 2506 are also collectively referred to herein as “uop schedulers 2502, 2504, 2506.” In at least one embodiment, allocator/register renamer 2540 allocates machine buffers and resources that each uop needs

logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer 2540 also allocates an entry for each uop in one of two uop queues, memory uop queue 2542 for memory operations and integer/floating point uop queue 2544 for non-memory operations, in front of memory scheduler 2546 and uop schedulers 2502, 2504, 2506. In at least one embodiment, uop schedulers 2502, 2504, 2506, determine when a uop is ready to execute based on readiness of their dependent input register operand sources and availability of execution resources uops need to complete their operation. In at least one embodiment, fast scheduler 2502 may schedule on each half of a main clock cycle while slow/general floating point scheduler 2504 and simple floating point scheduler 2506 may schedule once per main processor clock cycle. In at least one embodiment, uop schedulers 2502, 2504, 2506 arbitrate for dispatch ports to schedule uops for execution.

[0539] In at least one embodiment, execution block 2511 includes, without limitation, an integer register file/bypass network 2508, a floating point register file/bypass network (“FP register file/bypass network”) 2510, address generation units (“AGUs”) 2512 and 2514, fast Arithmetic Logic Units (ALUs) (“fast ALUs”) 2516 and 2518, a slow Arithmetic Logic Unit (“slow ALU”) 2520, a floating point ALU (“FP”) 2522, and a floating point move unit (“FP move”) 2524. In at least one embodiment, integer register file/bypass network 2508 and floating point register file/bypass network 2510 are also referred to herein as “register files 2508, 2510.” In at least one embodiment, AGUs 2512 and 2514, fast ALUs 2516 and 2518, slow ALU 2520, floating point ALU 2522, and floating point move unit 2524 are also referred to herein as “execution units 2512, 2514, 2516, 2518, 2520, 2522, and 2524.” In at least one embodiment, execution block 2511 may include, without limitation, any number (including zero) and type of register files, bypass networks, address generation units, and execution units, in any combination.

[0540] In at least one embodiment, register networks 2508, 2510 may be arranged between uop schedulers 2502, 2504, 2506, and execution units 2512, 2514, 2516, 2518, 2520, 2522, and 2524. In at least one embodiment, integer register file/bypass network 2508 performs integer operations. In at least one embodiment, floating point register file/bypass network 2510 performs floating point operations. In at least one embodiment, each of register networks 2508, 2510 may include, without limitation, a bypass network that may bypass or forward just completed results that have not yet been written into a register file to new dependent uops. In at least one embodiment, register networks 2508, 2510 may communicate

may include, without limitation, two separate register files, one register file for a low-order thirty-two bits of data and a second register file for a high order thirty-two bits of data. In at least one embodiment, floating point register file/bypass network 2510 may include, without
5 limitation, 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

[0541] In at least one embodiment, execution units 2512, 2514, 2516, 2518, 2520, 2522, 2524 may execute instructions. In at least one embodiment, register networks 2508, 2510 store integer and floating point data operand values that micro-instructions need to execute.

10 In at least one embodiment, processor 2500 may include, without limitation, any number and combination of execution units 2512, 2514, 2516, 2518, 2520, 2522, 2524. In at least one embodiment, floating point ALU 2522 and floating point move unit 2524, may execute floating point, MMX, SIMD, AVX and SSE, or other operations, including specialized machine learning instructions. In at least one embodiment, floating point ALU 2522 may
15 include, without limitation, a 64-bit by 64-bit floating point divider to execute divide, square root, and remainder micro ops. In at least one embodiment, instructions involving a floating point value may be handled with floating point hardware. In at least one embodiment, ALU operations may be passed to fast ALUs 2516, 2518. In at least one embodiment, fast ALUS 2516, 2518 may execute fast operations with an effective latency of half a clock cycle. In at
20 least one embodiment, most complex integer operations go to slow ALU 2520 as slow ALU 2520 may include, without limitation, integer execution hardware for long-latency type of operations, such as a multiplier, shifts, flag logic, and branch processing. In at least one embodiment, memory load/store operations may be executed by AGUs 2512, 2514. In at least one embodiment, fast ALU 2516, fast ALU 2518, and slow ALU 2520 may perform
25 integer operations on 64-bit data operands. In at least one embodiment, fast ALU 2516, fast ALU 2518, and slow ALU 2520 may be implemented to support a variety of data bit sizes including sixteen, thirty-two, 128, 256, etc. In at least one embodiment, floating point ALU 2522 and floating point move unit 2524 may be implemented to support a range of operands having bits of various widths, such as 128-bit wide packed data operands in conjunction with
30 SIMD and multimedia instructions.

[0542] In at least one embodiment, uop schedulers 2502, 2504, 2506 dispatch dependent operations before a parent load has finished executing. In at least one embodiment, as uops may be speculatively scheduled and executed in processor 2500, processor 2500 may also

data cache, there may be dependent operations in flight in a pipeline that have left a scheduler with temporarily incorrect data. In at least one embodiment, a replay mechanism tracks and re-executes instructions that use incorrect data. In at least one embodiment, dependent
5 operations might need to be replayed and independent ones may be allowed to complete. In at least one embodiment, schedulers and a replay mechanism of at least one embodiment of a processor may also be designed to catch instruction sequences for text string comparison operations.

[0543] In at least one embodiment, “registers” may refer to on-board processor storage
10 locations that may be used as part of instructions to identify operands. In at least one embodiment, registers may be those that may be usable from outside of a processor (from a programmer’s perspective). In at least one embodiment, registers might not be limited to a particular type of circuit. Rather, in at least one embodiment, a register may store data, provide data, and perform functions described herein. In at least one embodiment, registers
15 described herein may be implemented by circuitry within a processor using any number of different techniques, such as dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. In at least one embodiment, integer registers store 32-bit integer data. A register file of at least one embodiment also contains eight multimedia SIMD registers for
20 packed data.

[0544] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic 915 may be incorporated
25 into execution block 2511 and other memory or registers shown or not shown. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs illustrated in execution block 2511. Moreover, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of execution block 2511 to perform one or more machine learning algorithms, neural
30 network architectures, use cases, or training techniques described herein.

[0545] FIG. 26 illustrates a deep learning application processor 2600, according to at least one embodiment. In at least one embodiment, deep learning application processor 2600 uses

learning application processor 2600 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, deep learning application processor 2600 is an application-specific integrated circuit (ASIC). In at least one
5 embodiment, application processor 2600 performs matrix multiply operations either “hard-wired” into hardware as a result of performing one or more instructions or both. In at least one embodiment, deep learning application processor 2600 includes, without limitation, processing clusters 2610(1)-2610(12), Inter-Chip Links (“ICLs”) 2620(1)-2620(12), Inter-Chip
10 Controllers (“ICCs”) 2630(1)-2630(2), high-bandwidth memory second generation (“HBM2”) 2640(1)-2640(4), memory controllers (“Mem Ctrlrs”) 2642(1)-2642(4), high bandwidth memory physical layer (“HBM PHY”) 2644(1)-2644(4), a management-controller central processing unit (“management-controller CPU”) 2650, a Serial Peripheral Interface, Inter-Integrated Circuit, and General Purpose Input/Output block (“SPI, I²C, GPIO”) 2660, a peripheral component interconnect express controller and direct memory access block (“PCIe
15 Controller and DMA”) 2670, and a sixteen-lane peripheral component interconnect express port (“PCI Express x 16”) 2680.

[0546] In at least one embodiment, processing clusters 2610 may perform deep learning operations, including inference or prediction operations based on weight parameters calculated one or more training techniques, including those described herein. In at least one
20 embodiment, each processing cluster 2610 may include, without limitation, any number and type of processors. In at least one embodiment, deep learning application processor 2600 may include any number and type of processing clusters 2600. In at least one embodiment, Inter-Chip Links 2620 are bi-directional. In at least one embodiment, Inter-Chip Links 2620 and Inter-Chip Controllers 2630 enable multiple deep learning application processors 2600 to
25 exchange information, including activation information resulting from performing one or more machine learning algorithms embodied in one or more neural networks. In at least one embodiment, deep learning application processor 2600 may include any number (including zero) and type of ICLs 2620 and ICCs 2630.

[0547] In at least one embodiment, HBM2s 2640 provide a total of 32 Gigabytes (GB) of
30 memory. In at least one embodiment, HBM2 2640(i) is associated with both memory controller 2642(i) and HBM PHY 2644(i) where “i” is an arbitrary integer. In at least one embodiment, any number of HBM2s 2640 may provide any type and total amount of high bandwidth memory and may be associated with any number (including zero) and type of

2660, PCIe Controller and DMA 2670, and/or PCIe 2680 may be replaced with any number and type of blocks that enable any number and type of communication standards in any technically feasible fashion.

5 [0548] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to deep learning
10 application processor 2600. In at least one embodiment, deep learning application processor 2600 is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by deep learning application processor 2600. In at least one embodiment, processor 2600 may be used to perform one or more neural network use cases described herein.

15 [0549] FIG. 27 is a block diagram of a neuromorphic processor 2700, according to at least one embodiment. In at least one embodiment, neuromorphic processor 2700 may receive one or more inputs from sources external to neuromorphic processor 2700. In at least one embodiment, these inputs may be transmitted to one or more neurons 2702 within neuromorphic processor 2700. In at least one embodiment, neurons 2702 and components
20 thereof may be implemented using circuitry or logic, including one or more arithmetic logic units (ALUs). In at least one embodiment, neuromorphic processor 2700 may include, without limitation, thousands or millions of instances of neurons 2702, but any suitable number of neurons 2702 may be used. In at least one embodiment, each instance of neuron 2702 may include a neuron input 2704 and a neuron output 2706. In at least one
25 embodiment, neurons 2702 may generate outputs that may be transmitted to inputs of other instances of neurons 2702. For example, in at least one embodiment, neuron inputs 2704 and neuron outputs 2706 may be interconnected via synapses 2708.

[0550] In at least one embodiment, neurons 2702 and synapses 2708 may be interconnected such that neuromorphic processor 2700 operates to process or analyze information received
30 by neuromorphic processor 2700. In at least one embodiment, neurons 2702 may transmit an output pulse (or “fire” or “spike”) when inputs received through neuron input 2704 exceed a threshold. In at least one embodiment, neurons 2702 may sum or integrate signals received at

implemented as leaky integrate-and-fire neurons, wherein if a sum (referred to as a “membrane potential”) exceeds a threshold value, neuron 2702 may generate an output (or “fire”) using a transfer function such as a sigmoid or threshold function. In at least one embodiment, a leaky integrate-and-fire neuron may sum signals received at neuron inputs 2704 into a membrane potential and may also apply a decay factor (or leak) to reduce a membrane potential. In at least one embodiment, a leaky integrate-and-fire neuron may fire if multiple input signals are received at neuron inputs 2704 rapidly enough to exceed a threshold value (i.e., before a membrane potential decays too low to fire). In at least one embodiment, neurons 2702 may be implemented using circuits or logic that receive inputs, integrate inputs into a membrane potential, and decay a membrane potential. In at least one embodiment, inputs may be averaged, or any other suitable transfer function may be used. Furthermore, in at least one embodiment, neurons 2702 may include, without limitation, comparator circuits or logic that generate an output spike at neuron output 2706 when result of applying a transfer function to neuron input 2704 exceeds a threshold. In at least one embodiment, once neuron 2702 fires, it may disregard previously received input information by, for example, resetting a membrane potential to 0 or another suitable default value. In at least one embodiment, once membrane potential is reset to 0, neuron 2702 may resume normal operation after a suitable period of time (or refractory period).

[0551] In at least one embodiment, neurons 2702 may be interconnected through synapses 2708. In at least one embodiment, synapses 2708 may operate to transmit signals from an output of a first neuron 2702 to an input of a second neuron 2702. In at least one embodiment, neurons 2702 may transmit information over more than one instance of synapse 2708. In at least one embodiment, one or more instances of neuron output 2706 may be connected, via an instance of synapse 2708, to an instance of neuron input 2704 in same neuron 2702. In at least one embodiment, an instance of neuron 2702 generating an output to be transmitted over an instance of synapse 2708 may be referred to as a “pre-synaptic neuron” with respect to that instance of synapse 2708. In at least one embodiment, an instance of neuron 2702 receiving an input transmitted over an instance of synapse 2708 may be referred to as a “post-synaptic neuron” with respect to that instance of synapse 2708. Because an instance of neuron 2702 may receive inputs from one or more instances of synapse 2708, and may also transmit outputs over one or more instances of synapse 2708, a single instance of neuron 2702 may therefore be both a “pre-synaptic neuron” and “post-

embodiment.

[0552] In at least one embodiment, neurons 2702 may be organized into one or more layers. In at least one embodiment, each instance of neuron 2702 may have one neuron
5 output 2706 that may fan out through one or more synapses 2708 to one or more neuron inputs 2704. In at least one embodiment, neuron outputs 2706 of neurons 2702 in a first layer 2710 may be connected to neuron inputs 2704 of neurons 2702 in a second layer 2712. In at least one embodiment, layer 2710 may be referred to as a “feed-forward layer.” In at least one embodiment, each instance of neuron 2702 in an instance of first layer 2710 may fan out
10 to each instance of neuron 2702 in second layer 2712. In at least one embodiment, first layer 2710 may be referred to as a “fully connected feed-forward layer.” In at least one embodiment, each instance of neuron 2702 in an instance of second layer 2712 may fan out to fewer than all instances of neuron 2702 in a third layer 2714. In at least one embodiment, second layer 2712 may be referred to as a “sparsely connected feed-forward layer.” In at
15 least one embodiment, neurons 2702 in second layer 2712 may fan out to neurons 2702 in multiple other layers, including to neurons 2702 also in second layer 2712. In at least one embodiment, second layer 2712 may be referred to as a “recurrent layer.” In at least one embodiment, neuromorphic processor 2700 may include, without limitation, any suitable combination of recurrent layers and feed-forward layers, including, without limitation, both
20 sparsely connected feed-forward layers and fully connected feed-forward layers.

[0553] In at least one embodiment, neuromorphic processor 2700 may include, without limitation, a reconfigurable interconnect architecture or dedicated hard-wired interconnects to connect synapse 2708 to neurons 2702. In at least one embodiment, neuromorphic processor 2700 may include, without limitation, circuitry or logic that allows synapses to be allocated to
25 different neurons 2702 as needed based on neural network topology and neuron fan-in/out. For example, in at least one embodiment, synapses 2708 may be connected to neurons 2702 using an interconnect fabric, such as network-on-chip, or with dedicated connections. In at least one embodiment, synapse interconnections and components thereof may be implemented using circuitry or logic.

30 [0554] FIG. 28 is a block diagram of a processing system, according to at least one embodiment. In at least one embodiment, system 2800 includes one or more processors 2802 and one or more graphics processors 2808, and may be a single processor desktop system, a

2802 or processor cores 2807. In at least one embodiment, system 2800 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

- 5 [0555] In at least one embodiment, system 2800 can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming console, a handheld game console, or an online game console. In at least one embodiment, system 2800 is a mobile phone, a smart phone, a tablet computing device or a mobile Internet device. In at least one embodiment, processing system 2800 can also include,
- 10 couple with, or be integrated within a wearable device, such as a smart watch wearable device, a smart eyewear device, an augmented reality device, or a virtual reality device. In at least one embodiment, processing system 2800 is a television or set top box device having one or more processors 2802 and a graphical interface generated by one or more graphics processors 2808.
- 15 [0556] In at least one embodiment, one or more processors 2802 each include one or more processor cores 2807 to process instructions which, when executed, perform operations for system and user software. In at least one embodiment, each of one or more processor cores 2807 is configured to process a specific instruction sequence 2809. In at least one embodiment, instruction sequence 2809 may facilitate Complex Instruction Set Computing
- 20 (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). In at least one embodiment, processor cores 2807 may each process a different instruction sequence 2809, which may include instructions to facilitate emulation of other instruction sequences. In at least one embodiment, processor core 2807 may also include other processing devices, such a Digital Signal Processor (DSP).
- 25 [0557] In at least one embodiment, processor 2802 includes a cache memory 2804. In at least one embodiment, processor 2802 can have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory is shared among various components of processor 2802. In at least one embodiment, processor 2802 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which
- 30 may be shared among processor cores 2807 using known cache coherency techniques. In at least one embodiment, a register file 2806 is additionally included in processor 2802, which may include different types of registers for storing different types of data (e.g., integer

least one embodiment, register file 2806 may include general-purpose registers or other registers.

[0558] In at least one embodiment, one or more processor(s) 2802 are coupled with one or more interface bus(es) 2810 to transmit communication signals such as address, data, or control signals between processor 2802 and other components in system 2800. In at least one embodiment, interface bus 2810 can be a processor bus, such as a version of a Direct Media Interface (DMI) bus. In at least one embodiment, interface bus 2810 is not limited to a DMI bus, and may include one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express), memory busses, or other types of interface busses. In at least one embodiment processor(s) 2802 include an integrated memory controller 2816 and a platform controller hub 2830. In at least one embodiment, memory controller 2816 facilitates communication between a memory device and other components of system 2800, while platform controller hub (PCH) 2830 provides connections to I/O devices via a local I/O bus.

[0559] In at least one embodiment, a memory device 2820 can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as process memory. In at least one embodiment, memory device 2820 can operate as system memory for system 2800, to store data 2822 and instructions 2821 for use when one or more processors 2802 executes an application or process. In at least one embodiment, memory controller 2816 also couples with an optional external graphics processor 2812, which may communicate with one or more graphics processors 2808 in processors 2802 to perform graphics and media operations. In at least one embodiment, a display device 2811 can connect to processor(s) 2802. In at least one embodiment, display device 2811 can include one or more of an internal display device, as in a mobile electronic device or a laptop device, or an external display device attached via a display interface (e.g., DisplayPort, etc.). In at least one embodiment, display device 2811 can include a head mounted display (HMD) such as a stereoscopic display device for use in virtual reality (VR) applications or augmented reality (AR) applications.

[0560] In at least one embodiment, platform controller hub 2830 enables peripherals to connect to memory device 2820 and processor 2802 via a high-speed I/O bus. In at least one embodiment, I/O peripherals include, but are not limited to, an audio controller 2846, a

sensors 2825, a data storage device 2824 (e.g., hard disk drive, flash memory, etc.). In at least one embodiment, data storage device 2824 can connect via a storage interface (e.g., SATA) or via a peripheral bus, such as a Peripheral Component Interconnect bus (e.g., PCI, PCI Express). In at least one embodiment, touch sensors 2825 can include touch screen sensors, pressure sensors, or fingerprint sensors. In at least one embodiment, wireless transceiver 2826 can be a Wi-Fi transceiver, a Bluetooth transceiver, or a mobile network transceiver such as a 3G, 4G, or Long Term Evolution (LTE) transceiver. In at least one embodiment, firmware interface 2828 enables communication with system firmware, and can be, for example, a unified extensible firmware interface (UEFI). In at least one embodiment, network controller 2834 can enable a network connection to a wired network. In at least one embodiment, a high-performance network controller (not shown) couples with interface bus 2810. In at least one embodiment, audio controller 2846 is a multi-channel high definition audio controller. In at least one embodiment, system 2800 includes an optional legacy I/O controller 2840 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to system 2800. In at least one embodiment, platform controller hub 2830 can also connect to one or more Universal Serial Bus (USB) controllers 2842 connect input devices, such as keyboard and mouse 2843 combinations, a camera 2844, or other USB input devices.

[0561] In at least one embodiment, an instance of memory controller 2816 and platform controller hub 2830 may be integrated into a discreet external graphics processor, such as external graphics processor 2812. In at least one embodiment, platform controller hub 2830 and/or memory controller 2816 may be external to one or more processor(s) 2802. For example, in at least one embodiment, system 2800 can include an external memory controller 2816 and platform controller hub 2830, which may be configured as a memory controller hub and peripheral controller hub within a system chipset that is in communication with processor(s) 2802.

[0562] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic 915 may be incorporated into graphics processor 2808. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in a 3D pipeline. Moreover, in at least one embodiment, inferencing and/or training operations

least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 2808 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0563] FIG. 29 is a block diagram of a processor 2900 having one or more processor cores 2902A-2902N, an integrated memory controller 2914, and an integrated graphics processor 2908, according to at least one embodiment. In at least one embodiment, processor 2900 can include additional cores up to and including additional core 2902N represented by dashed lined boxes. In at least one embodiment, each of processor cores 2902A-2902N includes one or more internal cache units 2904A-2904N. In at least one embodiment, each processor core also has access to one or more shared cached units 2906.

[0564] In at least one embodiment, internal cache units 2904A-2904N and shared cache units 2906 represent a cache memory hierarchy within processor 2900. In at least one embodiment, cache memory units 2904A-2904N may include at least one level of instruction and data cache within each processor core and one or more levels of shared mid-level cache, such as a Level 2 (L2), Level 3 (L3), Level 4 (L4), or other levels of cache, where a highest level of cache before external memory is classified as an LLC. In at least one embodiment, cache coherency logic maintains coherency between various cache units 2906 and 2904A-2904N.

[0565] In at least one embodiment, processor 2900 may also include a set of one or more bus controller units 2916 and a system agent core 2910. In at least one embodiment, bus controller units 2916 manage a set of peripheral buses, such as one or more PCI or PCI express busses. In at least one embodiment, system agent core 2910 provides management functionality for various processor components. In at least one embodiment, system agent core 2910 includes one or more integrated memory controllers 2914 to manage access to various external memory devices (not shown).

[0566] In at least one embodiment, one or more of processor cores 2902A-2902N include support for simultaneous multi-threading. In at least one embodiment, system agent core 2910 includes components for coordinating and operating cores 2902A-2902N during multi-threaded processing. In at least one embodiment, system agent core 2910 may additionally

more power states of processor cores 2902A-2902N and graphics processor 2908.

[0567] In at least one embodiment, processor 2900 additionally includes graphics processor 2908 to execute graphics processing operations. In at least one embodiment, graphics processor 2908 couples with shared cache units 2906, and system agent core 2910, including one or more integrated memory controllers 2914. In at least one embodiment, system agent core 2910 also includes a display controller 2911 to drive graphics processor output to one or more coupled displays. In at least one embodiment, display controller 2911 may also be a separate module coupled with graphics processor 2908 via at least one interconnect, or may be integrated within graphics processor 2908.

[0568] In at least one embodiment, a ring-based interconnect unit 2912 is used to couple internal components of processor 2900. In at least one embodiment, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques. In at least one embodiment, graphics processor 2908 couples with ring interconnect 2912 via an I/O link 2913.

[0569] In at least one embodiment, I/O link 2913 represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded memory module 2918, such as an eDRAM module. In at least one embodiment, each of processor cores 2902A-2902N and graphics processor 2908 use embedded memory module 2918 as a shared Last Level Cache.

[0570] In at least one embodiment, processor cores 2902A-2902N are homogeneous cores executing a common instruction set architecture. In at least one embodiment, processor cores 2902A-2902N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 2902A-2902N execute a common instruction set, while one or more other cores of processor cores 2902A-2902N executes a subset of a common instruction set or a different instruction set. In at least one embodiment, processor cores 2902A-2902N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. In at least one embodiment, processor 2900 can be implemented on one or more chips or as an SoC integrated circuit.

operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic 915 may be incorporated into graphics processor 2908. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in a 3D pipeline, graphics core(s) 2902, shared function logic, or other logic in FIG. 29. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of processor 2900 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0572] FIG. 30 is a block diagram of a graphics processor 3000, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. In at least one embodiment, graphics processor 3000 communicates via a memory mapped I/O interface to registers on graphics processor 3000 and with commands placed into memory. In at least one embodiment, graphics processor 3000 includes a memory interface 3014 to access memory. In at least one embodiment, memory interface 3014 is an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0573] In at least one embodiment, graphics processor 3000 also includes a display controller 3002 to drive display output data to a display device 3020. In at least one embodiment, display controller 3002 includes hardware for one or more overlay planes for display device 3020 and composition of multiple layers of video or user interface elements. In at least one embodiment, display device 3020 can be an internal or external display device. In at least one embodiment, display device 3020 is a head mounted display device, such as a virtual reality (VR) display device or an augmented reality (AR) display device. In at least one embodiment, graphics processor 3000 includes a video codec engine 3006 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint

formats.

[0574] In at least one embodiment, graphics processor 3000 includes a block image transfer (BLIT) engine 3004 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in at least one embodiment, 2D graphics operations are performed using one or more components of a graphics processing engine (GPE) 3010. In at least one embodiment, GPE 3010 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0575] In at least one embodiment, GPE 3010 includes a 3D pipeline 3012 for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). In at least one embodiment, 3D pipeline 3012 includes programmable and fixed function elements that perform various tasks and/or spawn execution threads to a 3D/Media sub-system 3015.

While 3D pipeline 3012 can be used to perform media operations, in at least one embodiment, GPE 3010 also includes a media pipeline 3016 that is used to perform media operations, such as video post-processing and image enhancement.

[0576] In at least one embodiment, media pipeline 3016 includes fixed function or programmable logic units to perform one or more specialized media operations, such as video decode acceleration, video de-interlacing, and video encode acceleration in place of, or on behalf of, video codec engine 3006. In at least one embodiment, media pipeline 3016 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 3015. In at least one embodiment, spawned threads perform computations for media operations on one or more graphics execution units included in 3D/Media sub-system 3015.

[0577] In at least one embodiment, 3D/Media subsystem 3015 includes logic for executing threads spawned by 3D pipeline 3012 and media pipeline 3016. In at least one embodiment, 3D pipeline 3012 and media pipeline 3016 send thread execution requests to 3D/Media subsystem 3015, which includes thread dispatch logic for arbitrating and dispatching various requests to available thread execution resources. In at least one embodiment, execution resources include an array of graphics execution units to process 3D and media threads. In at least one embodiment, 3D/Media subsystem 3015 includes one or more internal caches for

shared memory, including registers and addressable memory, to share data between threads and to store output data.

[0578] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic 915 may be incorporated into graphics processor 3000. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in 3D pipeline 3012. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 3000 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0579] FIG. 31 is a block diagram of a graphics processing engine 3110 of a graphics processor in accordance with at least one embodiment. In at least one embodiment, graphics processing engine (GPE) 3110 is a version of GPE 3010 shown in FIG. 30. In at least one embodiment, a media pipeline 3116 is optional and may not be explicitly included within GPE 3110. In at least one embodiment, a separate media and/or image processor is coupled to GPE 3110.

[0580] In at least one embodiment, GPE 3110 is coupled to or includes a command streamer 3103, which provides a command stream to a 3D pipeline 3112 and/or media pipeline 3116. In at least one embodiment, command streamer 3103 is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In at least one embodiment, command streamer 3103 receives commands from memory and sends commands to 3D pipeline 3112 and/or media pipeline 3116. In at least one embodiment, commands are instructions, primitives, or micro-operations fetched from a ring buffer, which stores commands for 3D pipeline 3112 and media pipeline 3116. In at least one embodiment, a ring buffer can additionally include batch command buffers storing batches of multiple commands. In at least one embodiment, commands for 3D pipeline 3112 can also include references to data stored in memory, such as, but not limited to, vertex and

pipeline 3116. In at least one embodiment, 3D pipeline 3112 and media pipeline 3116 process commands and data by performing operations or by dispatching one or more execution threads to a graphics core array 3114. In at least one embodiment, graphics core array 3114 includes one or more blocks of graphics cores (e.g., graphics core(s) 3115A, graphics core(s) 3115B), each block including one or more graphics cores. In at least one embodiment, each graphics core includes a set of graphics execution resources that includes general-purpose and graphics specific execution logic to perform graphics and compute operations, as well as fixed function texture processing and/or machine learning and artificial intelligence acceleration logic, including inference and/or training logic 915 in FIG. 9A and FIG. 9B.

[0581] In at least one embodiment, 3D pipeline 3112 includes fixed function and programmable logic to process one or more shader programs, such as vertex shaders, geometry shaders, pixel shaders, fragment shaders, compute shaders, or other shader programs, by processing instructions and dispatching execution threads to graphics core array 3114. In at least one embodiment, graphics core array 3114 provides a unified block of execution resources for use in processing shader programs. In at least one embodiment, a multi-purpose execution logic (e.g., execution units) within graphics core(s) 3115A-3115B of graphic core array 3114 includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

[0582] In at least one embodiment, graphics core array 3114 also includes execution logic to perform media functions, such as video and/or image processing. In at least one embodiment, execution units additionally include general-purpose logic that is programmable to perform parallel general-purpose computational operations, in addition to graphics processing operations.

[0583] In at least one embodiment, output data generated by threads executing on graphics core array 3114 can output data to memory in a unified return buffer (URB) 3118. In at least one embodiment, URB 3118 can store data for multiple threads. In at least one embodiment, URB 3118 may be used to send data between different threads executing on graphics core array 3114. In at least one embodiment, URB 3118 may additionally be used for synchronization between threads on graphics core array 3114 and fixed function logic within shared function logic 3120.

core array 3114 includes a variable number of graphics cores, each having a variable number of execution units based on a target power and performance level of GPE 3110. In at least one embodiment, execution resources are dynamically scalable, such that execution resources
5 may be enabled or disabled as needed.

[0585] In at least one embodiment, graphics core array 3114 is coupled to shared function logic 3120 that includes multiple resources that are shared between graphics cores in graphics core array 3114. In at least one embodiment, shared functions performed by shared function logic 3120 are embodied in hardware logic units that provide specialized supplemental
10 functionality to graphics core array 3114. In at least one embodiment, shared function logic 3120 includes but is not limited to a sampler unit 3121, a math unit 3122, and inter-thread communication (ITC) logic 3123. In at least one embodiment, one or more cache(s) 3125 are included in, or coupled to, shared function logic 3120.

[0586] In at least one embodiment, a shared function is used if demand for a specialized
15 function is insufficient for inclusion within graphics core array 3114. In at least one embodiment, a single instantiation of a specialized function is used in shared function logic 3120 and shared among other execution resources within graphics core array 3114. In at least one embodiment, specific shared functions within shared function logic 3120 that are used extensively by graphics core array 3114 may be included within shared function logic 3126
20 within graphics core array 3114. In at least one embodiment, shared function logic 3126 within graphics core array 3114 can include some or all logic within shared function logic 3120. In at least one embodiment, all logic elements within shared function logic 3120 may be duplicated within shared function logic 3126 of graphics core array 3114. In at least one embodiment, shared function logic 3120 is excluded in favor of shared function logic 3126
25 within graphics core array 3114.

[0587] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic 915 may be incorporated
30 into graphics processor 3110. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in 3D pipeline 3112, graphics core(s) 3115, shared function logic 3126, shared function logic 3120,

operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 3110 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0588] FIG. 32 is a block diagram of hardware logic of a graphics processor core 3200, according to at least one embodiment described herein. In at least one embodiment, graphics processor core 3200 is included within a graphics core array. In at least one embodiment, graphics processor core 3200, sometimes referred to as a core slice, can be one or multiple graphics cores within a modular graphics processor. In at least one embodiment, graphics processor core 3200 is exemplary of one graphics core slice, and a graphics processor as described herein may include multiple graphics core slices based on target power and performance envelopes. In at least one embodiment, each graphics core 3200 can include a fixed function block 3230 coupled with multiple sub-cores 3201A-3201F, also referred to as sub-slices, that include modular blocks of general-purpose and fixed function logic.

[0589] In at least one embodiment, fixed function block 3230 includes a geometry and fixed function pipeline 3236 that can be shared by all sub-cores in graphics processor 3200, for example, in lower performance and/or lower power graphics processor implementations. In at least one embodiment, geometry and fixed function pipeline 3236 includes a 3D fixed function pipeline, a video front-end unit, a thread spawner and thread dispatcher, and a unified return buffer manager, which manages unified return buffers.

[0590] In at least one embodiment, fixed function block 3230 also includes a graphics SoC interface 3237, a graphics microcontroller 3238, and a media pipeline 3239. In at least one embodiment, graphics SoC interface 3237 provides an interface between graphics core 3200 and other processor cores within a system on a chip integrated circuit. In at least one embodiment, graphics microcontroller 3238 is a programmable sub-processor that is configurable to manage various functions of graphics processor 3200, including thread dispatch, scheduling, and pre-emption. In at least one embodiment, media pipeline 3239 includes logic to facilitate decoding, encoding, pre-processing, and/or post-processing of multimedia data, including image and video data. In at least one embodiment, media pipeline

cores 3201A-3201F.

[0591] In at least one embodiment, SoC interface 3237 enables graphics core 3200 to communicate with general-purpose application processor cores (e.g., CPUs) and/or other components within an SoC, including memory hierarchy elements such as a shared last level cache memory, system RAM, and/or embedded on-chip or on-package DRAM. In at least one embodiment, SoC interface 3237 can also enable communication with fixed function devices within an SoC, such as camera imaging pipelines, and enables use of and/or implements global memory atomics that may be shared between graphics core 3200 and CPUs within an SoC. In at least one embodiment, graphics SoC interface 3237 can also implement power management controls for graphics processor core 3200 and enable an interface between a clock domain of graphics processor core 3200 and other clock domains within an SoC. In at least one embodiment, SoC interface 3237 enables receipt of command buffers from a command streamer and global thread dispatcher that are configured to provide commands and instructions to each of one or more graphics cores within a graphics processor. In at least one embodiment, commands and instructions can be dispatched to media pipeline 3239, when media operations are to be performed, or a geometry and fixed function pipeline (e.g., geometry and fixed function pipeline 3236, and/or a geometry and fixed function pipeline 3214) when graphics processing operations are to be performed.

[0592] In at least one embodiment, graphics microcontroller 3238 can be configured to perform various scheduling and management tasks for graphics core 3200. In at least one embodiment, graphics microcontroller 3238 can perform graphics and/or compute workload scheduling on various graphics parallel engines within execution unit (EU) arrays 3202A-3202F, 3204A-3204F within sub-cores 3201A-3201F. In at least one embodiment, host software executing on a CPU core of an SoC including graphics core 3200 can submit workloads to one of multiple graphic processor paths, which invokes a scheduling operation on an appropriate graphics engine. In at least one embodiment, scheduling operations include determining which workload to run next, submitting a workload to a command streamer, pre-empting existing workloads running on an engine, monitoring progress of a workload, and notifying host software when a workload is complete. In at least one embodiment, graphics microcontroller 3238 can also facilitate low-power or idle states for graphics core 3200, providing graphics core 3200 with an ability to save and restore registers within graphics core

graphics driver software on a system.

[0593] In at least one embodiment, graphics core 3200 may have greater than or fewer than illustrated sub-cores 3201A-3201F, up to N modular sub-cores. For each set of N sub-cores, in at least one embodiment, graphics core 3200 can also include shared function logic 3210, shared and/or cache memory 3212, geometry/fixed function pipeline 3214, as well as additional fixed function logic 3216 to accelerate various graphics and compute processing operations. In at least one embodiment, shared function logic 3210 can include logic units (e.g., sampler, math, and/or inter-thread communication logic) that can be shared by each N sub-cores within graphics core 3200. In at least one embodiment, shared and/or cache memory 3212 can be a last-level cache for N sub-cores 3201A-3201F within graphics core 3200 and can also serve as shared memory that is accessible by multiple sub-cores. In at least one embodiment, geometry/fixed function pipeline 3214 can be included instead of geometry/fixed function pipeline 3236 within fixed function block 3230 and can include similar logic units.

[0594] In at least one embodiment, graphics core 3200 includes additional fixed function logic 3216 that can include various fixed function acceleration logic for use by graphics core 3200. In at least one embodiment, additional fixed function logic 3216 includes an additional geometry pipeline for use in position-only shading. In position-only shading, at least two geometry pipelines exist, whereas in a full geometry pipeline within geometry and fixed function pipelines 3214, 3236, and a cull pipeline, which is an additional geometry pipeline that may be included within additional fixed function logic 3216. In at least one embodiment, a cull pipeline is a trimmed down version of a full geometry pipeline. In at least one embodiment, a full pipeline and a cull pipeline can execute different instances of an application, each instance having a separate context. In at least one embodiment, position only shading can hide long cull runs of discarded triangles, enabling shading to be completed earlier in some instances. For example, in at least one embodiment, cull pipeline logic within additional fixed function logic 3216 can execute position shaders in parallel with a main application and generally generates critical results faster than a full pipeline, as a cull pipeline fetches and shades position attributes of vertices, without performing rasterization and rendering of pixels to a frame buffer. In at least one embodiment, a cull pipeline can use generated critical results to compute visibility information for all triangles without regard to whether those triangles are culled. In at least one embodiment, a full pipeline (which in this

culled triangles to shade only visible triangles that are finally passed to a rasterization phase.

[0595] In at least one embodiment, additional fixed function logic 3216 can also include machine-learning acceleration logic, such as fixed function matrix multiplication logic, for implementations including optimizations for machine learning training or inferencing.

[0596] In at least one embodiment, within each graphics sub-core 3201A-3201F includes a set of execution resources that may be used to perform graphics, media, and compute operations in response to requests by graphics pipeline, media pipeline, or shader programs.

In at least one embodiment, graphics sub-cores 3201A-3201F include multiple EU arrays 3202A-3202F, 3204A-3204F, thread dispatch and inter-thread communication (TD/IC) logic 3203A-3203F, a 3D (e.g., texture) sampler 3205A-3205F, a media sampler 3206A-3206F, a shader processor 3207A-3207F, and shared local memory (SLM) 3208A-3208F. In at least one embodiment, EU arrays 3202A-3202F, 3204A-3204F each include multiple execution units, which are general-purpose graphics processing units capable of performing floating-point and integer/fixed-point logic operations in service of a graphics, media, or compute operation, including graphics, media, or compute shader programs. In at least one embodiment, TD/IC logic 3203A-3203F performs local thread dispatch and thread control operations for execution units within a sub-core and facilitates communication between threads executing on execution units of a sub-core. In at least one embodiment, 3D samplers 3205A-3205F can read texture or other 3D graphics related data into memory. In at least one embodiment, 3D samplers can read texture data differently based on a configured sample state and texture format associated with a given texture. In at least one embodiment, media samplers 3206A-3206F can perform similar read operations based on a type and format associated with media data. In at least one embodiment, each graphics sub-core 3201A-3201F can alternately include a unified 3D and media sampler. In at least one embodiment, threads executing on execution units within each of sub-cores 3201A-3201F can make use of shared local memory 3208A-3208F within each sub-core, to enable threads executing within a thread group to execute using a common pool of on-chip memory.

[0597] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, portions or all of inference and/or training logic 915 may be incorporated

inferencing techniques described herein may use one or more of ALUs embodied in a 3D pipeline, graphics microcontroller 3238, geometry and fixed function pipeline 3214 and 3236, or other logic in FIG. 32. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 3200 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0598] FIGS. 33A-33B illustrate thread execution logic 3300 including an array of processing elements of a graphics processor core according to at least one embodiment. FIG. 33A illustrates at least one embodiment, in which thread execution logic 3300 is used. FIG. 33B illustrates exemplary internal details of a graphics execution unit 3308, according to at least one embodiment.

[0599] As illustrated in FIG. 33A, in at least one embodiment, thread execution logic 3300 includes a shader processor 3302, a thread dispatcher 3304, an instruction cache 3306, a scalable execution unit array including a plurality of execution units 3307A-3307N and 3308A-3308N, a sampler 3310, a data cache 3312, and a data port 3314. In at least one embodiment, a scalable execution unit array can dynamically scale by enabling or disabling one or more execution units (e.g., any of execution unit 3308A-N or 3307A-N) based on computational requirements of a workload, for example. In at least one embodiment, scalable execution units are interconnected via an interconnect fabric that links to each execution unit. In at least one embodiment, thread execution logic 3300 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 3306, data port 3314, sampler 3310, and execution units 3307 or 3308. In at least one embodiment, each execution unit (e.g., 3307A) is a stand-alone programmable general-purpose computational unit that is capable of executing multiple simultaneous hardware threads while processing multiple data elements in parallel for each thread. In at least one embodiment, array of execution units 3307 and/or 3308 is scalable to include any number individual execution units.

[0600] In at least one embodiment, execution units 3307 and/or 3308 are primarily used to execute shader programs. In at least one embodiment, shader processor 3302 can process

a thread dispatcher 3304. In at least one embodiment, thread dispatcher 3304 includes logic to arbitrate thread initiation requests from graphics and media pipelines and instantiate requested threads on one or more execution units in execution units 3307 and/or 3308. For example, in at least one embodiment, a geometry pipeline can dispatch vertex, tessellation, or geometry shaders to thread execution logic for processing. In at least one embodiment, thread dispatcher 3304 can also process runtime thread spawning requests from executing shader programs.

[0601] In at least one embodiment, execution units 3307 and/or 3308 support an instruction set that includes native support for many standard 3D graphics shader instructions, such that shader programs from graphics libraries (e.g., Direct 3D and OpenGL) are executed with a minimal translation. In at least one embodiment, execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, and/or vertex shaders), pixel processing (e.g., pixel shaders, fragment shaders) and general-purpose processing (e.g., compute and media shaders). In at least one embodiment, each of execution units 3307 and/or 3308, which include one or more arithmetic logic units (ALUs), is capable of multi-issue single instruction multiple data (SIMD) execution and multi-threaded operation enables an efficient execution environment despite higher latency memory accesses. In at least one embodiment, each hardware thread within each execution unit has a dedicated high-bandwidth register file and associated independent thread-state. In at least one embodiment, execution is multi-issue per clock to pipelines capable of integer, single and double precision floating point operations, SIMD branch capability, logical operations, transcendental operations, and other miscellaneous operations. In at least one embodiment, while waiting for data from memory or one of shared functions, dependency logic within execution units 3307 and/or 3308 causes a waiting thread to sleep until requested data has been returned. In at least one embodiment, while an awaiting thread is sleeping, hardware resources may be devoted to processing other threads. For example, in at least one embodiment, during a delay associated with a vertex shader operation, an execution unit can perform operations for a pixel shader, fragment shader, or another type of shader program, including a different vertex shader.

[0602] In at least one embodiment, each execution unit in execution units 3307 and/or 3308 operates on arrays of data elements. In at least one embodiment, a number of data elements is an “execution size,” or number of channels for an instruction. In at least one embodiment,

control within instructions. In at least one embodiment, a number of channels may be independent of a number of physical arithmetic logic units (ALUs) or floating point units (FPUs) for a particular graphics processor. In at least one embodiment, execution units 3307
5 and/or 3308 support integer and floating-point data types.

[0603] In at least one embodiment, an execution unit instruction set includes SIMD instructions. In at least one embodiment, various data elements can be stored as a packed data type in a register and execution unit will process various elements based on data size of elements. For example, in at least one embodiment, when operating on a 256-bit wide vector,
10 256 bits of a vector are stored in a register and an execution unit operates on a vector as four separate 64-bit packed data elements (Quad-Word (QW) size data elements), eight separate 32-bit packed data elements (Double Word (DW) size data elements), sixteen separate 16-bit packed data elements (Word (W) size data elements), or thirty-two separate 8-bit data elements (byte (B) size data elements). However, in at least one embodiment, different
15 vector widths and register sizes are possible.

[0604] In at least one embodiment, one or more execution units can be combined into a fused execution unit 3309A-3309N having thread control logic (3311A-3311N) that is common to fused EUs such as execution unit 3307A fused with execution unit 3308A into fused execution unit 3309A. In at least one embodiment, multiple EUs can be fused into an
20 EU group. In at least one embodiment, each EU in a fused EU group can be configured to execute a separate SIMD hardware thread, with a number of EUs in a fused EU group possibly varying according to various embodiments. In at least one embodiment, various SIMD widths can be performed per-EU, including but not limited to SIMD8, SIMD16, and SIMD32. In at least one embodiment, each fused graphics execution unit 3309A-3309N
25 includes at least two execution units. For example, in at least one embodiment, fused execution unit 3309A includes a first EU 3307A, second EU 3308A, and thread control logic 3311A that is common to first EU 3307A and second EU 3308A. In at least one embodiment, thread control logic 3311A controls threads executed on fused graphics execution unit 3309A, allowing each EU within fused execution units 3309A-3309N to
30 execute using a common instruction pointer register.

[0605] In at least one embodiment, one or more internal instruction caches (e.g., 3306) are included in thread execution logic 3300 to cache thread instructions for execution units. In at

during thread execution. In at least one embodiment, sampler 3310 is included to provide texture sampling for 3D operations and media sampling for media operations. In at least one embodiment, sampler 3310 includes specialized texture or media sampling functionality to process texture or media data during sampling process before providing sampled data to an execution unit.

[0606] During execution, in at least one embodiment, graphics and media pipelines send thread initiation requests to thread execution logic 3300 via thread spawning and dispatch logic. In at least one embodiment, once a group of geometric objects has been processed and rasterized into pixel data, pixel processor logic (e.g., pixel shader logic, fragment shader logic, etc.) within shader processor 3302 is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In at least one embodiment, a pixel shader or a fragment shader calculates values of various vertex attributes that are to be interpolated across a rasterized object. In at least one embodiment, pixel processor logic within shader processor 3302 then executes an application programming interface (API)-supplied pixel or fragment shader program. In at least one embodiment, to execute a shader program, shader processor 3302 dispatches threads to an execution unit (e.g., 3308A) via thread dispatcher 3304. In at least one embodiment, shader processor 3302 uses texture sampling logic in sampler 3310 to access texture data in texture maps stored in memory. In at least one embodiment, arithmetic operations on texture data and input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

[0607] In at least one embodiment, data port 3314 provides a memory access mechanism for thread execution logic 3300 to output processed data to memory for further processing on a graphics processor output pipeline. In at least one embodiment, data port 3314 includes or couples to one or more cache memories (e.g., data cache 3312) to cache data for memory access via a data port.

[0608] As illustrated in FIG. 33B, in at least one embodiment, a graphics execution unit 3308 can include an instruction fetch unit 3337, a general register file array (GRF) 3324, an architectural register file array (ARF) 3326, a thread arbiter 3322, a send unit 3330, a branch unit 3332, a set of SIMD floating point units (FPUs) 3334, and a set of dedicated integer SIMD ALUs 3335. In at least one embodiment, GRF 3324 and ARF 3326 includes a set of

hardware thread that may be active in graphics execution unit 3308. In at least one embodiment, per thread architectural state is maintained in ARF 3326, while data used during thread execution is stored in GRF 3324. In at least one embodiment, execution state of each thread, including instruction pointers for each thread, can be held in thread-specific registers in ARF 3326.

[0609] In at least one embodiment, graphics execution unit 3308 has an architecture that is a combination of Simultaneous Multi-Threading (SMT) and fine-grained Interleaved Multi-Threading (IMT). In at least one embodiment, architecture has a modular configuration that can be fine-tuned at design time based on a target number of simultaneous threads and number of registers per execution unit, where execution unit resources are divided across logic used to execute multiple simultaneous threads.

[0610] In at least one embodiment, graphics execution unit 3308 can co-issue multiple instructions, which may each be different instructions. In at least one embodiment, thread arbiter 3322 of graphics execution unit thread 3308 can dispatch instructions to one of send unit 3330, branch unit 3332, or SIMD FPU(s) 3334 for execution. In at least one embodiment, each execution thread can access 128 general-purpose registers within GRF 3324, where each register can store 32 bytes, accessible as a SIMD 8-element vector of 32-bit data elements. In at least one embodiment, each execution unit thread has access to 4 kilobytes within GRF 3324, although embodiments are not so limited, and greater or fewer register resources may be provided in other embodiments. In at least one embodiment, up to seven threads can execute simultaneously, although a number of threads per execution unit can also vary according to embodiments. In at least one embodiment, in which seven threads may access 4 kilobytes, GRF 3324 can store a total of 28 kilobytes. In at least one embodiment, flexible addressing modes can permit registers to be addressed together to build effectively wider registers or to represent strided rectangular block data structures.

[0611] In at least one embodiment, memory operations, sampler operations, and other longer-latency system communications are dispatched via “send” instructions that are executed by message passing to send unit 3330. In at least one embodiment, branch instructions are dispatched to branch unit 3332 to facilitate SIMD divergence and eventual convergence.

SIMD floating point units (FPU(s)) 3334 to perform floating-point operations. In at least one embodiment, FPU(s) 3334 also support integer computation. In at least one embodiment, FPU(s) 3334 can SIMD execute up to M number of 32-bit floating-point (or integer)

5 operations, or SIMD execute up to 2M 16-bit integer or 16-bit floating-point operations. In at least one embodiment, at least one FPU provides extended math capability to support high-throughput transcendental math functions and double precision 64-bit floating-point. In at least one embodiment, a set of 8-bit integer SIMD ALUs 3335 are also present, and may be specifically optimized to perform operations associated with machine learning computations.

10 [0613] In at least one embodiment, arrays of multiple instances of graphics execution unit 3308 can be instantiated in a graphics sub-core grouping (e.g., a sub-slice). In at least one embodiment, execution unit 3308 can execute instructions across a plurality of execution channels. In at least one embodiment, each thread executed on graphics execution unit 3308 is executed on a different channel.

15 [0614] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, portions or all of inference and/or training logic 915 may be incorporated into thread execution logic 3300. Moreover, in at least one embodiment, inferencing and/or
20 training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs thread of execution logic 3300 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

25 [0615] FIG. 34 illustrates a parallel processing unit (“PPU”) 3400, according to at least one embodiment. In at least one embodiment, PPU 3400 is configured with machine-readable code that, if executed by PPU 3400, causes PPU 3400 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, PPU 3400 is a multi-threaded processor that is implemented on one or more integrated circuit devices
30 and that utilizes multithreading as a latency-hiding technique designed to process computer-readable instructions (also referred to as machine-readable instructions or simply instructions) on multiple threads in parallel. In at least one embodiment, a thread refers to a thread of

3400. In at least one embodiment, PPU 3400 is a graphics processing unit (“GPU”) configured to implement a graphics rendering pipeline for processing three-dimensional (“3D”) graphics data in order to generate two-dimensional (“2D”) image data for display on a display device such as a liquid crystal display (“LCD”) device. In at least one embodiment, PPU 3400 is utilized to perform computations such as linear algebra operations and machine-learning operations. FIG. 34 illustrates an example parallel processor for illustrative purposes only and should be construed as a non-limiting example of processor architectures contemplated within scope of this disclosure and that any suitable processor may be employed to supplement and/or substitute for same.

[0616] In at least one embodiment, one or more PPUs 3400 are configured to accelerate High Performance Computing (“HPC”), data center, and machine learning applications. In at least one embodiment, PPU 3400 is configured to accelerate deep learning systems and applications including following non-limiting examples: autonomous vehicle platforms, deep learning, high-accuracy speech, image, text recognition systems, intelligent video analytics, molecular simulations, drug discovery, disease diagnosis, weather forecasting, big data analytics, astronomy, molecular dynamics simulation, financial modeling, robotics, factory automation, real-time language translation, online search optimizations, and personalized user recommendations, and more.

[0617] In at least one embodiment, PPU 3400 includes, without limitation, an Input/Output (“I/O”) unit 3406, a front-end unit 3410, a scheduler unit 3412, a work distribution unit 3414, a hub 3416, a crossbar (“XBar”) 3420, one or more general processing clusters (“GPCs”) 3418, and one or more partition units (“memory partition units”) 3422. In at least one embodiment, PPU 3400 is connected to a host processor or other PPUs 3400 via one or more high-speed GPU interconnects (“GPU interconnects”) 3408. In at least one embodiment, PPU 3400 is connected to a host processor or other peripheral devices via a system bus 3402. In at least one embodiment, PPU 3400 is connected to a local memory comprising one or more memory devices (“memory”) 3404. In at least one embodiment, memory devices 3404 include, without limitation, one or more dynamic random access memory (“DRAM”) devices. In at least one embodiment, one or more DRAM devices are configured and/or configurable as high-bandwidth memory (“HBM”) subsystems, with multiple DRAM dies stacked within each device.

based multi-lane communications link that is used by systems to scale and include one or more PPUs 3400 combined with one or more central processing units (“CPUs”), supports cache coherence between PPUs 3400 and CPUs, and CPU mastering. In at least one
5 embodiment, data and/or commands are transmitted by high-speed GPU interconnect 3408 through hub 3416 to/from other units of PPU 3400 such as one or more copy engines, video encoders, video decoders, power management units, and other components which may not be explicitly illustrated in FIG. 34.

[0619] In at least one embodiment, I/O unit 3406 is configured to transmit and receive
10 communications (e.g., commands, data) from a host processor (not illustrated in FIG. 34) over system bus 3402. In at least one embodiment, I/O unit 3406 communicates with host processor directly via system bus 3402 or through one or more intermediate devices such as a memory bridge. In at least one embodiment, I/O unit 3406 may communicate with one or more other processors, such as one or more of PPUs 3400 via system bus 3402. In at least
15 one embodiment, I/O unit 3406 implements a Peripheral Component Interconnect Express (“PCIe”) interface for communications over a PCIe bus. In at least one embodiment, I/O unit 3406 implements interfaces for communicating with external devices.

[0620] In at least one embodiment, I/O unit 3406 decodes packets received via system bus 3402. In at least one embodiment, at least some packets represent commands configured to
20 cause PPU 3400 to perform various operations. In at least one embodiment, I/O unit 3406 transmits decoded commands to various other units of PPU 3400 as specified by commands. In at least one embodiment, commands are transmitted to front-end unit 3410 and/or transmitted to hub 3416 or other units of PPU 3400 such as one or more copy engines, a video encoder, a video decoder, a power management unit, etc. (not explicitly illustrated in
25 FIG. 34). In at least one embodiment, I/O unit 3406 is configured to route communications between and among various logical units of PPU 3400.

[0621] In at least one embodiment, a program executed by host processor encodes a command stream in a buffer that provides workloads to PPU 3400 for processing. In at least one embodiment, a workload comprises instructions and data to be processed by those
30 instructions. In at least one embodiment, a buffer is a region in a memory that is accessible (e.g., read/write) by both a host processor and PPU 3400 — a host interface unit may be configured to access that buffer in a system memory connected to system bus 3402 via

embodiment, a host processor writes a command stream to a buffer and then transmits a pointer to a start of a command stream to PPU 3400 such that front-end unit 3410 receives pointers to one or more command streams and manages one or more command streams,
5 reading commands from command streams and forwarding commands to various units of PPU 3400.

[0622] In at least one embodiment, front-end unit 3410 is coupled to scheduler unit 3412 that configures various GPCs 3418 to process tasks defined by one or more command streams. In at least one embodiment, scheduler unit 3412 is configured to track state
10 information related to various tasks managed by scheduler unit 3412 where state information may indicate which of GPCs 3418 a task is assigned to, whether task is active or inactive, a priority level associated with task, and so forth. In at least one embodiment, scheduler unit 3412 manages execution of a plurality of tasks on one or more of GPCs 3418.

[0623] In at least one embodiment, scheduler unit 3412 is coupled to work distribution unit
15 3414 that is configured to dispatch tasks for execution on GPCs 3418. In at least one embodiment, work distribution unit 3414 tracks a number of scheduled tasks received from scheduler unit 3412 and work distribution unit 3414 manages a pending task pool and an active task pool for each of GPCs 3418. In at least one embodiment, pending task pool comprises a number of slots (e.g., 32 slots) that contain tasks assigned to be processed by a
20 particular GPC 3418; an active task pool may comprise a number of slots (e.g., 4 slots) for tasks that are actively being processed by GPCs 3418 such that as one of GPCs 3418 completes execution of a task, that task is evicted from that active task pool for GPC 3418 and another task from a pending task pool is selected and scheduled for execution on GPC 3418. In at least one embodiment, if an active task is idle on GPC 3418, such as while
25 waiting for a data dependency to be resolved, then that active task is evicted from GPC 3418 and returned to that pending task pool while another task in that pending task pool is selected and scheduled for execution on GPC 3418.

[0624] In at least one embodiment, work distribution unit 3414 communicates with one or more GPCs 3418 via XBar 3420. In at least one embodiment, XBar 3420 is an interconnect
30 network that couples many of units of PPU 3400 to other units of PPU 3400 and can be configured to couple work distribution unit 3414 to a particular GPC 3418. In at least one

hub 3416.

[0625] In at least one embodiment, tasks are managed by scheduler unit 3412 and dispatched to one of GPCs 3418 by work distribution unit 3414. In at least one embodiment, GPC 3418 is configured to process task and generate results. In at least one embodiment, results may be consumed by other tasks within GPC 3418, routed to a different GPC 3418 via XBar 3420, or stored in memory 3404. In at least one embodiment, results can be written to memory 3404 via partition units 3422, which implement a memory interface for reading and writing data to/from memory 3404. In at least one embodiment, results can be transmitted to another PPU or CPU via high-speed GPU interconnect 3408. In at least one embodiment, PPU 3400 includes, without limitation, a number U of partition units 3422 that is equal to a number of separate and distinct memory devices 3404 coupled to PPU 3400, as described in more detail herein in conjunction with FIG. 36.

[0626] In at least one embodiment, a host processor executes a driver kernel that implements an application programming interface (“API”) that enables one or more applications executing on a host processor to schedule operations for execution on PPU 3400. In at least one embodiment, multiple compute applications are simultaneously executed by PPU 3400 and PPU 3400 provides isolation, quality of service (“QoS”), and independent address spaces for multiple compute applications. In at least one embodiment, an application generates instructions (e.g., in form of API calls) that cause a driver kernel to generate one or more tasks for execution by PPU 3400 and that driver kernel outputs tasks to one or more streams being processed by PPU 3400. In at least one embodiment, each task comprises one or more groups of related threads, which may be referred to as a warp. In at least one embodiment, a warp comprises a plurality of related threads (e.g., 32 threads) that can be executed in parallel. In at least one embodiment, cooperating threads can refer to a plurality of threads including instructions to perform task and that exchange data through shared memory. In at least one embodiment, threads and cooperating threads are described in more detail in conjunction with FIG. 36.

[0627] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning

least one embodiment, deep learning application processor is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by PPU 3400. In at least one embodiment, PPU
5 3400 may be used to perform one or more neural network use cases described herein.

[0628] FIG. 35 illustrates a general processing cluster (“GPC”) 3500, according to at least one embodiment. In at least one embodiment, GPC 3500 is GPC 3418 of FIG. 34. In at least one embodiment, each GPC 3500 includes, without limitation, a number of hardware units for processing tasks and each GPC 3500 includes, without limitation, a pipeline manager
10 3502, a pre-raster operations unit (“preROP”) 3504, a raster engine 3508, a work distribution crossbar (“WDX”) 3516, a memory management unit (“MMU”) 3518, one or more Data Processing Clusters (“DPCs”) 3506, and any suitable combination of parts.

[0629] In at least one embodiment, operation of GPC 3500 is controlled by pipeline manager 3502. In at least one embodiment, pipeline manager 3502 manages configuration of
15 one or more DPCs 3506 for processing tasks allocated to GPC 3500. In at least one embodiment, pipeline manager 3502 configures at least one of one or more DPCs 3506 to implement at least a portion of a graphics rendering pipeline. In at least one embodiment, DPC 3506 is configured to execute a vertex shader program on a programmable streaming multi-processor (“SM”) 3514. In at least one embodiment, pipeline manager 3502 is
20 configured to route packets received from a work distribution unit to appropriate logical units within GPC 3500, in at least one embodiment, and some packets may be routed to fixed function hardware units in preROP 3504 and/or raster engine 3508 while other packets may be routed to DPCs 3506 for processing by a primitive engine 3512 or SM 3514. In at least one embodiment, pipeline manager 3502 configures at least one of DPCs 3506 to implement
25 a neural network model and/or a computing pipeline.

[0630] In at least one embodiment, preROP unit 3504 is configured, in at least one embodiment, to route data generated by raster engine 3508 and DPCs 3506 to a Raster Operations (“ROP”) unit in partition unit 3422, described in more detail above in conjunction with FIG. 34. In at least one embodiment, preROP unit 3504 is configured to perform
30 optimizations for color blending, organize pixel data, perform address translations, and more. In at least one embodiment, raster engine 3508 includes, without limitation, a number of fixed function hardware units configured to perform various raster operations, in at least one

raster engine, a culling engine, a clipping engine, a fine raster engine, a tile coalescing engine, and any suitable combination thereof. In at least one embodiment, setup engine receives transformed vertices and generates plane equations associated with geometric primitive defined by vertices; plane equations are transmitted to a coarse raster engine to generate coverage information (e.g., an x, y coverage mask for a tile) for primitive; output of a coarse raster engine is transmitted to a culling engine where fragments associated with a primitive that fail a z-test are culled, and transmitted to a clipping engine where fragments lying outside a viewing frustum are clipped. In at least one embodiment, fragments that survive clipping and culling are passed to a fine raster engine to generate attributes for pixel fragments based on plane equations generated by a setup engine. In at least one embodiment, an output of raster engine 3508 comprises fragments to be processed by any suitable entity, such as by a fragment shader implemented within DPC 3506.

[0631] In at least one embodiment, each DPC 3506 included in GPC 3500 comprises, without limitation, an M-Pipe Controller (“MPC”) 3510; primitive engine 3512; one or more SMs 3514; and any suitable combination thereof. In at least one embodiment, MPC 3510 controls operation of DPC 3506, routing packets received from pipeline manager 3502 to appropriate units in DPC 3506. In at least one embodiment, packets associated with a vertex are routed to primitive engine 3512, which is configured to fetch vertex attributes associated with a vertex from memory; in contrast, packets associated with a shader program may be transmitted to SM 3514.

[0632] In at least one embodiment, SM 3514 comprises, without limitation, a programmable streaming processor that is configured to process tasks represented by a number of threads. In at least one embodiment, SM 3514 is multi-threaded and configured to execute a plurality of threads (e.g., 32 threads) from a particular group of threads concurrently and implements a Single-Instruction, Multiple-Data (“SIMD”) architecture where each thread in a group of threads (e.g., a warp) is configured to process a different set of data based on same set of instructions. In at least one embodiment, all threads in group of threads execute a common set of instructions. In at least one embodiment, SM 3514 implements a Single-Instruction, Multiple Thread (“SIMT”) architecture wherein each thread in a group of threads is configured to process a different set of data based on that common set of instructions, but where individual threads in a group of threads are allowed to diverge during execution. In at least one embodiment, a program counter, call stack, and execution

within warps when threads within a warp diverge. In another embodiment, a program counter, call stack, and execution state is maintained for each individual thread, enabling equal concurrency between all threads, within and between warps. In at least one
5 embodiment, execution state is maintained for each individual thread and threads executing common instructions may be converged and executed in parallel for better efficiency. At least one embodiment of SM 3514 is described in more detail herein.

[0633] In at least one embodiment, MMU 3518 provides an interface between GPC 3500 and a memory partition unit (e.g., partition unit 3422 of FIG. 34) and MMU 3518 provides
10 translation of virtual addresses into physical addresses, memory protection, and arbitration of memory requests. In at least one embodiment, MMU 3518 provides one or more translation lookaside buffers (“TLBs”) for performing translation of virtual addresses into physical addresses in memory.

[0634] Inference and/or training logic 915 are used to perform inferencing and/or training
15 operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to GPC 3500. In at least one embodiment, GPC 3500 is used to infer or predict information based on a trained
20 machine learning model (e.g., neural network) that has been trained by another processor or system or by GPC 3500. In at least one embodiment, GPC 3500 may be used to perform one or more neural network use cases described herein.

[0635] FIG. 36 illustrates a memory partition unit 3600 of a parallel processing unit (“PPU”), in accordance with at least one embodiment. In at least one embodiment, memory
25 partition unit 3600 includes, without limitation, a Raster Operations (“ROP”) unit 3602, a level two (“L2”) cache 3604, a memory interface 3606, and any suitable combination thereof. In at least one embodiment, memory interface 3606 is coupled to memory. In at least one embodiment, memory interface 3606 may implement 32, 64, 128, 1024-bit data buses, or like, for high-speed data transfer. In at least one embodiment, PPU incorporates U memory
30 interfaces 3606 where U is a positive integer, with one memory interface 3606 per pair of partition units 3600, where each pair of partition units 3600 is connected to a corresponding memory device. For example, in at least one embodiment, PPU may be connected to up to Y

version 5, synchronous dynamic random access memory (“GDDR5 SDRAM”).

[0636] In at least one embodiment, memory interface 3606 implements a high bandwidth memory second generation (“HBM2”) memory interface and Y equals half of U . In at least one embodiment, HBM2 memory stacks are located on a physical package with a PPU, providing substantial power and area savings compared with conventional GDDR5 SDRAM systems. In at least one embodiment, each HBM2 stack includes, without limitation, four memory dies with $Y=4$, with each HBM2 stack including two 128-bit channels per die for a total of 8 channels and a data bus width of 1024 bits. In at least one embodiment, that memory supports Single-Error Correcting Double-Error Detecting (“SECCDED”) Error Correction Code (“ECC”) to protect data. In at least one embodiment, ECC can provide higher reliability for compute applications that are sensitive to data corruption.

[0637] In at least one embodiment, PPU implements a multi-level memory hierarchy. In at least one embodiment, memory partition unit 3600 supports a unified memory to provide a single unified virtual address space for central processing unit (“CPU”) and PPU memory, enabling data sharing between virtual memory systems. In at least one embodiment frequency of accesses by a PPU to a memory located on other processors is traced to ensure that memory pages are moved to physical memory of PPU that is accessing pages more frequently. In at least one embodiment, high-speed GPU interconnect 3408 supports address translation services allowing PPU to directly access a CPU’s page tables and providing full access to CPU memory by a PPU.

[0638] In at least one embodiment, copy engines transfer data between multiple PPUs or between PPUs and CPUs. In at least one embodiment, copy engines can generate page faults for addresses that are not mapped into page tables and memory partition unit 3600 then services page faults, mapping addresses into page table, after which copy engine performs a transfer. In at least one embodiment, memory is pinned (i.e., non-pageable) for multiple copy engine operations between multiple processors, substantially reducing available memory. In at least one embodiment, with hardware page faulting, addresses can be passed to copy engines without regard as to whether memory pages are resident, and a copy process is transparent.

[0639] Data from memory 3404 of FIG. 34 or other system memory is fetched by memory partition unit 3600 and stored in L2 cache 3604, which is located on-chip and is shared

unit 3600, in at least one embodiment, includes, without limitation, at least a portion of L2 cache associated with a corresponding memory device. In at least one embodiment, lower level caches are implemented in various units within GPCs. In at least one embodiment, each of SMs 3514 in FIG. 35 may implement a Level 1 (“L1”) cache wherein that L1 cache is private memory that is dedicated to a particular SM 3514 and data from L2 cache 3604 is fetched and stored in each L1 cache for processing in functional units of SMs 3514. In at least one embodiment, L2 cache 3604 is coupled to memory interface 3606 and XBar 3420 shown in FIG. 34.

[0640] ROP unit 3602 performs graphics raster operations related to pixel color, such as color compression, pixel blending, and more, in at least one embodiment. ROP unit 3602, in at least one embodiment, implements depth testing in conjunction with raster engine 3508, receiving a depth for a sample location associated with a pixel fragment from a culling engine of raster engine 3508. In at least one embodiment, depth is tested against a corresponding depth in a depth buffer for a sample location associated with a fragment. In at least one embodiment, if that fragment passes that depth test for that sample location, then ROP unit 3602 updates depth buffer and transmits a result of that depth test to raster engine 3508. It will be appreciated that a number of partition units 3600 may be different than a number of GPCs and, therefore, each ROP unit 3602 can, in at least one embodiment, be coupled to each GPC. In at least one embodiment, ROP unit 3602 tracks packets received from different GPCs and determines whether a result generated by ROP unit 3602 is to be routed to through XBar 3420.

[0641] FIG. 37 illustrates a streaming multi-processor (“SM”) 3700, according to at least one embodiment. In at least one embodiment, SM 3700 is SM of FIG. 35. In at least one embodiment, SM 3700 includes, without limitation, an instruction cache 3702, one or more scheduler units 3704, a register file 3708, one or more processing cores (“cores”) 3710, one or more special function units (“SFUs”) 3712, one or more load/store units (“LSUs”) 3714, an interconnect network 3716, a shared memory/level one (“L1”) cache 3718, and/or any suitable combination thereof.

[0642] In at least one embodiment, a work distribution unit dispatches tasks for execution on general processing clusters (“GPCs”) of parallel processing units (“PPUs”) and each task is allocated to a particular Data Processing Cluster (“DPC”) within a GPC and, if a task is

embodiment, scheduler unit 3704 receives tasks from a work distribution unit and manages instruction scheduling for one or more thread blocks assigned to SM 3700. In at least one embodiment, scheduler unit 3704 schedules thread blocks for execution as warps of parallel threads, wherein each thread block is allocated at least one warp. In at least one embodiment, each warp executes threads. In at least one embodiment, scheduler unit 3704 manages a plurality of different thread blocks, allocating warps to different thread blocks and then dispatching instructions from plurality of different cooperative groups to various functional units (e.g., processing cores 3710, SFUs 3712, and LSUs 3714) during each clock cycle.

[0643] In at least one embodiment, Cooperative Groups may refer to a programming model for organizing groups of communicating threads that allows developers to express granularity at which threads are communicating, enabling expression of richer, more efficient parallel decompositions. In at least one embodiment, cooperative launch APIs support synchronization amongst thread blocks for execution of parallel algorithms. In at least one embodiment, applications of conventional programming models provide a single, simple construct for synchronizing cooperating threads: a barrier across all threads of a thread block (e.g., `syncthreads()` function). However, in at least one embodiment, programmers may define groups of threads at smaller than thread block granularities and synchronize within defined groups to enable greater performance, design flexibility, and software reuse in form of collective group-wide function interfaces. In at least one embodiment, Cooperative Groups enables programmers to define groups of threads explicitly at sub-block (i.e., as small as a single thread) and multi-block granularities, and to perform collective operations such as synchronization on threads in a cooperative group. In at least one embodiment, that programming model supports clean composition across software boundaries, so that libraries and utility functions can synchronize safely within their local context without having to make assumptions about convergence. In at least one embodiment, Cooperative Groups primitives enable new patterns of cooperative parallelism, including, without limitation, producer-consumer parallelism, opportunistic parallelism, and global synchronization across an entire grid of thread blocks.

[0644] In at least one embodiment, a dispatch unit 3706 is configured to transmit instructions to one or more functional units and scheduler unit 3704 and includes, without limitation, two dispatch units 3706 that enable two different instructions from a common

unit 3704 includes a single dispatch unit 3706 or additional dispatch units 3706.

[0645] In at least one embodiment, each SM 3700, in at least one embodiment, includes, without limitation, register file 3708 that provides a set of registers for functional units of SM 3700. In at least one embodiment, register file 3708 is divided between each functional unit such that each functional unit is allocated a dedicated portion of register file 3708. In at least one embodiment, register file 3708 is divided between different warps being executed by SM 3700 and register file 3708 provides temporary storage for operands connected to data paths of functional units. In at least one embodiment, each SM 3700 comprises, without limitation, a plurality of L processing cores 3710, where L is a positive integer. In at least one embodiment, SM 3700 includes, without limitation, a large number (e.g., 128 or more) of distinct processing cores 3710. In at least one embodiment, each processing core 3710 includes, without limitation, a fully-pipelined, single-precision, double-precision, and/or mixed precision processing unit that includes, without limitation, a floating point arithmetic logic unit and an integer arithmetic logic unit. In at least one embodiment, floating point arithmetic logic units implement IEEE 754-2008 standard for floating point arithmetic. In at least one embodiment, processing cores 3710 include, without limitation, 64 single-precision (32-bit) floating point cores, 64 integer cores, 32 double-precision (64-bit) floating point cores, and 8 tensor cores.

[0646] Tensor cores are configured to perform matrix operations in accordance with at least one embodiment. In at least one embodiment, one or more tensor cores are included in processing cores 3710. In at least one embodiment, tensor cores are configured to perform deep learning matrix arithmetic, such as convolution operations for neural network training and inferencing. In at least one embodiment, each tensor core operates on a 4x4 matrix and performs a matrix multiply and accumulate operation, $D = A \times B + C$, where A, B, C, and D are 4x4 matrices.

[0647] In at least one embodiment, matrix multiply inputs A and B are 16-bit floating point matrices and accumulation matrices C and D are 16-bit floating point or 32-bit floating point matrices. In at least one embodiment, tensor cores operate on 16-bit floating point input data with 32-bit floating point accumulation. In at least one embodiment, 16-bit floating point multiply uses 64 operations and results in a full precision product that is then accumulated using 32-bit floating point addition with other intermediate products for a 4x4x4 matrix

dimensional matrix operations, built up from these smaller elements, in at least one embodiment. In at least one embodiment, an API, such as a CUDA 9 C++ API, exposes specialized matrix load, matrix multiply and accumulate, and matrix store operations to efficiently use tensor cores from a CUDA-C++ program. In at least one embodiment, at a
5 CUDA level, a warp-level interface assumes 16x16 size matrices spanning all 32 threads of warp.

[0648] In at least one embodiment, each SM 3700 comprises, without limitation, M SFUs 3712 that perform special functions (e.g., attribute evaluation, reciprocal square root, and
10 like). In at least one embodiment, SFUs 3712 include, without limitation, a tree traversal unit configured to traverse a hierarchical tree data structure. In at least one embodiment, SFUs 3712 include, without limitation, a texture unit configured to perform texture map filtering operations. In at least one embodiment, texture units are configured to load texture maps (e.g., a 2D array of texels) from memory and sample texture maps to produce sampled texture
15 values for use in shader programs executed by SM 3700. In at least one embodiment, texture maps are stored in shared memory/L1 cache 3718. In at least one embodiment, texture units implement texture operations such as filtering operations using mip-maps (e.g., texture maps of varying levels of detail), in accordance with at least one embodiment. In at least one embodiment, each SM 3700 includes, without limitation, two texture units.

[0649] Each SM 3700 comprises, without limitation, N LSUs 3714 that implement load and store operations between shared memory/L1 cache 3718 and register file 3708, in at least one embodiment. Interconnect network 3716 connects each functional unit to register file 3708 and LSU 3714 to register file 3708 and shared memory/L1 cache 3718 in at least one
20 embodiment. In at least one embodiment, interconnect network 3716 is a crossbar that can be configured to connect any functional units to any registers in register file 3708 and connect
25 LSUs 3714 to register file 3708 and memory locations in shared memory/L1 cache 3718.

[0650] In at least one embodiment, shared memory/L1 cache 3718 is an array of on-chip memory that allows for data storage and communication between SM 3700 and primitive engine and between threads in SM 3700, in at least one embodiment. In at least one
30 embodiment, shared memory/L1 cache 3718 comprises, without limitation, 128 KB of storage capacity and is in a path from SM 3700 to a partition unit. In at least one embodiment, shared memory/L1 cache 3718, in at least one embodiment, is used to cache

L2 cache, and memory are backing stores.

[0651] Combining data cache and shared memory functionality into a single memory block provides improved performance for both types of memory accesses, in at least one

5 embodiment. In at least one embodiment, capacity is used or is usable as a cache by programs that do not use shared memory, such as if shared memory is configured to use half of a capacity, and texture and load/store operations can use remaining capacity. Integration within shared memory/L1 cache 3718 enables shared memory/L1 cache 3718 to function as a high-throughput conduit for streaming data while simultaneously providing high-bandwidth
10 and low-latency access to frequently reused data, in accordance with at least one embodiment. In at least one embodiment, when configured for general purpose parallel computation, a simpler configuration can be used compared with graphics processing. In at least one embodiment, fixed function graphics processing units are bypassed, creating a much simpler programming model. In a general purpose parallel computation configuration, a
15 work distribution unit assigns and distributes blocks of threads directly to DPCs, in at least one embodiment. In at least one embodiment, threads in a block execute a common program, using a unique thread ID in calculation to ensure each thread generates unique results, using SM 3700 to execute program and perform calculations, shared memory/L1 cache 3718 to communicate between threads, and LSU 3714 to read and write global memory through
20 shared memory/L1 cache 3718 and memory partition unit. In at least one embodiment, when configured for general purpose parallel computation, SM 3700 writes commands that scheduler unit 3704 can use to launch new work on DPCs.

[0652] In at least one embodiment, a PPU is included in or coupled to a desktop computer, a laptop computer, a tablet computer, servers, supercomputers, a smart-phone (e.g., a
25 wireless, hand-held device), personal digital assistant (“PDA”), a digital camera, a vehicle, a head mounted display, a hand-held electronic device, and more. In at least one embodiment, a PPU is embodied on a single semiconductor substrate. In at least one embodiment, a PPU is included in a system-on-a-chip (“SoC”) along with one or more other devices such as additional PPUs, memory, a reduced instruction set computer (“RISC”) CPU, a memory
30 management unit (“MMU”), a digital-to-analog converter (“DAC”), and like.

[0653] In at least one embodiment, a PPU may be included on a graphics card that includes one or more memory devices. In at least one embodiment, that graphics card may be

one embodiment, that PPU may be an integrated graphics processing unit (“iGPU”) included in chipset of a motherboard.

[0654] Inference and/or training logic 915 are used to perform inferencing and/or training

5 operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to SM 3700. In at least one embodiment, SM 3700 is used to infer or predict information based on a trained
10 machine learning model (e.g., neural network) that has been trained by another processor or system or by SM 3700. In at least one embodiment, SM 3700 may be used to perform one or more neural network use cases described herein.

[0655] Embodiments are disclosed related a virtualized computing platform for advanced computing, such as image inferencing and image processing in medical applications.

15 Without limitation, embodiments may include radiography, magnetic resonance imaging (MRI), nuclear medicine, ultrasound, sonography, elastography, photoacoustic imaging, tomography, echocardiography, functional near-infrared spectroscopy, and magnetic particle imaging, or a combination thereof. In at least one embodiment, a virtualized computing platform and associated processes described herein may additionally or alternatively be used,
20 without limitation, in forensic science analysis, sub-surface detection and imaging (e.g., oil exploration, archaeology, paleontology, etc.), topography, oceanography, geology, osteology, meteorology, intelligent area or object tracking and monitoring, sensor data processing (e.g., RADAR, SONAR, LIDAR, etc.), and/or genomics and gene sequencing.

[0656] With reference to FIG. 38, FIG. 38 is an example data flow diagram for a process

25 3800 of generating and deploying an image processing and inferencing pipeline, in accordance with at least one embodiment. In at least one embodiment, process 3800 may be deployed for use with imaging devices, processing devices, genomics devices, gene sequencing devices, radiology devices, and/or other device types at one or more facilities
30 3802, such as medical facilities, hospitals, healthcare institutes, clinics, research or diagnostic labs, etc. In at least one embodiment, process 3800 may be deployed to perform genomics analysis and inferencing on sequencing data. Examples of genomic analyses that may be

calling, mutation detection, and gene expression quantification.

[0657] In at least one embodiment, process 3800 may be executed within a training system 3804 and/or a deployment system 3806. In at least one embodiment, training system 3804
5 may be used to perform training, deployment, and implementation of machine learning models (e.g., neural networks, object detection algorithms, computer vision algorithms, etc.) for use in deployment system 3806. In at least one embodiment, deployment system 3806 may be configured to offload processing and compute resources among a distributed computing environment to reduce infrastructure requirements at facility 3802. In at least one
10 embodiment, deployment system 3806 may provide a streamlined platform for selecting, customizing, and implementing virtual instruments for use with imaging devices (e.g., MRI, CT Scan, X-Ray, Ultrasound, etc.) or sequencing devices at facility 3802. In at least one embodiment, virtual instruments may include software-defined applications for performing one or more processing operations with respect to imaging data generated by imaging
15 devices, sequencing devices, radiology devices, and/or other device types. In at least one embodiment, one or more applications in a pipeline may use or call upon services (e.g., inference, visualization, compute, AI, etc.) of deployment system 3806 during execution of applications.

[0658] In at least one embodiment, some of applications used in advanced processing and
20 inferencing pipelines may use machine learning models or other AI to perform one or more processing steps. In at least one embodiment, machine learning models may be trained at facility 3802 using data 3808 (such as imaging data) generated at facility 3802 (and stored on one or more picture archiving and communication system (PACS) servers at facility 3802), may be trained using imaging or sequencing data 3808 from another facility or facilities (e.g.,
25 a different hospital, lab, clinic, etc.), or a combination thereof. In at least one embodiment, training system 3804 may be used to provide applications, services, and/or other resources for generating working, deployable machine learning models for deployment system 3806.

[0659] In at least one embodiment, a model registry 3824 may be backed by object storage that may support versioning and object metadata. In at least one embodiment, object storage
30 may be accessible through, for example, a cloud storage (e.g., a cloud 3926 of FIG. 39) compatible application programming interface (API) from within a cloud platform. In at least one embodiment, machine learning models within model registry 3824 may be uploaded, listed,

one embodiment, an API may provide access to methods that allow users with appropriate credentials to associate models with applications, such that models may be executed as part of execution of containerized instantiations of applications.

- 5 [0660] In at least one embodiment, a training pipeline 3904 (FIG. 39) may include a scenario where facility 3802 is training their own machine learning model, or has an existing machine learning model that needs to be optimized or updated. In at least one embodiment, imaging data 3808 generated by imaging device(s), sequencing devices, and/or other device types may be received. In at least one embodiment, once imaging data 3808 is received, AI-
- 10 assisted annotation 3810 may be used to aid in generating annotations corresponding to imaging data 3808 to be used as ground truth data for a machine learning model. In at least one embodiment, AI-assisted annotation 3810 may include one or more machine learning models (e.g., convolutional neural networks (CNNs)) that may be trained to generate annotations corresponding to certain types of imaging data 3808 (e.g., from certain devices)
- 15 and/or certain types of anomalies in imaging data 3808. In at least one embodiment, AI-assisted annotations 3810 may then be used directly, or may be adjusted or fine-tuned using an annotation tool (e.g., by a researcher, a clinician, a doctor, a scientist, etc.), to generate ground truth data. In at least one embodiment, in some examples, labeled clinic data 3812 (e.g., annotations provided by a clinician, doctor, scientist, technician, etc.) may be used as
- 20 ground truth data for training a machine learning model. In at least one embodiment, AI-assisted annotations 3810, labeled clinic data 3812, or a combination thereof may be used as ground truth data for training a machine learning model. In at least one embodiment, a trained machine learning model may be referred to as an output model 3816, and may be used by deployment system 3806, as described herein.
- 25 [0661] In at least one embodiment, training pipeline 3904 (FIG. 39) may include a scenario where facility 3802 needs a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3806, but facility 3802 may not currently have such a machine learning model (or may not have a model that is optimized, efficient, or effective for such purposes). In at least one embodiment, an existing
- 30 machine learning model may be selected from model registry 3824. In at least one embodiment, model registry 3824 may include machine learning models trained to perform a variety of different inference tasks on imaging data. In at least one embodiment, machine learning models in model registry 3824 may have been trained on imaging data from different

machine learning models may have been trained on imaging data from one location, two locations, or any number of locations. In at least one embodiment, when being trained on imaging data from a specific location, training may take place at that location, or at least in a manner that protects confidentiality of imaging data or restricts imaging data from being transferred off-premises (e.g., to comply with HIPAA regulations, privacy regulations, etc.). In at least one embodiment, once a model is trained – or partially trained – at one location, a machine learning model may be added to model registry 3824. In at least one embodiment, a machine learning model may then be retrained, or updated, at any number of other facilities, and a retrained or updated model may be made available in model registry 3824. In at least one embodiment, a machine learning model may then be selected from model registry 3824 – and referred to as output model 3816 – and may be used in deployment system 3806 to perform one or more processing tasks for one or more applications of a deployment system.

[0662] In at least one embodiment, training pipeline 3904 (FIG. 39) may be used in a scenario that includes facility 3802 requiring a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3806, but facility 3802 may not currently have such a machine learning model (or may not have a model that is optimized, efficient, or effective for such purposes). In at least one embodiment, a machine learning model selected from model registry 3824 might not be fine-tuned or optimized for imaging data 3808 generated at facility 3802 because of differences in populations, genetic variations, robustness of training data used to train a machine learning model, diversity in anomalies of training data, and/or other issues with training data. In at least one embodiment, AI-assisted annotation 3810 may be used to aid in generating annotations corresponding to imaging data 3808 to be used as ground truth data for retraining or updating a machine learning model. In at least one embodiment, labeled clinic data 3812 (e.g., annotations provided by a clinician, doctor, scientist, etc.) may be used as ground truth data for training a machine learning model. In at least one embodiment, retraining or updating a machine learning model may be referred to as model training 3814. In at least one embodiment, model training 3814 – e.g., AI-assisted annotations 3810, labeled clinic data 3812, or a combination thereof – may be used as ground truth data for retraining or updating a machine learning model.

[0663] In at least one embodiment, deployment system 3806 may include software 3818, services 3820, hardware 3822, and/or other components, features, and functionality. In at

software 3818 may be built on top of services 3820 and may use services 3820 to perform some or all of processing tasks, and services 3820 and software 3818 may be built on top of hardware 3822 and use hardware 3822 to execute processing, storage, and/or other compute tasks of deployment system 3806.

[0664] In at least one embodiment, software 3818 may include any number of different containers, where each container may execute an instantiation of an application. In at least one embodiment, each application may perform one or more processing tasks in an advanced processing and inferencing pipeline (e.g., inferencing, object detection, feature detection, segmentation, image enhancement, calibration, etc.). In at least one embodiment, for each type of imaging device (e.g., CT, MRI, X-Ray, ultrasound, sonography, echocardiography, etc.), sequencing device, radiology device, genomics device, etc., there may be any number of containers that may perform a data processing task with respect to imaging data 3808 (or other data types, such as those described herein) generated by a device. In at least one embodiment, an advanced processing and inferencing pipeline may be defined based on selections of different containers that are desired or required for processing imaging data 3808, in addition to containers that receive and configure imaging data for use by each container and/or for use by facility 3802 after processing through a pipeline (e.g., to convert outputs back to a usable data type, such as digital imaging and communications in medicine (DICOM) data, radiology information system (RIS) data, clinical information system (CIS) data, remote procedure call (RPC) data, data substantially compliant with a representation state transfer (REST) interface, data substantially compliant with a file-based interface, and/or raw data, for storage and display at facility 3802). In at least one embodiment, a combination of containers within software 3818 (e.g., that make up a pipeline) may be referred to as a virtual instrument (as described in more detail herein), and a virtual instrument may leverage services 3820 and hardware 3822 to execute some or all processing tasks of applications instantiated in containers.

[0665] In at least one embodiment, a data processing pipeline may receive input data (e.g., imaging data 3808) in a DICOM, RIS, CIS, REST compliant, RPC, raw, and/or other format in response to an inference request (e.g., a request from a user of deployment system 3806, such as a clinician, a doctor, a radiologist, etc.). In at least one embodiment, input data may be representative of one or more images, video, and/or other data representations generated by one or more imaging devices, sequencing devices, radiology devices, genomics devices,

part of data processing pipeline to prepare data for processing by one or more applications. In at least one embodiment, post-processing may be performed on an output of one or more inferencing tasks or other processing tasks of a pipeline to prepare an output data for a next application and/or to prepare output data for transmission and/or use by a user (e.g., as a response to an inference request). In at least one embodiment, inferencing tasks may be performed by one or more machine learning models, such as trained or deployed neural networks, which may include output models 3816 of training system 3804.

[0666] In at least one embodiment, tasks of data processing pipeline may be encapsulated in a container(s) that each represent a discrete, fully functional instantiation of an application and virtualized computing environment that is able to reference machine learning models. In at least one embodiment, containers or applications may be published into a private (e.g., limited access) area of a container registry (described in more detail herein), and trained or deployed models may be stored in model registry 3824 and associated with one or more applications. In at least one embodiment, images of applications (e.g., container images) may be available in a container registry, and once selected by a user from a container registry for deployment in a pipeline, an image may be used to generate a container for an instantiation of an application for use by a user's system.

[0667] In at least one embodiment, developers (e.g., software developers, clinicians, doctors, etc.) may develop, publish, and store applications (e.g., as containers) for performing image processing and/or inferencing on supplied data. In at least one embodiment, development, publishing, and/or storing may be performed using a software development kit (SDK) associated with a system (e.g., to ensure that an application and/or container developed is compliant with or compatible with a system). In at least one embodiment, an application that is developed may be tested locally (e.g., at a first facility, on data from a first facility) with an SDK which may support at least some of services 3820 as a system (e.g., system 3900 of FIG. 39). In at least one embodiment, because DICOM objects may contain anywhere from one to hundreds of images or other data types, and due to a variation in data, a developer may be responsible for managing (e.g., setting constructs for, building pre-processing into an application, etc.) extraction and preparation of incoming DICOM data. In at least one embodiment, once validated by system 3900 (e.g., for accuracy, safety, patient privacy, etc.), an application may be available in a container registry for selection and/or implementation by a user (e.g., a hospital, clinic, lab, healthcare provider, etc.) to perform

user.

[0668] In at least one embodiment, developers may then share applications or containers through a network for access and use by users of a system (e.g., system 3900 of FIG. 39). In at least one embodiment, completed and validated applications or containers may be stored in a container registry and associated machine learning models may be stored in model registry 3824. In at least one embodiment, a requesting entity (e.g., a user at a medical facility) – who provides an inference or image processing request – may browse a container registry and/or model registry 3824 for an application, container, dataset, machine learning model, etc., select a desired combination of elements for inclusion in data processing pipeline, and submit an imaging processing request. In at least one embodiment, a request may include input data (and associated patient data, in some examples) that is necessary to perform a request, and/or may include a selection of application(s) and/or machine learning models to be executed in processing a request. In at least one embodiment, a request may then be passed to one or more components of deployment system 3806 (e.g., a cloud) to perform processing of data processing pipeline. In at least one embodiment, processing by deployment system 3806 may include referencing selected elements (e.g., applications, containers, models, etc.) from a container registry and/or model registry 3824. In at least one embodiment, once results are generated by a pipeline, results may be returned to a user for reference (e.g., for viewing in a viewing application suite executing on a local, on-premises workstation or terminal). In at least one embodiment, a radiologist may receive results from an data processing pipeline including any number of application and/or containers, where results may include anomaly detection in X-rays, CT scans, MRIs, etc.

[0669] In at least one embodiment, to aid in processing or execution of applications or containers in pipelines, services 3820 may be leveraged. In at least one embodiment, services 3820 may include compute services, artificial intelligence (AI) services, visualization services, and/or other service types. In at least one embodiment, services 3820 may provide functionality that is common to one or more applications in software 3818, so functionality may be abstracted to a service that may be called upon or leveraged by applications. In at least one embodiment, functionality provided by services 3820 may run dynamically and more efficiently, while also scaling well by allowing applications to process data in parallel (e.g., using a parallel computing platform 3930 (FIG. 39)). In at least one embodiment, rather than each application that shares a same functionality offered by a service 3820 being

and among various applications. In at least one embodiment, services may include an inference server or engine that may be used for executing detection or segmentation tasks, as non-limiting examples. In at least one embodiment, a model training service may be included that may provide machine learning model training and/or retraining capabilities. In at least one embodiment, a data augmentation service may further be included that may provide GPU accelerated data (e.g., DICOM, RIS, CIS, REST compliant, RPC, raw, etc.) extraction, resizing, scaling, and/or other augmentation. In at least one embodiment, a visualization service may be used that may add image rendering effects – such as ray-tracing, rasterization, denoising, sharpening, etc. – to add realism to two-dimensional (2D) and/or three-dimensional (3D) models. In at least one embodiment, virtual instrument services may be included that provide for beam-forming, segmentation, inferencing, imaging, and/or support for other applications within pipelines of virtual instruments.

[0670] In at least one embodiment, where a service 3820 includes an AI service (e.g., an inference service), one or more machine learning models associated with an application for anomaly detection (e.g., tumors, growth abnormalities, scarring, etc.) may be executed by calling upon (e.g., as an API call) an inference service (e.g., an inference server) to execute machine learning model(s), or processing thereof, as part of application execution. In at least one embodiment, where another application includes one or more machine learning models for segmentation tasks, an application may call upon an inference service to execute machine learning models for performing one or more of processing operations associated with segmentation tasks. In at least one embodiment, software 3818 implementing advanced processing and inferencing pipeline that includes segmentation application and anomaly detection application may be streamlined because each application may call upon a same inference service to perform one or more inferencing tasks.

[0671] In at least one embodiment, hardware 3822 may include GPUs, CPUs, graphics cards, an AI/deep learning system (e.g., an AI supercomputer, such as NVIDIA's DGX supercomputer system), a cloud platform, or a combination thereof. In at least one embodiment, different types of hardware 3822 may be used to provide efficient, purpose-built support for software 3818 and services 3820 in deployment system 3806. In at least one embodiment, use of GPU processing may be implemented for processing locally (e.g., at facility 3802), within an AI/deep learning system, in a cloud system, and/or in other processing components of deployment system 3806 to improve efficiency, accuracy, and

heart attack detection (e.g., in real-time), image quality in rendering, etc. In at least one embodiment, a facility may include imaging devices, genomics devices, sequencing devices, and/or other device types on-premises that may leverage GPUs to generate imaging data

5 representative of a subject's anatomy.

[0672] In at least one embodiment, software 3818 and/or services 3820 may be optimized for GPU processing with respect to deep learning, machine learning, and/or high-performance computing, as non-limiting examples. In at least one embodiment, at least some of computing environment of deployment system 3806 and/or training system 3804 may be
10 executed in a datacenter one or more supercomputers or high performance computing systems, with GPU optimized software (e.g., hardware and software combination of NVIDIA's DGX system). In at least one embodiment, datacenters may be compliant with provisions of HIPAA, such that receipt, processing, and transmission of imaging data and/or other patient data is securely handled with respect to privacy of patient data. In at least one
15 embodiment, hardware 3822 may include any number of GPUs that may be called upon to perform processing of data in parallel, as described herein. In at least one embodiment, cloud platform may further include GPU processing for GPU-optimized execution of deep learning tasks, machine learning tasks, or other computing tasks. In at least one embodiment, cloud platform (e.g., NVIDIA's NGC) may be executed using an AI/deep learning
20 supercomputer(s) and/or GPU-optimized software (e.g., as provided on NVIDIA's DGX systems) as a hardware abstraction and scaling platform. In at least one embodiment, cloud platform may integrate an application container clustering system or orchestration system (e.g., KUBERNETES) on multiple GPUs to enable seamless scaling and load balancing.

[0673] FIG. 39 is a system diagram for an example system 3900 for generating and
25 deploying an imaging deployment pipeline, in accordance with at least one embodiment. In at least one embodiment, system 3900 may be used to implement process 3800 of FIG. 38 and/or other processes including advanced processing and inferencing pipelines. In at least one embodiment, system 3900 may include training system 3804 and deployment system 3806. In at least one embodiment, training system 3804 and deployment system 3806 may be
30 implemented using software 3818, services 3820, and/or hardware 3822, as described herein.

[0674] In at least one embodiment, system 3900 (e.g., training system 3804 and/or deployment system 3806) may implemented in a cloud computing environment (e.g., using

respect to a healthcare services facility, or as a combination of both cloud and local computing resources. In at least one embodiment, in embodiments where cloud computing is implemented, patient data may be separated from, or unprocessed by, by one or more components of system 3900 that would render processing non-compliant with HIPAA and/or other data handling and privacy regulations or laws. In at least one embodiment, access to APIs in cloud 3926 may be restricted to authorized users through enacted security measures or protocols. In at least one embodiment, a security protocol may include web tokens that may be signed by an authentication (e.g., AuthN, AuthZ, Gluecon, etc.) service and may carry appropriate authorization. In at least one embodiment, APIs of virtual instruments (described herein), or other instantiations of system 3900, may be restricted to a set of public IPs that have been vetted or authorized for interaction.

[0675] In at least one embodiment, various components of system 3900 may communicate between and among one another using any of a variety of different network types, including but not limited to local area networks (LANs) and/or wide area networks (WANs) via wired and/or wireless communication protocols. In at least one embodiment, communication between facilities and components of system 3900 (e.g., for transmitting inference requests, for receiving results of inference requests, etc.) may be communicated over a data bus or data busses, wireless data protocols (Wi-Fi), wired data protocols (e.g., Ethernet), etc.

[0676] In at least one embodiment, training system 3804 may execute training pipelines 3904, similar to those described herein with respect to FIG. 38. In at least one embodiment, where one or more machine learning models are to be used in deployment pipelines 3910 by deployment system 3806, training pipelines 3904 may be used to train or retrain one or more (e.g., pre-trained) models, and/or implement one or more of pre-trained models 3906 (e.g., without a need for retraining or updating). In at least one embodiment, as a result of training pipelines 3904, output model(s) 3816 may be generated. In at least one embodiment, training pipelines 3904 may include any number of processing steps, such as but not limited to imaging data (or other input data) conversion or adaption (e.g., using DICOM adapter 3902A to convert DICOM images to another format suitable for processing by respective machine learning models, such as Neuroimaging Informatics Technology Initiative (NIfTI) format), AI-assisted annotation 3810, labeling or annotating of imaging data 3808 to generate labeled clinic data 3812, model selection from a model registry, model training 3814, training, retraining, or updating models, and/or other processing steps. In at least one embodiment, for

pipelines 3904 may be used. In at least one embodiment, training pipeline 3904 similar to a first example described with respect to FIG. 38 may be used for a first machine learning model, training pipeline 3904 similar to a second example described with respect to FIG. 38 may be used for a second machine learning model, and training pipeline 3904 similar to a third example described with respect to FIG. 38 may be used for a third machine learning model. In at least one embodiment, any combination of tasks within training system 3804 may be used depending on what is required for each respective machine learning model. In at least one embodiment, one or more of machine learning models may already be trained and ready for deployment so machine learning models may not undergo any processing by training system 3804, and may be implemented by deployment system 3806.

[0677] In at least one embodiment, output model(s) 3816 and/or pre-trained model(s) 3906 may include any types of machine learning models depending on implementation or embodiment. In at least one embodiment, and without limitation, machine learning models used by system 3900 may include machine learning model(s) using linear regression, logistic regression, decision trees, support vector machines (SVM), Naïve Bayes, k-nearest neighbor (Knn), K means clustering, random forest, dimensionality reduction algorithms, gradient boosting algorithms, neural networks (e.g., auto-encoders, convolutional, recurrent, perceptrons, Long/Short Term Memory (LSTM), Hopfield, Boltzmann, deep belief, deconvolutional, generative adversarial, liquid state machine, etc.), and/or other types of machine learning models.

[0678] In at least one embodiment, training pipelines 3904 may include AI-assisted annotation, as described in more detail herein with respect to at least FIG. 42B. In at least one embodiment, labeled clinic data 3812 (e.g., traditional annotation) may be generated by any number of techniques. In at least one embodiment, labels or other annotations may be generated within a drawing program (e.g., an annotation program), a computer aided design (CAD) program, a labeling program, another type of program suitable for generating annotations or labels for ground truth, and/or may be hand drawn, in some examples. In at least one embodiment, ground truth data may be synthetically produced (e.g., generated from computer models or renderings), real produced (e.g., designed and produced from real-world data), machine-automated (e.g., using feature analysis and learning to extract features from data and then generate labels), human annotated (e.g., labeler, or annotation expert, defines location of labels), and/or a combination thereof. In at least one embodiment, for each

may be corresponding ground truth data generated by training system 3804. In at least one embodiment, AI-assisted annotation may be performed as part of deployment pipelines 3910; either in addition to, or in lieu of AI-assisted annotation included in training pipelines 3904.

5 In at least one embodiment, system 3900 may include a multi-layer platform that may include a software layer (e.g., software 3818) of diagnostic applications (or other application types) that may perform one or more medical imaging and diagnostic functions. In at least one embodiment, system 3900 may be communicatively coupled to (e.g., via encrypted links) PACS server networks of one or more facilities. In at least one embodiment, system 3900
10 may be configured to access and referenced data (e.g., DICOM data, RIS data, raw data, CIS data, REST compliant data, RPC data, raw data, etc.) from PACS servers (e.g., via a DICOM adapter 3902, or another data type adapter such as RIS, CIS, REST compliant, RPC, raw, etc.) to perform operations, such as training machine learning models, deploying machine learning models, image processing, inferencing, and/or other operations.

15 [0679] In at least one embodiment, a software layer may be implemented as a secure, encrypted, and/or authenticated API through which applications or containers may be invoked (e.g., called) from an external environment(s) (e.g., facility 3802). In at least one embodiment, applications may then call or execute one or more services 3820 for performing compute, AI, or visualization tasks associated with respective applications, and software 3818
20 and/or services 3820 may leverage hardware 3822 to perform processing tasks in an effective and efficient manner.

[0680] In at least one embodiment, deployment system 3806 may execute deployment pipelines 3910. In at least one embodiment, deployment pipelines 3910 may include any number of applications that may be sequentially, non-sequentially, or otherwise applied to
25 imaging data (and/or other data types) generated by imaging devices, sequencing devices, genomics devices, etc. – including AI-assisted annotation, as described above. In at least one embodiment, as described herein, a deployment pipeline 3910 for an individual device may be referred to as a virtual instrument for a device (e.g., a virtual ultrasound instrument, a virtual CT scan instrument, a virtual sequencing instrument, etc.). In at least one
30 embodiment, for a single device, there may be more than one deployment pipeline 3910 depending on information desired from data generated by a device. In at least one embodiment, where detections of anomalies are desired from an MRI machine, there may be

MRI machine, there may be a second deployment pipeline 3910.

[0681] In at least one embodiment, applications available for deployment pipelines 3910 may include any application that may be used for performing processing tasks on imaging data or other data from devices. In at least one embodiment, different applications may be responsible for image enhancement, segmentation, reconstruction, anomaly detection, object detection, feature detection, treatment planning, dosimetry, beam planning (or other radiation treatment procedures), and/or other analysis, image processing, or inferencing tasks. In at least one embodiment, deployment system 3806 may define constructs for each of applications, such that users of deployment system 3806 (e.g., medical facilities, labs, clinics, etc.) may understand constructs and adapt applications for implementation within their respective facility. In at least one embodiment, an application for image reconstruction may be selected for inclusion in deployment pipeline 3910, but data type generated by an imaging device may be different from a data type used within an application. In at least one embodiment, DICOM adapter 3902B (and/or a DICOM reader) or another data type adapter or reader (e.g., RIS, CIS, REST compliant, RPC, raw, etc.) may be used within deployment pipeline 3910 to convert data to a form useable by an application within deployment system 3806. In at least one embodiment, access to DICOM, RIS, CIS, REST compliant, RPC, raw, and/or other data type libraries may be accumulated and pre-processed, including decoding, extracting, and/or performing any convolutions, color corrections, sharpness, gamma, and/or other augmentations to data. In at least one embodiment, DICOM, RIS, CIS, REST compliant, RPC, and/or raw data may be unordered and a pre-pass may be executed to organize or sort collected data. In at least one embodiment, because various applications may share common image operations, in some embodiments, a data augmentation library (e.g., as one of services 3820) may be used to accelerate these operations. In at least one embodiment, to avoid bottlenecks of conventional processing approaches that rely on CPU processing, parallel computing platform 3930 may be used for GPU acceleration of these processing tasks.

[0682] In at least one embodiment, an image reconstruction application may include a processing task that includes use of a machine learning model. In at least one embodiment, a user may desire to use their own machine learning model, or to select a machine learning model from model registry 3824. In at least one embodiment, a user may implement their own machine learning model or select a machine learning model for inclusion in an

be selectable and customizable, and by defining constructs of applications, deployment and implementation of applications for a particular user are presented as a more seamless user experience. In at least one embodiment, by leveraging other features of system 3900 – such as services 3820 and hardware 3822 – deployment pipelines 3910 may be even more user friendly, provide for easier integration, and produce more accurate, efficient, and timely results.

[0683] In at least one embodiment, deployment system 3806 may include a user interface 3914 (e.g., a graphical user interface, a web interface, etc.) that may be used to select applications for inclusion in deployment pipeline(s) 3910, arrange applications, modify or change applications or parameters or constructs thereof, use and interact with deployment pipeline(s) 3910 during set-up and/or deployment, and/or to otherwise interact with deployment system 3806. In at least one embodiment, although not illustrated with respect to training system 3804, user interface 3914 (or a different user interface) may be used for selecting models for use in deployment system 3806, for selecting models for training, or retraining, in training system 3804, and/or for otherwise interacting with training system 3804.

[0684] In at least one embodiment, pipeline manager 3912 may be used, in addition to an application orchestration system 3928, to manage interaction between applications or containers of deployment pipeline(s) 3910 and services 3820 and/or hardware 3822. In at least one embodiment, pipeline manager 3912 may be configured to facilitate interactions from application to application, from application to service 3820, and/or from application or service to hardware 3822. In at least one embodiment, although illustrated as included in software 3818, this is not intended to be limiting, and in some examples (e.g., as illustrated in FIG. 40) pipeline manager 3912 may be included in services 3820. In at least one embodiment, application orchestration system 3928 (e.g., Kubernetes, DOCKER, etc.) may include a container orchestration system that may group applications into containers as logical units for coordination, management, scaling, and deployment. In at least one embodiment, by associating applications from deployment pipeline(s) 3910 (e.g., a reconstruction application, a segmentation application, etc.) with individual containers, each application may execute in a self-contained environment (e.g., at a kernel level) to increase speed and efficiency.

may be individually developed, modified, and deployed (e.g., a first user or developer may develop, modify, and deploy a first application and a second user or developer may develop, modify, and deploy a second application separate from a first user or developer), which may allow for focus on, and attention to, a task of a single application and/or container(s) without being hindered by tasks of another application(s) or container(s). In at least one embodiment, communication, and cooperation between different containers or applications may be aided by pipeline manager 3912 and application orchestration system 3928. In at least one embodiment, so long as an expected input and/or output of each container or application is known by a system (e.g., based on constructs of applications or containers), application orchestration system 3928 and/or pipeline manager 3912 may facilitate communication among and between, and sharing of resources among and between, each of applications or containers. In at least one embodiment, because one or more of applications or containers in deployment pipeline(s) 3910 may share same services and resources, application orchestration system 3928 may orchestrate, load balance, and determine sharing of services or resources between and among various applications or containers. In at least one embodiment, a scheduler may be used to track resource requirements of applications or containers, current usage or planned usage of these resources, and resource availability. In at least one embodiment, a scheduler may thus allocate resources to different applications and distribute resources between and among applications in view of requirements and availability of a system. In some examples, a scheduler (and/or other component of application orchestration system 3928) may determine resource availability and distribution based on constraints imposed on a system (e.g., user constraints), such as quality of service (QoS), urgency of need for data outputs (e.g., to determine whether to execute real-time processing or delayed processing), etc.

[0686] In at least one embodiment, services 3820 leveraged by and shared by applications or containers in deployment system 3806 may include compute services 3916, AI services 3918, visualization services 3920, and/or other service types. In at least one embodiment, applications may call (e.g., execute) one or more of services 3820 to perform processing operations for an application. In at least one embodiment, compute services 3916 may be leveraged by applications to perform super-computing or other high-performance computing (HPC) tasks. In at least one embodiment, compute service(s) 3916 may be leveraged to perform parallel processing (e.g., using a parallel computing platform 3930) for processing

substantially simultaneously. In at least one embodiment, parallel computing platform 3930 (e.g., NVIDIA's CUDA) may enable general purpose computing on GPUs (GPGPU) (e.g., GPUs 3922). In at least one embodiment, a software layer of parallel computing platform 3930 may provide access to virtual instruction sets and parallel computational elements of GPUs, for execution of compute kernels. In at least one embodiment, parallel computing platform 3930 may include memory and, in some embodiments, a memory may be shared between and among multiple containers, and/or between and among different processing tasks within a single container. In at least one embodiment, inter-process communication (IPC) calls may be generated for multiple containers and/or for multiple processes within a container to use same data from a shared segment of memory of parallel computing platform 3930 (e.g., where multiple different stages of an application or multiple applications are processing same information). In at least one embodiment, rather than making a copy of data and moving data to different locations in memory (e.g., a read/write operation), same data in same location of a memory may be used for any number of processing tasks (e.g., at a same time, at different times, etc.). In at least one embodiment, as data is used to generate new data as a result of processing, this information of a new location of data may be stored and shared between various applications. In at least one embodiment, location of data and a location of updated or modified data may be part of a definition of how a payload is understood within containers.

[0687] In at least one embodiment, AI services 3918 may be leveraged to perform inferencing services for executing machine learning model(s) associated with applications (e.g., tasked with performing one or more processing tasks of an application). In at least one embodiment, AI services 3918 may leverage AI system 3924 to execute machine learning model(s) (e.g., neural networks, such as CNNs) for segmentation, reconstruction, object detection, feature detection, classification, and/or other inferencing tasks. In at least one embodiment, applications of deployment pipeline(s) 3910 may use one or more of output models 3816 from training system 3804 and/or other models of applications to perform inference on imaging data (e.g., DICOM data, RIS data, CIS data, REST compliant data, RPC data, raw data, etc.). In at least one embodiment, two or more examples of inferencing using application orchestration system 3928 (e.g., a scheduler) may be available. In at least one embodiment, a first category may include a high priority/low latency path that may achieve higher service level agreements, such as for performing inference on urgent requests

second category may include a standard priority path that may be used for requests that may be non-urgent or where analysis may be performed at a later time. In at least one embodiment, application orchestration system 3928 may distribute resources (e.g., services 3820 and/or hardware 3822) based on priority paths for different inferencing tasks of AI services 3918.

[0688] In at least one embodiment, shared storage may be mounted to AI services 3918 within system 3900. In at least one embodiment, shared storage may operate as a cache (or other storage device type) and may be used to process inference requests from applications.

In at least one embodiment, when an inference request is submitted, a request may be received by a set of API instances of deployment system 3806, and one or more instances may be selected (e.g., for best fit, for load balancing, etc.) to process a request. In at least one embodiment, to process a request, a request may be entered into a database, a machine learning model may be located from model registry 3824 if not already in a cache, a validation step may ensure appropriate machine learning model is loaded into a cache (e.g., shared storage), and/or a copy of a model may be saved to a cache. In at least one embodiment, a scheduler (e.g., of pipeline manager 3912) may be used to launch an application that is referenced in a request if an application is not already running or if there are not enough instances of an application. In at least one embodiment, if an inference server is not already launched to execute a model, an inference server may be launched. In at least one embodiment, any number of inference servers may be launched per model. In at least one embodiment, in a pull model, in which inference servers are clustered, models may be cached whenever load balancing is advantageous. In at least one embodiment, inference servers may be statically loaded in corresponding, distributed servers.

[0689] In at least one embodiment, inferencing may be performed using an inference server that runs in a container. In at least one embodiment, an instance of an inference server may be associated with a model (and optionally a plurality of versions of a model). In at least one embodiment, if an instance of an inference server does not exist when a request to perform inference on a model is received, a new instance may be loaded. In at least one embodiment, when starting an inference server, a model may be passed to an inference server such that a same container may be used to serve different models so long as inference server is running as a different instance.

given application may be received, and a container (e.g., hosting an instance of an inference server) may be loaded (if not already), and a start procedure may be called. In at least one embodiment, pre-processing logic in a container may load, decode, and/or perform any
5 additional pre-processing on incoming data (e.g., using a CPU(s) and/or GPU(s)). In at least one embodiment, once data is prepared for inference, a container may perform inference as necessary on data. In at least one embodiment, this may include a single inference call on one image (e.g., a hand X-ray), or may require inference on hundreds of images (e.g., a chest CT). In at least one embodiment, an application may summarize results before completing,
10 which may include, without limitation, a single confidence score, pixel level-segmentation, voxel-level segmentation, generating a visualization, or generating text to summarize findings. In at least one embodiment, different models or applications may be assigned different priorities. For example, some models may have a real-time (TAT less than one minute) priority while others may have lower priority (e.g., TAT less than 10 minutes). In at
15 least one embodiment, model execution times may be measured from requesting institution or entity and may include partner network traversal time, as well as execution on an inference service.

[0691] In at least one embodiment, transfer of requests between services 3820 and inference applications may be hidden behind a software development kit (SDK), and robust
20 transport may be provide through a queue. In at least one embodiment, a request will be placed in a queue via an API for an individual application/tenant ID combination and an SDK will pull a request from a queue and give a request to an application. In at least one embodiment, a name of a queue may be provided in an environment from where an SDK will pick it up. In at least one embodiment, asynchronous communication through a queue may
25 be useful as it may allow any instance of an application to pick up work as it becomes available. In at least one embodiment, results may be transferred back through a queue, to ensure no data is lost. In at least one embodiment, queues may also provide an ability to segment work, as highest priority work may go to a queue with most instances of an application connected to it, while lowest priority work may go to a queue with a single
30 instance connected to it that processes tasks in an order received. In at least one embodiment, an application may run on a GPU-accelerated instance generated in cloud 3926, and an inference service may perform inferencing on a GPU.

generate visualizations for viewing outputs of applications and/or deployment pipeline(s) 3910. In at least one embodiment, GPUs 3922 may be leveraged by visualization services 3920 to generate visualizations. In at least one embodiment, rendering effects, such as ray-tracing, may be implemented by visualization services 3920 to generate higher quality visualizations. In at least one embodiment, visualizations may include, without limitation, 2D image renderings, 3D volume renderings, 3D volume reconstruction, 2D tomographic slices, virtual reality displays, augmented reality displays, etc. In at least one embodiment, virtualized environments may be used to generate a virtual interactive display or environment (e.g., a virtual environment) for interaction by users of a system (e.g., doctors, nurses, radiologists, etc.). In at least one embodiment, visualization services 3920 may include an internal visualizer, cinematics, and/or other rendering or image processing capabilities or functionality (e.g., ray tracing, rasterization, internal optics, etc.).

[0693] In at least one embodiment, hardware 3822 may include GPUs 3922, AI system 3924, cloud 3926, and/or any other hardware used for executing training system 3804 and/or deployment system 3806. In at least one embodiment, GPUs 3922 (e.g., NVIDIA's TESLA and/or QUADRO GPUs) may include any number of GPUs that may be used for executing processing tasks of compute services 3916, AI services 3918, visualization services 3920, other services, and/or any of features or functionality of software 3818. For example, with respect to AI services 3918, GPUs 3922 may be used to perform pre-processing on imaging data (or other data types used by machine learning models), post-processing on outputs of machine learning models, and/or to perform inferencing (e.g., to execute machine learning models). In at least one embodiment, cloud 3926, AI system 3924, and/or other components of system 3900 may use GPUs 3922. In at least one embodiment, cloud 3926 may include a GPU-optimized platform for deep learning tasks. In at least one embodiment, AI system 3924 may use GPUs, and cloud 3926 – or at least a portion tasked with deep learning or inferencing – may be executed using one or more AI systems 3924. As such, although hardware 3822 is illustrated as discrete components, this is not intended to be limiting, and any components of hardware 3822 may be combined with, or leveraged by, any other components of hardware 3822.

[0694] In at least one embodiment, AI system 3924 may include a purpose-built computing system (e.g., a super-computer or an HPC) configured for inferencing, deep learning, machine learning, and/or other artificial intelligence tasks. In at least one embodiment, AI

stack) that may be executed using a plurality of GPUs 3922, in addition to CPUs, RAM, storage, and/or other components, features, or functionality. In at least one embodiment, one or more AI systems 3924 may be implemented in cloud 3926 (e.g., in a data center) for performing some or all of AI-based processing tasks of system 3900.

[0695] In at least one embodiment, cloud 3926 may include a GPU-accelerated infrastructure (e.g., NVIDIA's NGC) that may provide a GPU-optimized platform for executing processing tasks of system 3900. In at least one embodiment, cloud 3926 may include an AI system(s) 3924 for performing one or more of AI-based tasks of system 3900 (e.g., as a hardware abstraction and scaling platform). In at least one embodiment, cloud 3926 may integrate with application orchestration system 3928 leveraging multiple GPUs to enable seamless scaling and load balancing between and among applications and services 3820. In at least one embodiment, cloud 3926 may tasked with executing at least some of services 3820 of system 3900, including compute services 3916, AI services 3918, and/or visualization services 3920, as described herein. In at least one embodiment, cloud 3926 may perform small and large batch inference (e.g., executing NVIDIA's TENSOR RT), provide an accelerated parallel computing API and platform 3930 (e.g., NVIDIA's CUDA), execute application orchestration system 3928 (e.g., KUBERNETES), provide a graphics rendering API and platform (e.g., for ray-tracing, 2D graphics, 3D graphics, and/or other rendering techniques to produce higher quality cinematics), and/or may provide other functionality for system 3900.

[0696] In at least one embodiment, in an effort to preserve patient confidentiality (e.g., where patient data or records are to be used off-premises), cloud 3926 may include a registry – such as a deep learning container registry. In at least one embodiment, a registry may store containers for instantiations of applications that may perform pre-processing, post-processing, or other processing tasks on patient data. In at least one embodiment, cloud 3926 may receive data that includes patient data as well as sensor data in containers, perform requested processing for just sensor data in those containers, and then forward a resultant output and/or visualizations to appropriate parties and/or devices (e.g., on-premises medical devices used for visualization or diagnoses), all without having to extract, store, or otherwise access patient data. In at least one embodiment, confidentiality of patient data is preserved in compliance with HIPAA and/or other data regulations.

processing imaging data, in accordance with at least one embodiment. In at least one embodiment, system 3900 – and specifically deployment system 3806 – may be used to customize, update, and/or integrate deployment pipeline(s) 3910A into one or more production environments. In at least one embodiment, deployment pipeline 3910A of FIG. 40 includes a non-limiting example of a deployment pipeline 3910A that may be custom defined by a particular user (or team of users) at a facility (e.g., at a hospital, clinic, lab, research environment, etc.). In at least one embodiment, to define deployment pipelines 3910A for a CT scanner 4002, a user may select – from a container registry, for example – one or more applications that perform specific functions or tasks with respect to imaging data generated by CT scanner 4002. In at least one embodiment, applications may be applied to deployment pipeline 3910A as containers that may leverage services 3820 and/or hardware 3822 of system 3900. In addition, deployment pipeline 3910A may include additional processing tasks or applications that may be implemented to prepare data for use by applications (e.g., DICOM adapter 3902B and DICOM reader 4006 may be used in deployment pipeline 3910A to prepare data for use by CT reconstruction 4008, organ segmentation 4010, etc.). In at least one embodiment, deployment pipeline 3910A may be customized or selected for consistent deployment, one time use, or for another frequency or interval. In at least one embodiment, a user may desire to have CT reconstruction 4008 and organ segmentation 4010 for several subjects over a specific interval, and thus may deploy pipeline 3910A for that period of time. In at least one embodiment, a user may select, for each request from system 3900, applications that a user wants to perform processing on that data for that request. In at least one embodiment, deployment pipeline 3910A may be adjusted at any interval and, because of adaptability and scalability of a container structure within system 3900, this may be a seamless process.

[0698] In at least one embodiment, deployment pipeline 3910A of FIG. 40 may include CT scanner 4002 generating imaging data of a patient or subject. In at least one embodiment, imaging data from CT scanner 4002 may be stored on a PACS server(s) 4004 associated with a facility housing CT scanner 4002. In at least one embodiment, PACS server(s) 4004 may include software and/or hardware components that may directly interface with imaging modalities (e.g., CT scanner 4002) at a facility. In at least one embodiment, DICOM adapter 3902B may enable sending and receipt of DICOM objects using DICOM protocols. In at least one embodiment, DICOM adapter 3902B may aid in preparation or configuration of

one embodiment, once DICOM data is processed through DICOM adapter 3902B, pipeline manager 3912 may route data through to deployment pipeline 3910A. In at least one embodiment, DICOM reader 4006 may extract image files and any associated metadata from DICOM data (e.g., raw sinogram data, as illustrated in visualization 4016A). In at least one embodiment, working files that are extracted may be stored in a cache for faster processing by other applications in deployment pipeline 3910A. In at least one embodiment, once DICOM reader 4006 has finished extracting and/or storing data, a signal of completion may be communicated to pipeline manager 3912. In at least one embodiment, pipeline manager 3912 may then initiate or call upon one or more other applications or containers in deployment pipeline 3910A.

[0699] In at least one embodiment, CT reconstruction 4008 application and/or container may be executed once data (e.g., raw sinogram data) is available for processing by CT reconstruction 4008 application. In at least one embodiment, CT reconstruction 4008 may read raw sinogram data from a cache, reconstruct an image file out of raw sinogram data (e.g., as illustrated in visualization 4016B), and store resulting image file in a cache. In at least one embodiment, at completion of reconstruction, pipeline manager 3912 may be signaled that reconstruction task is complete. In at least one embodiment, once reconstruction is complete, and a reconstructed image file may be stored in a cache (or other storage device), organ segmentation 4010 application and/or container may be triggered by pipeline manager 3912. In at least one embodiment, organ segmentation 4010 application and/or container may read an image file from a cache, normalize or convert an image file to format suitable for inference (e.g., convert an image file to an input resolution of a machine learning model), and run inference against a normalized image. In at least one embodiment, to run inference on a normalized image, organ segmentation 4010 application and/or container may rely on services 3820, and pipeline manager 3912 and/or application orchestration system 3928 may facilitate use of services 3820 by organ segmentation 4010 application and/or container. In at least one embodiment, for example, organ segmentation 4010 application and/or container may leverage AI services 3918 to perform inference on a normalized image, and AI services 3918 may leverage hardware 3822 (e.g., AI system 3924) to execute AI services 3918. In at least one embodiment, a result of an inference may be a mask file (e.g., as illustrated in visualization 4016C) that may be stored in a cache (or other storage device).

extracted from DICOM data have completed processing, a signal may be generated for pipeline manager 3912. In at least one embodiment, pipeline manager 3912 may then execute DICOM writer 4012 to read results from a cache (or other storage device), package results into a DICOM format (e.g., as DICOM output 4014) for use by users at a facility who generated a request. In at least one embodiment, DICOM output 4014 may then be transmitted to DICOM adapter 3902B to prepare DICOM output 4014 for storage on PACS server(s) 4004 (e.g., for viewing by a DICOM viewer at a facility). In at least one embodiment, in response to a request for reconstruction and segmentation, visualizations 4016B and 4016C may be generated and available to a user for diagnoses, research, and/or for other purposes.

[0701] Although illustrated as consecutive application in deployment pipeline 3910A, CT reconstruction 4008 and organ segmentation 4010 applications may be processed in parallel in at least one embodiment. In at least one embodiment, where applications do not have dependencies on one another, and data is available for each application (e.g., after DICOM reader 4006 extracts data), applications may be executed at a same time, substantially at a same time, or with some overlap. In at least one embodiment, where two or more applications require similar services 3820, a scheduler of system 3900 may be used to load balance and distribute compute or processing resources between and among various applications. In at least one embodiment, in some embodiments, parallel computing platform 3930 may be used to perform parallel processing for applications to decrease run-time of deployment pipeline 3910A to provide real-time results.

[0702] In at least one embodiment, and with reference to FIGS. 41A-41B, deployment system 3806 may be implemented as one or more virtual instruments to perform different functionalities – such as image processing, segmentation, enhancement, AI, visualization, and inferencing – with imaging devices (e.g., CT scanners, X-ray machines, MRI machines, etc.), sequencing devices, genomics devices, and/or other device types. In at least one embodiment, system 3900 may allow for creation and provision of virtual instruments that may include a software-defined deployment pipeline 3910 that may receive raw/unprocessed input data generated by a device(s) and output processed/reconstructed data. In at least one embodiment, deployment pipelines 3910 (e.g., 3910A and 3910B) that represent virtual instruments may implement intelligence into a pipeline, such as by leveraging machine learning models, to provide containerized inference support to a system. In at least one

instantiations of applications. In at least one embodiment, such as where real-time processing is desired, deployment pipelines 3910 representing virtual instruments may be static (e.g., containers and/or applications may be set), while in other examples, container and/or applications for virtual instruments may be selected (e.g., on a per-request basis) from a pool of applications or resources (e.g., within a container registry).

[0703] In at least one embodiment, system 3900 may be instantiated or executed as one or more virtual instruments on-premise at a facility in, for example, a computing system deployed next to or otherwise in communication with a radiology machine, an imaging device, and/or another device type at a facility. In at least one embodiment, however, an on-premise installation may be instantiated or executed within a computing system of a device itself (e.g., a computing system integral to an imaging device), in a local datacenter (e.g., a datacenter on-premise), and/or in a cloud-environment (e.g., in cloud 3926). In at least one embodiment, deployment system 3806, operating as a virtual instrument, may be instantiated by a supercomputer or other HPC system in some examples. In at least one embodiment, on-premise installation may allow for high-bandwidth uses (via, for example, higher throughput local communication interfaces, such as RF over Ethernet) for real-time processing. In at least one embodiment, real-time or near real-time processing may be particularly useful where a virtual instrument supports an ultrasound device or other imaging modality where immediate visualizations are expected or required for accurate diagnoses and analyses. In at least one embodiment, a cloud-computing architecture may be capable of dynamic bursting to a cloud computing service provider, or other compute cluster, when local demand exceeds on-premise capacity or capability. In at least one embodiment, a cloud architecture, when implemented, may be tuned for training neural networks or other machine learning models, as described herein with respect to training system 3804. In at least one embodiment, with training pipelines in place, machine learning models may be continuously learn and improve as they process additional data from devices they support. In at least one embodiment, virtual instruments may be continually improved using additional data, new data, existing machine learning models, and/or new or updated machine learning models.

[0704] In at least one embodiment, a computing system may include some or all of hardware 3822 described herein, and hardware 3822 may be distributed in any of a number of ways including within a device, as part of a computing device coupled to and located proximate a device, in a local datacenter at a facility, and/or in cloud 3926. In at least one

created in software (e.g., as discrete containerized instantiations of applications), behavior, operation, and configuration of virtual instruments, as well as outputs generated by virtual instruments, may be modified or customized as desired, without having to change or alter raw output of a device that a virtual instrument supports.

[0705] FIG. 41A includes an example data flow diagram of a virtual instrument supporting an ultrasound device, in accordance with at least one embodiment. In at least one embodiment, deployment pipeline 3910B may leverage one or more of services 3820 of system 3900. In at least one embodiment, deployment pipeline 3910B and services 3820 may leverage hardware 3822 of a system either locally or in cloud 3926. In at least one embodiment, although not illustrated, process 4100 may be facilitated by pipeline manager 3912, application orchestration system 3928, and/or parallel computing platform 3930.

[0706] In at least one embodiment, process 4100 may include receipt of imaging data from an ultrasound device 4102. In at least one embodiment, imaging data may be stored on

PACS server(s) in a DICOM format (or other format, such as RIS, CIS, REST compliant, RPC, raw, etc.), and may be received by system 3900 for processing through deployment pipeline 3910 selected or customized as a virtual instrument (e.g., a virtual ultrasound) for ultrasound device 4102. In at least one embodiment, imaging data may be received directly from an imaging device (e.g., ultrasound device 4102) and processed by a virtual instrument.

In at least one embodiment, a transducer or other signal converter communicatively coupled between an imaging device and a virtual instrument may convert signal data generated by an imaging device to image data that may be processed by a virtual instrument. In at least one embodiment, raw data and/or image data may be applied to DICOM reader 4006 to extract data for use by applications or containers of deployment pipeline 3910B. In at least one embodiment, DICOM reader 4006 may leverage data augmentation library 4114 (e.g., NVIDIA's DALI) as a service 3820 (e.g., as one of compute service(s) 3916) for extracting, resizing, rescaling, and/or otherwise preparing data for use by applications or containers.

[0707] In at least one embodiment, once data is prepared, a reconstruction 4106 application and/or container may be executed to reconstruct data from ultrasound device 4102 into an image file. In at least one embodiment, after reconstruction 4106, or at a same time as reconstruction 4106, a detection 4108 application and/or container may be executed for anomaly detection, object detection, feature detection, and/or other detection tasks related to

used during detection 4108 to identify anomalies, objects, features, etc. In at least one embodiment, detection 4108 application may leverage an inference engine 4116 (e.g., as one of AI service(s) 3918) to perform inference on data to generate detections. In at least one
5 embodiment, one or more machine learning models (e.g., from training system 3804) may be executed or called by detection 4108 application.

[0708] In at least one embodiment, once reconstruction 4106 and/or detection 4108 is/are complete, data output from these application and/or containers may be used to generate visualizations 4110, such as visualization 4112 (e.g., a grayscale output) displayed on a
10 workstation or display terminal. In at least one embodiment, visualization may allow a technician or other user to visualize results of deployment pipeline 3910B with respect to ultrasound device 4102. In at least one embodiment, visualization 4110 may be executed by leveraging a render component 4118 of system 3900 (e.g., one of visualization service(s) 3920). In at least one embodiment, render component 4118 may execute a 2D, OpenGL, or
15 ray-tracing service to generate visualization 4112.

[0709] FIG. 41B includes an example data flow diagram of a virtual instrument supporting a CT scanner, in accordance with at least one embodiment. In at least one embodiment, deployment pipeline 3910C may leverage one or more of services 3820 of system 3900. In at least one embodiment, deployment pipeline 3910C and services 3820 may leverage hardware
20 3822 of a system either locally or in cloud 3926. In at least one embodiment, although not illustrated, process 4120 may be facilitated by pipeline manager 3912, application orchestration system 3928, and/or parallel computing platform 3930.

[0710] In at least one embodiment, process 4120 may include CT scanner 4122 generating raw data that may be received by DICOM reader 4006 (e.g., directly, via a PACS server
25 4004, after processing, etc.). In at least one embodiment, a Virtual CT (instantiated by deployment pipeline 3910C) may include a first, real-time pipeline for monitoring a patient (e.g., patient movement detection AI 4126) and/or for adjusting or optimizing exposure of CT scanner 4122 (e.g., using exposure control AI 4124). In at least one embodiment, one or more of applications (e.g., 4124 and 4126) may leverage a service 3820, such as AI service(s)
30 3918. In at least one embodiment, outputs of exposure control AI 4124 application (or container) and/or patient movement detection AI 4126 application (or container) may be used

of CT scanner 4122) and/or informing a patient to move less.

[0711] In at least one embodiment, deployment pipeline 3910C may include a non-real-time pipeline for analyzing data generated by CT scanner 4122. In at least one embodiment, a second pipeline may include CT reconstruction 4008 application and/or container, a coarse detection AI 4128 application and/or container, a fine detection AI 4132 application and/or container (e.g., where certain results are detected by coarse detection AI 4128), a

visualization 4130 application and/or container, and a DICOM writer 4012 (and/or other data type writer, such as RIS, CIS, REST compliant, RPC, raw, etc.) application and/or container.

In at least one embodiment, raw data generated by CT scanner 4122 may be passed through pipelines of deployment pipeline 3910C (instantiated as a virtual CT instrument) to generate results. In at least one embodiment, results from DICOM writer 4012 may be transmitted for display and/or may be stored on PACS server(s) 4004 for later retrieval, analysis, or display by a technician, practitioner, or other user.

[0712] FIG. 42A illustrates a data flow diagram for a process 4200 to train, retrain, or update a machine learning model, in accordance with at least one embodiment. In at least one embodiment, process 4200 may be executed using, as a non-limiting example, system 3900 of FIG. 39. In at least one embodiment, process 4200 may leverage services 3820 and/or hardware 3822 of system 3900, as described herein. In at least one embodiment, refined models 4212 generated by process 4200 may be executed by deployment system 3806 for one or more containerized applications in deployment pipelines 3910.

[0713] In at least one embodiment, model training 3814 may include retraining or updating an initial model 4204 (e.g., a pre-trained model) using new training data (e.g., new input data, such as customer dataset 4206, and/or new ground truth data associated with input data). In

at least one embodiment, to retrain, or update, initial model 4204, output or loss layer(s) of initial model 4204 may be reset, or deleted, and/or replaced with an updated or new output or loss layer(s). In at least one embodiment, initial model 4204 may have previously fine-tuned parameters (e.g., weights and/or biases) that remain from prior training, so training or retraining 3814 may not take as long or require as much processing as training a model from scratch. In at least one embodiment, during model training 3814, by having reset or replaced output or loss layer(s) of initial model 4204, parameters may be updated and re-tuned for a

generating predictions on new, customer dataset 4206 (e.g., image data 3808 of FIG. 38).

[0714] In at least one embodiment, pre-trained models 3906 may be stored in a data store, or registry (e.g., model registry 3824 of FIG. 38). In at least one embodiment, pre-trained

5 models 3906 may have been trained, at least in part, at one or more facilities other than a facility executing process 4200. In at least one embodiment, to protect privacy and rights of patients, subjects, or clients of different facilities, pre-trained models 3906 may have been trained, on-premise, using customer or patient data generated on-premise. In at least one embodiment, pre-trained models 3906 may be trained using cloud 3926 and/or other

10 hardware 3822, but confidential, privacy protected patient data may not be transferred to, used by, or accessible to any components of cloud 3926 (or other off premise hardware). In at least one embodiment, where a pre-trained model 3906 is trained at using patient data from more than one facility, pre-trained model 3906 may have been individually trained for each facility prior to being trained on patient or customer data from another facility. In at least one
15 embodiment, such as where a customer or patient data has been released of privacy concerns (e.g., by waiver, for experimental use, etc.), or where a customer or patient data is included in a public data set, a customer or patient data from any number of facilities may be used to train pre-trained model 3906 on-premise and/or off premise, such as in a datacenter or other cloud computing infrastructure.

20 [0715] In at least one embodiment, when selecting applications for use in deployment pipelines 3910, a user may also select machine learning models to be used for specific applications. In at least one embodiment, a user may not have a model for use, so a user may select a pre-trained model 3906 to use with an application. In at least one embodiment, pre-trained model 3906 may not be optimized for generating accurate results on customer dataset
25 4206 of a facility of a user (e.g., based on patient diversity, demographics, types of medical imaging devices used, etc.). In at least one embodiment, prior to deploying pre-trained model 3906 into deployment pipeline 3910 for use with an application(s), pre-trained model 3906 may be updated, retrained, and/or fine-tuned for use at a respective facility.

[0716] In at least one embodiment, a user may select pre-trained model 3906 that is to be
30 updated, retrained, and/or fine-tuned, and pre-trained model 3906 may be referred to as initial model 4204 for training system 3804 within process 4200. In at least one embodiment, customer dataset 4206 (e.g., imaging data, genomics data, sequencing data, or other data

may include, without limitation, transfer learning) on initial model 4204 to generate refined model 4212. In at least one embodiment, ground truth data corresponding to customer dataset 4206 may be generated by training system 3804. In at least one embodiment, ground truth data may be generated, at least in part, by clinicians, scientists, doctors, practitioners, at a facility (e.g., as labeled clinic data 3812 of FIG. 38).

[0717] In at least one embodiment, AI-assisted annotation 3810 may be used in some examples to generate ground truth data. In at least one embodiment, AI-assisted annotation 3810 (e.g., implemented using an AI-assisted annotation SDK) may leverage machine learning models (e.g., neural networks) to generate suggested or predicted ground truth data for a customer dataset. In at least one embodiment, user 4210 may use annotation tools within a user interface (a graphical user interface (GUI)) on computing device 4208.

[0718] In at least one embodiment, user 4210 may interact with a GUI via computing device 4208 to edit or fine-tune annotations or auto-annotations. In at least one embodiment, a polygon editing feature may be used to move vertices of a polygon to more accurate or fine-tuned locations.

[0719] In at least one embodiment, once customer dataset 4206 has associated ground truth data, ground truth data (e.g., from AI-assisted annotation, manual labeling, etc.) may be used by during model training 3814 to generate refined model 4212. In at least one embodiment, customer dataset 4206 may be applied to initial model 4204 any number of times, and ground truth data may be used to update parameters of initial model 4204 until an acceptable level of accuracy is attained for refined model 4212. In at least one embodiment, once refined model 4212 is generated, refined model 4212 may be deployed within one or more deployment pipelines 3910 at a facility for performing one or more processing tasks with respect to medical imaging data.

[0720] In at least one embodiment, refined model 4212 may be uploaded to pre-trained models 3906 in model registry 3824 to be selected by another facility. In at least one embodiment, this process may be completed at any number of facilities such that refined model 4212 may be further refined on new datasets any number of times to generate a more universal model.

[0721] FIG. 42B is an example illustration of a client-server architecture 4232 to enhance annotation tools with pre-trained annotation models, in accordance with at least one

instantiated based on a client-server architecture 4232. In at least one embodiment, annotation tools 4236 in imaging applications may aid radiologists, for example, identify organs and abnormalities. In at least one embodiment, imaging applications may include software tools that help user 4210 to identify, as a non-limiting example, a few extreme points on a particular organ of interest in raw images 4234 (e.g., in a 3D MRI or CT scan) and receive auto-annotated results for all 2D slices of a particular organ. In at least one embodiment, results may be stored in a data store as training data 4238 and used as (for example and without limitation) ground truth data for training. In at least one embodiment, when computing device 4208 sends extreme points for AI-assisted annotation 3810, a deep learning model, for example, may receive this data as input and return inference results of a segmented organ or abnormality. In at least one embodiment, pre-instantiated annotation tools, such as AI-Assisted Annotation Tool 4236B in FIG. 42B, may be enhanced by making API calls (e.g., API Call 4244) to a server, such as an Annotation Assistant Server 4240 that may include a set of pre-trained models 4242 stored in an annotation model registry, for example. In at least one embodiment, an annotation model registry may store pre-trained models 4242 (e.g., machine learning models, such as deep learning models) that are pre-trained to perform AI-assisted annotation on a particular organ or abnormality. In at least one embodiment, these models may be further updated by using training pipelines 3904. In at least one embodiment, pre-installed annotation tools may be improved over time as new labeled clinic data 3812 is added.

[0722] Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B.

[0723] At least one embodiment of the disclosure can be described in view of the following clauses:

[0724] In at least one embodiment, a single semiconductor platform may refer to a sole unitary semiconductor-based integrated circuit or chip. In at least one embodiment, multi-chip modules may be used with increased connectivity which simulate on-chip operation, and make substantial improvements over utilizing a conventional central processing unit (“CPU”) and bus implementation. In at least one embodiment, various modules may also be situated separately or in various combinations of semiconductor platforms per desires of user.

of machine-readable executable code or computer control logic algorithms are stored in main memory 1504 and/or secondary storage. Computer programs, if executed by one or more processors, enable system 1500 to perform various functions in accordance with at least one
5 embodiment. In at least one embodiment, memory 1504, storage, and/or any other storage are possible examples of computer-readable media. In at least one embodiment, secondary storage may refer to any suitable storage device or system such as a hard disk drive and/or a removable storage drive, representing a floppy disk drive, a magnetic tape drive, a compact disk drive, digital versatile disk (“DVD”) drive, recording device, universal serial bus
10 (“USB”) flash memory, etc. In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of CPU 1502, parallel processing system 1512, an integrated circuit capable of at least a portion of capabilities of both CPU 1502, parallel processing system 1512, a chipset (e.g., a group of integrated circuits designed to work and sold as a unit for performing related functions, etc.), and/or any suitable
15 combination of integrated circuit(s).

[0726] In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of a general computer system, a circuit board system, a game console system dedicated for entertainment purposes, an application-specific system, and more. In at least one embodiment, computer system 1500 may take form of a desktop
20 computer, a laptop computer, a tablet computer, servers, supercomputers, a smart-phone (e.g., a wireless, hand-held device), personal digital assistant (“PDA”), a digital camera, a vehicle, a head mounted display, a hand-held electronic device, a mobile phone device, a television, workstation, game consoles, embedded system, and/or any other type of logic.

[0727] In at least one embodiment, parallel processing system 1512 includes, without
25 limitation, a plurality of parallel processing units (“PPUs”) 1514 and associated memories 1516. In at least one embodiment, PPUs 1514 are connected to a host processor or other peripheral devices via an interconnect 1518 and a switch 1520 or multiplexer. In at least one embodiment, parallel processing system 1512 distributes computational tasks across PPUs 1514 which can be parallelizable — for example, as part of distribution of computational
30 tasks across multiple graphics processing unit (“GPU”) thread blocks. In at least one embodiment, memory is shared and accessible (e.g., for read and/or write access) across some or all of PPUs 1514, although such shared memory may incur performance penalties relative to use of local memory and registers resident to a PPU 1514. In at least one

__syncthreads(), wherein all threads in a block (e.g., executed across multiple PPU 1514) to reach a certain point of execution of code before proceeding.

[0728] At least one embodiment of the disclosure can be described in view of the following clauses:

[0729] Clause 1. A computer system comprising one or more processors and computer-readable memory storing instructions executable by the one or more processors to cause the computer system to at least: identify a policy to cause a machine to execute at least one movement, the policy including at least a plurality of policy layers comprising: a first policy layer to cause the machine to execute a first motion that reaches an unbiased state, the first motion to be limited by at least a first parameter associated with the machine and a second parameter associated with an area in which the machine is to operate, and a second policy layer to cause the machine to execute a second motion without influencing the unbiased state associated with the first policy layer; and execute the policy to cause the machine to execute the at least one movement, the at least one movement comprising at least the first motion and the second motion.

[0730] Clause 2. The computer system according to clause 1, wherein the first policy layer comprises at least a first nonlinear second-order differential equation and the second policy layer comprises at least a second nonlinear second-order differential equation, the first nonlinear second-order differential equation unbiased to cause the first motion to come to rest and the second nonlinear second-order differential equation unbiased to cause the second motion to come to rest.

[0731] Clause 3. The computer system according to at least one or more of the preceding clauses, wherein the first policy layer is a first geometric fabric comprising a nonlinear second-order differential equation and the second policy layer is a second geometric fabric comprising another second-order differential equation.

[0732] Clause 4. The computer system according to at least one or more of the preceding clauses, wherein the first parameter comprises at least first data comprising a one or more limits associated with a joint of the machine and the second parameter comprises at least second data comprising a target position to be reached by the machine, the target position located in an Euclidean space associated with the area in which the machine is to operate.

preceding clauses, wherein the first policy layer is energized by a first Finsler energy and the second policy layer is energized by a second Finsler energy, the first Finsler energy is homogeneous of degree two and the second Finsler energy is homogenous of degree two.

5 [0734] Clause 6. The computer system according to at least one or more of the preceding clauses, wherein the machine is an articulated robot comprising at least one arm, the first policy layer to cause the at least one arm to execute the first motion comprising a straight line movement limited by the first parameter and the second parameter, and wherein
10 the first parameter corresponds to a joint of the at least one arm and the second parameter corresponds to coordinate in the area in which the articulated robot is to operate.

[0735] Clause 7. The computer system according to at least one or more of the preceding clauses, wherein the second motion is to occur subsequent to the first motion, the second motion to cause a gripper of the machine to execute a movement based on at least one task that the machine is to undertake.

15 [0736] Clause 8. The computer system according to at least one or more of the preceding clauses, wherein the plurality of policy layers comprises a third policy layer, the third policy layer to cause the machine to execute a third motion without influencing the unbiased state associated with the first policy layer and an unbiased state associated with the second policy layer, and wherein the third motion is to cause the machine to avoid at least
20 one obstacle in the area in which the machine is to operate.

[0737] Clause 9. A device comprising: one or more processors and memory storing executable instructions that, as a result of being executed by the one or more processors, cause the device to: generate a first policy layer to cause a machine to execute a first motion that causes the machine to accelerate to reach an unbiased state, and generate a second policy
25 layer to cause the machine to execute a second motion without influencing the unbiased state to be reached by machine; and execute the first policy layer and the second policy layer to cause the machine to execute the first motion and the second motion.

[0738] Clause 10. The device according to at least one or more of the preceding clauses, wherein the one or more processors and the memory storing executable instructions that, as a
30 result of being executed by the one or more processors, further cause the device to generate a policy comprising the first policy layer and the second policy layer, the policy to cause the machine to execute at least one action.

wherein the first policy layer and the second policy layer are generated consecutively starting with the first policy layer and followed by the second policy layer.

[0740] Clause 12. The device according to at least one or more of the preceding clauses, wherein the first policy layer comprises at least a first nonlinear second-order differential equation and the second policy layer comprises at least a second nonlinear second-order differential equation, the first nonlinear second-order differential equation unbiased to cause the first motion to come to rest and the second nonlinear second-order differential equation unbiased to cause the second motion to come to rest.

[0741] Clause 13. The device according to at least one or more of the preceding clauses, wherein the first policy layer is a first geometric fabric comprising a nonlinear second-order differential equation and the second policy layer is a second geometric fabric comprising another second-order differential equation.

[0742] Clause 14. The device according to at least one or more of the preceding clauses, wherein the first policy layer comprises at least first data comprising a one or more limits associated with a joint of the machine and the second policy layer comprises at least second data comprising a target position to be reached by the machine, the target position located in an Euclidean space associated with the area in which a portion of the machine is to operate.

[0743] Clause 15. The device according to at least one or more of the preceding clauses, wherein the first policy layer is energized by a first Finsler energy and the second policy layer is energized by a second Finsler energy.

[0744] Clause 16. The device according to at least one or more of the preceding clauses, wherein the machine is an articulated robot comprising at least one arm, the first policy layer to cause the at least one arm to execute the first motion comprising a substantially straight line movement.

[0745] Clause 17. The device according to at least one or more of the preceding clauses, wherein the second motion is to occur subsequent to the first motion, the second motion to cause a gripper of the machine to execute a movement based on at least one task that the machine is to undertake.

[0746] Clause 18. The device according to at least one or more of the preceding clauses, wherein the one or more processors and the memory storing executable instructions that, as a

third policy layer, the third policy layer to cause the machine to execute a third motion without influencing the unbiased state associated with the first policy layer and an unbiased state associated with the second policy layer, and wherein the third motion is to cause the machine to avoid at least one obstacle in the area in which the machine is to operate.

[0747] Clause 19. A computer-implemented method, comprising: generating a first policy layer to cause a machine to execute a first motion that causes the machine to accelerate to reach an unbiased state, and generating a second policy layer to cause the machine to execute a second motion without influencing the unbiased state to be reached by machine; and executing the first policy layer and the second policy layer to cause the machine to execute the first motion and the second motion.

[0748] Clause 20. The computer-implemented method according to at least one or more of the preceding clauses, further comprising a third policy layer, the third policy layer to cause the machine to execute a third motion without influencing the unbiased state associated with the first policy layer and an unbiased state associated with the second policy layer, and wherein the third motion is to cause the machine to avoid at least one obstacle in the area in which the machine is to operate.

[0749] Clause 21. The computer-implemented method according to at least one or more of the preceding clauses, wherein the first policy layer and the second policy layer are generated consecutively starting with the first policy layer and followed by the second policy layer.

[0750] Clause 22. The computer-implemented method according to at least one or more of the preceding clauses, wherein the first policy layer comprises at least a first nonlinear second-order differential equation and the second policy layer comprises at least a second nonlinear second-order differential equation, the first nonlinear second-order differential equation unbiased to cause the first motion to come to rest and the second nonlinear second-order differential equation unbiased to cause the second motion to come to rest.

[0751] Clause 23. The computer-implemented method according to at least one or more of the preceding clauses, wherein the first policy layer is a first geometric fabric comprising a nonlinear second-order differential equation and the second policy layer is a second geometric fabric comprising another second-order differential equation.

of the preceding clauses, wherein the first policy layer comprises at least first data comprising a one or more limits associated with a joint of the machine and the second policy layer comprises at least second data comprising a target position to be reached by the machine, the target position located in an Euclidean space associated with the area in which a portion of the machine is to operate.

[0753] Other variations are within spirit of present disclosure. Thus, while disclosed techniques are susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in drawings and have been described above in detail. It should be understood, however, that there is no intention to limit disclosure to specific form or forms disclosed, but on contrary, intention is to cover all modifications, alternative constructions, and equivalents falling within spirit and scope of disclosure, as defined in appended claims.

[0754] Use of terms “a” and “an” and “the” and similar referents in context of describing disclosed embodiments (especially in context of following claims) are to be construed to cover both singular and plural, unless otherwise indicated herein or clearly contradicted by context, and not as a definition of a term. Terms “comprising,” “having,” “including,” and “containing” are to be construed as open-ended terms (meaning “including, but not limited to,”) unless otherwise noted. “Connected,” when unmodified and referring to physical connections, is to be construed as partly or wholly contained within, attached to, or joined together, even if there is something intervening. Recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within range, unless otherwise indicated herein and each separate value is incorporated into specification as if it were individually recited herein. In at least one embodiment, use of term “set” (e.g., “a set of items”) or “subset” unless otherwise noted or contradicted by context, is to be construed as a nonempty collection comprising one or more members. Further, unless otherwise noted or contradicted by context, term “subset” of a corresponding set does not necessarily denote a proper subset of corresponding set, but subset and corresponding set may be equal.

[0755] Conjunctive language, such as phrases of form “at least one of A, B, and C,” or “at least one of A, B and C,” unless specifically stated otherwise or otherwise clearly contradicted by context, is otherwise understood with context as used in general to present

and C. For instance, in illustrative example of a set having three members, conjunctive phrases “at least one of A, B, and C” and “at least one of A, B and C” refer to any of following sets: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}. Thus, such conjunctive language is not generally intended to imply that certain embodiments require at least one of A, at least one of B and at least one of C each to be present. In addition, unless otherwise noted or contradicted by context, term “plurality” indicates a state of being plural (e.g., “a plurality of items” indicates multiple items). In at least one embodiment, number of items in a plurality is at least two, but can be more when so indicated either explicitly or by context. Further, unless stated otherwise or otherwise clear from context, phrase “based on” means “based at least in part on” and not “based solely on.”

[0756] Operations of processes described herein can be performed in any suitable order unless otherwise indicated herein or otherwise clearly contradicted by context. In at least one embodiment, a process such as those processes described herein (or variations and/or combinations thereof) is performed under control of one or more computer systems configured with executable instructions and is implemented as code (e.g., executable instructions, one or more computer programs or one or more applications) executing collectively on one or more processors, by hardware or combinations thereof. In at least one embodiment, code is stored on a computer-readable storage medium, for example, in form of a computer program comprising a plurality of instructions executable by one or more processors. In at least one embodiment, a computer-readable storage medium is a non-transitory computer-readable storage medium that excludes transitory signals (e.g., a propagating transient electric or electromagnetic transmission) but includes non-transitory data storage circuitry (e.g., buffers, cache, and queues) within transceivers of transitory signals. In at least one embodiment, code (e.g., executable code or source code) is stored on a set of one or more non-transitory computer-readable storage media having stored thereon executable instructions (or other memory to store executable instructions) that, when executed (i.e., as a result of being executed) by one or more processors of a computer system, cause computer system to perform operations described herein. In at least one embodiment, set of non-transitory computer-readable storage media comprises multiple non-transitory computer-readable storage media and one or more of individual non-transitory storage media of multiple non-transitory computer-readable storage media lack all of code while multiple non-transitory computer-readable storage media collectively store all of code. In at least one

executed by different processors — for example, a non-transitory computer-readable storage medium store instructions and a main central processing unit (“CPU”) executes some of instructions while a graphics processing unit (“GPU”) executes other instructions. In at least one embodiment, different components of a computer system have separate processors and different processors execute different subsets of instructions.

[0757] Accordingly, in at least one embodiment, computer systems are configured to implement one or more services that singly or collectively perform operations of processes described herein and such computer systems are configured with applicable hardware and/or software that enable performance of operations. Further, a computer system that implements at least one embodiment of present disclosure is a single device and, in another embodiment, is a distributed computer system comprising multiple devices that operate differently such that distributed computer system performs operations described herein and such that a single device does not perform all operations.

[0758] Use of any and all examples, or exemplary language (e.g., “such as”) provided herein, is intended merely to better illuminate embodiments of disclosure and does not pose a limitation on scope of disclosure unless otherwise claimed. No language in specification should be construed as indicating any non-claimed element as essential to practice of disclosure.

[0759] All references, including publications, patent applications, and patents, cited herein are hereby incorporated by reference to same extent as if each reference were individually and specifically indicated to be incorporated by reference and were set forth in its entirety herein.

[0760] In description and claims, terms “coupled” and “connected,” along with their derivatives, may be used. It should be understood that these terms may be not intended as synonyms for each other. Rather, in particular examples, “connected” or “coupled” may be used to indicate that two or more elements are in direct or indirect physical or electrical contact with each other. “Coupled” may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other.

[0761] Unless specifically stated otherwise, it may be appreciated that throughout specification terms such as “processing,” “computing,” “calculating,” “determining,” or like, refer to action and/or processes of a computer or computing system, or similar electronic

electronic, quantities within computing system's registers and/or memories into other data similarly represented as physical quantities within computing system's memories, registers or other such information storage, transmission or display devices.

- 5 [0762] In a similar manner, term "processor" may refer to any device or portion of a device that processes electronic data from registers and/or memory and transform that electronic data into other electronic data that may be stored in registers and/or memory. As non-limiting examples, "processor" may be a CPU or a GPU. A "computing platform" may comprise one or more processors. As used herein, "software" processes may include, for example,
- 10 software and/or hardware entities that perform work over time, such as tasks, threads, and intelligent agents. Also, each process may refer to multiple processes, for carrying out instructions in sequence or in parallel, continuously or intermittently. In at least one embodiment, terms "system" and "method" are used herein interchangeably insofar as system may embody one or more methods and methods may be considered a system.
- 15 [0763] In present document, references may be made to obtaining, acquiring, receiving, or inputting analog or digital data into a subsystem, computer system, or computer-implemented machine. In at least one embodiment, process of obtaining, acquiring, receiving, or inputting analog and digital data can be accomplished in a variety of ways such as by receiving data as a parameter of a function call or a call to an application programming interface. In at least
- 20 one embodiment, processes of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a serial or parallel interface. In at least one embodiment, processes of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a computer network from providing entity to acquiring entity. In at least one embodiment, references may also be made to providing,
- 25 outputting, transmitting, sending, or presenting analog or digital data. In various examples, processes of providing, outputting, transmitting, sending, or presenting analog or digital data can be accomplished by transferring data as an input or output parameter of a function call, a parameter of an application programming interface or interprocess communication mechanism.
- 30 [0764] Although descriptions herein set forth example implementations of described techniques, other architectures may be used to implement described functionality, and are intended to be within scope of this disclosure. Furthermore, although specific distributions of

responsibilities might be distributed and divided in different ways, depending on circumstances.

[0765] Furthermore, although subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that subject matter claimed in appended claims is not necessarily limited to specific features or acts described. Rather, specific features and acts are disclosed as exemplary forms of implementing the claims.

CLAIMS

WHAT IS CLAIMED IS:

1. A computer system comprising one or more processors and computer-readable
5 memory storing instructions executable by the one or more processors to cause the computer system to at least:
identify a policy to cause a machine to execute at least one movement, the policy including at least a plurality of policy layers comprising:
a first policy layer to cause the machine to execute a first motion that reaches
10 an unbiased state, the first motion to be limited by at least a first parameter associated with the machine and a second parameter associated with an area in which the machine is to operate, and
a second policy layer to cause the machine to execute a second motion without influencing the unbiased state associated with the first policy layer; and
15 execute the policy to cause the machine to execute the at least one movement, the at least one movement comprising at least the first motion and the second motion.
2. The computer system according to claim 1, wherein the first policy layer comprises at least a first nonlinear second-order differential equation and the second policy layer comprises at least a second nonlinear second-order differential equation, the first
20 nonlinear second-order differential equation unbiased to cause the first motion to come to rest and the second nonlinear second-order differential equation unbiased to cause the second motion to come to rest.
3. The computer system according to claim 1, wherein the first policy layer is a
25 first geometric fabric comprising a nonlinear second-order differential equation and the second policy layer is a second geometric fabric comprising another second-order differential equation.

comprises at least first data comprising a one or more limits associated with a joint of the machine and the second parameter comprises at least second data comprising a target position to be reached by the machine, the target position located in an Euclidean space associated with the area in which the machine is to operate.

5 5. The computer system according to claim 1, wherein the first policy layer is energized by a first Finsler energy and the second policy layer is energized by a second Finsler energy, the first Finsler energy is homogeneous of degree two and the second Finsler energy is homogenous of degree two.

10 6. The computer system according to claim 1, wherein the machine is an articulated robot comprising at least one arm, the first policy layer to cause the at least one arm to execute the first motion comprising a straight line movement limited by the first parameter and the second parameter, and wherein the first parameter corresponds to a joint of the at least one arm and the second parameter corresponds to coordinate in the area in which
15 the articulated robot is to operate.

7. The computer system according to claim 1, wherein the second motion is to occur subsequent to the first motion, the second motion to cause a gripper of the machine to execute a movement based on at least one task that the machine is to undertake.

20 8. The computer system according to claim 1, wherein the plurality of policy layers comprises a third policy layer, the third policy layer to cause the machine to execute a third motion without influencing the unbiased state associated with the first policy layer and an unbiased state associated with the second policy layer, and wherein the third motion is to cause the machine to avoid at least one obstacle in the area in which the machine is to operate.

25 9. A device comprising:
one or more processors and memory storing executable instructions that, as a result of being executed by the one or more processors, cause the device to:

generate a first policy layer to cause a machine to execute a first motion that causes the machine to accelerate to reach an unbiased state, and

30 generate a second policy layer to cause the machine to execute a second motion without influencing the unbiased state to be reached by machine; and

to execute the first motion and the second motion.

10. The device according to claim 9, wherein the one or more processors and the memory storing executable instructions that, as a result of being executed by the one or more
5 processors, further cause the device to generate a policy comprising the first policy layer and the second policy layer, the policy to cause the machine to execute at least one action.

11. The device according to claim 10, wherein the first policy layer and the second policy layer are generated consecutively starting with the first policy layer and followed by the second policy layer.

10 12. The device according to claim 9, wherein the first policy layer comprises at least a first nonlinear second-order differential equation and the second policy layer comprises at least a second nonlinear second-order differential equation, the first nonlinear second-order differential equation unbiased to cause the first motion to come to rest and the second nonlinear second-order differential equation unbiased to cause the second motion to
15 come to rest.

13. The device according to claim 9, wherein the first policy layer is a first geometric fabric comprising a nonlinear second-order differential equation and the second policy layer is a second geometric fabric comprising another second-order differential equation.

20 14. The device according to claim 9, wherein the first policy layer comprises at least first data comprising a one or more limits associated with a joint of the machine and the second policy layer comprises at least second data comprising a target position to be reached by the machine, the target position located in an Euclidean space associated with the area in which a portion of the machine is to operate.

25 15. The device according to claim 9, wherein the first policy layer is energized by a first Finsler energy and the second policy layer is energized by a second Finsler energy.

16. The device according to claim 9, wherein the machine is an articulated robot comprising at least one arm, the first policy layer to cause the at least one arm to execute the first motion comprising a substantially straight line movement.

subsequent to the first motion, the second motion to cause a gripper of the machine to execute a movement based on at least one task that the machine is to undertake.

18. The device according to claim 9, wherein the one or more processors and the
5 memory storing executable instructions that, as a result of being executed by the one or more processors, further cause the device to generate a third policy layer, the third policy layer to cause the machine to execute a third motion without influencing the unbiased state associated with the first policy layer and an unbiased state associated with the second policy layer, and wherein the third motion is to cause the machine to avoid at least one obstacle in the area in
10 which the machine is to operate.

19. A computer-implemented method, comprising:
generating a first policy layer to cause a machine to execute a first motion that causes the machine to accelerate to reach an unbiased state, and
generating a second policy layer to cause the machine to execute a second motion
15 without influencing the unbiased state to be reached by machine; and
executing the first policy layer and the second policy layer to cause the machine to execute the first motion and the second motion.

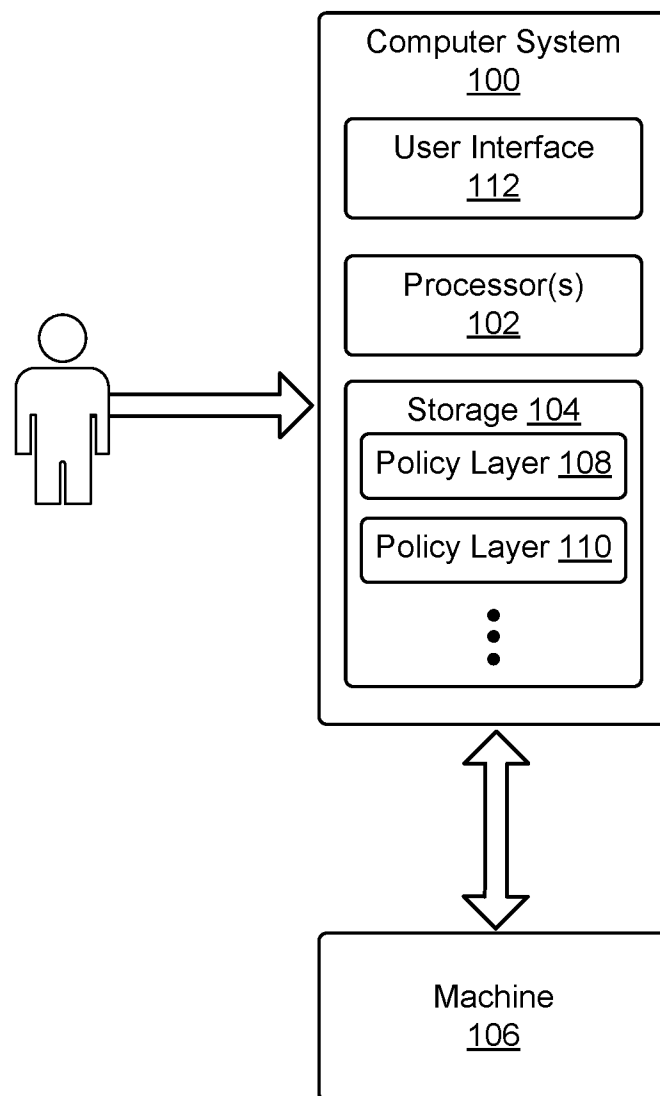
20. The computer-implemented method according to claim 19, further comprising a third policy layer, the third policy layer to cause the machine to execute a third motion
20 without influencing the unbiased state associated with the first policy layer and an unbiased state associated with the second policy layer, and wherein the third motion is to cause the machine to avoid at least one obstacle in the area in which the machine is to operate.

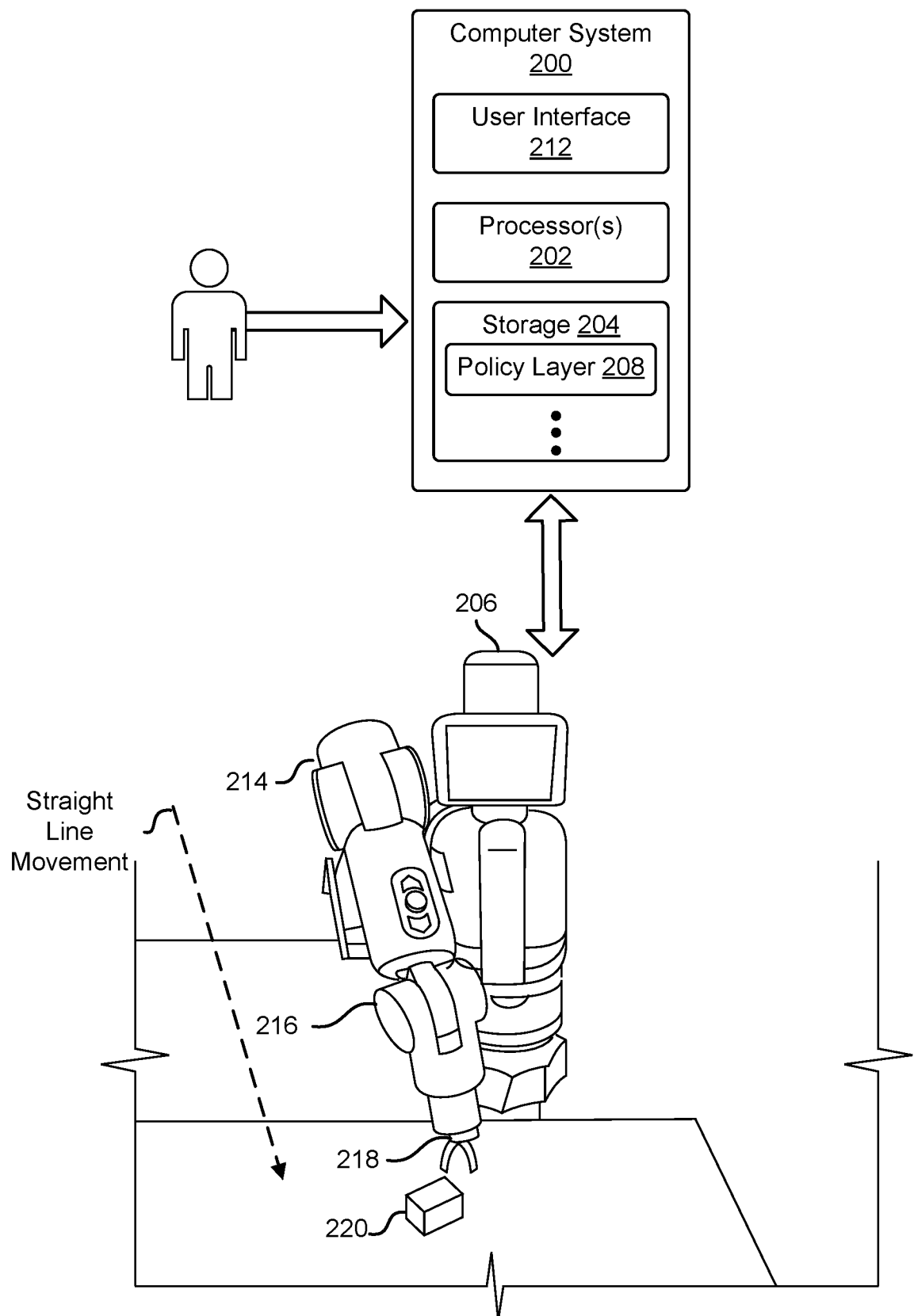
21. The computer-implemented method according to claim 19, wherein the first policy layer and the second policy layer are generated consecutively starting with the first
25 policy layer and followed by the second policy layer.

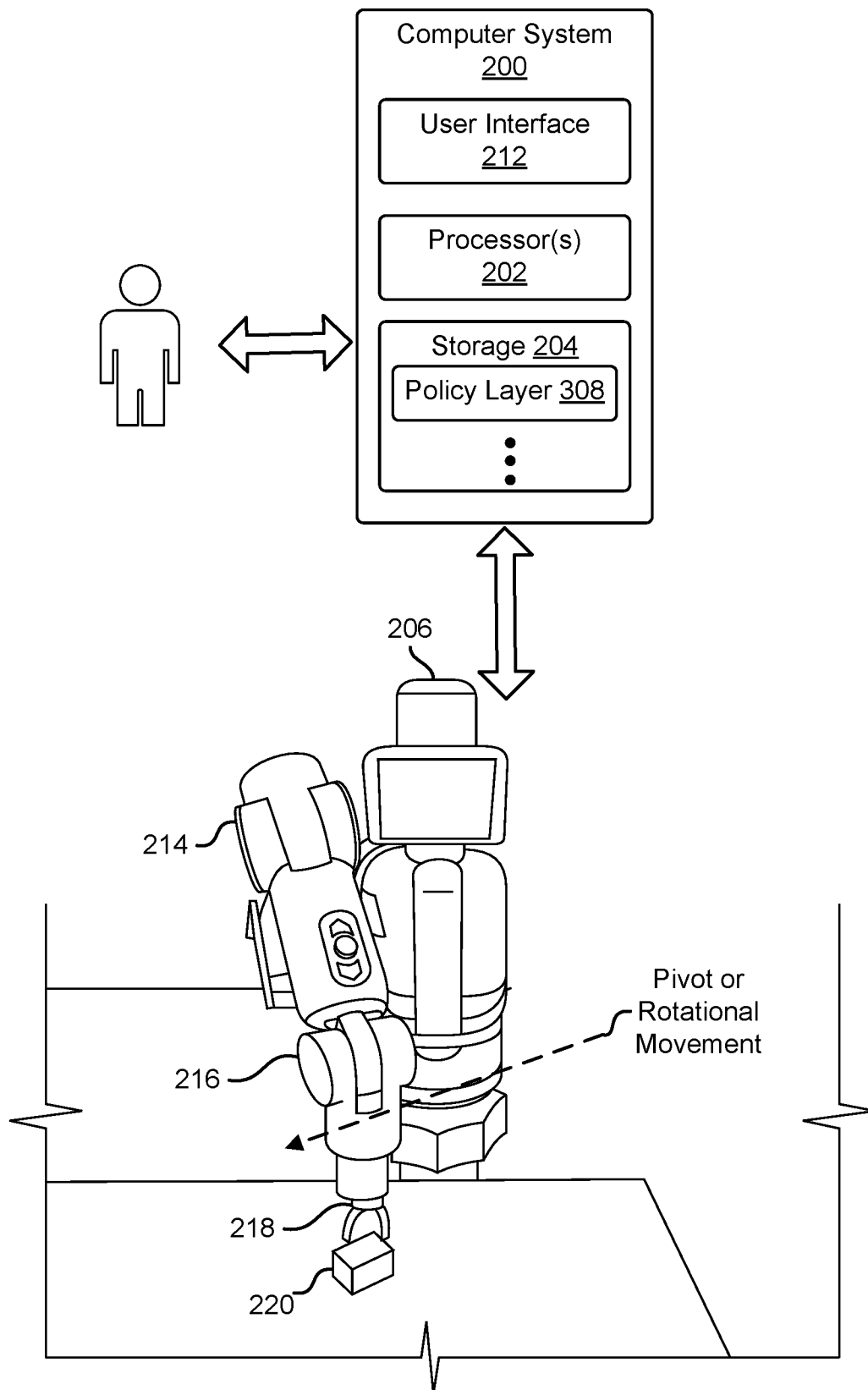
22. The computer-implemented method according to claim 19, wherein the first policy layer comprises at least a first nonlinear second-order differential equation and the second policy layer comprises at least a second nonlinear second-order differential equation, the first nonlinear second-order differential equation unbiased to cause the first motion to
30 come to rest and the second nonlinear second-order differential equation unbiased to cause the second motion to come to rest.

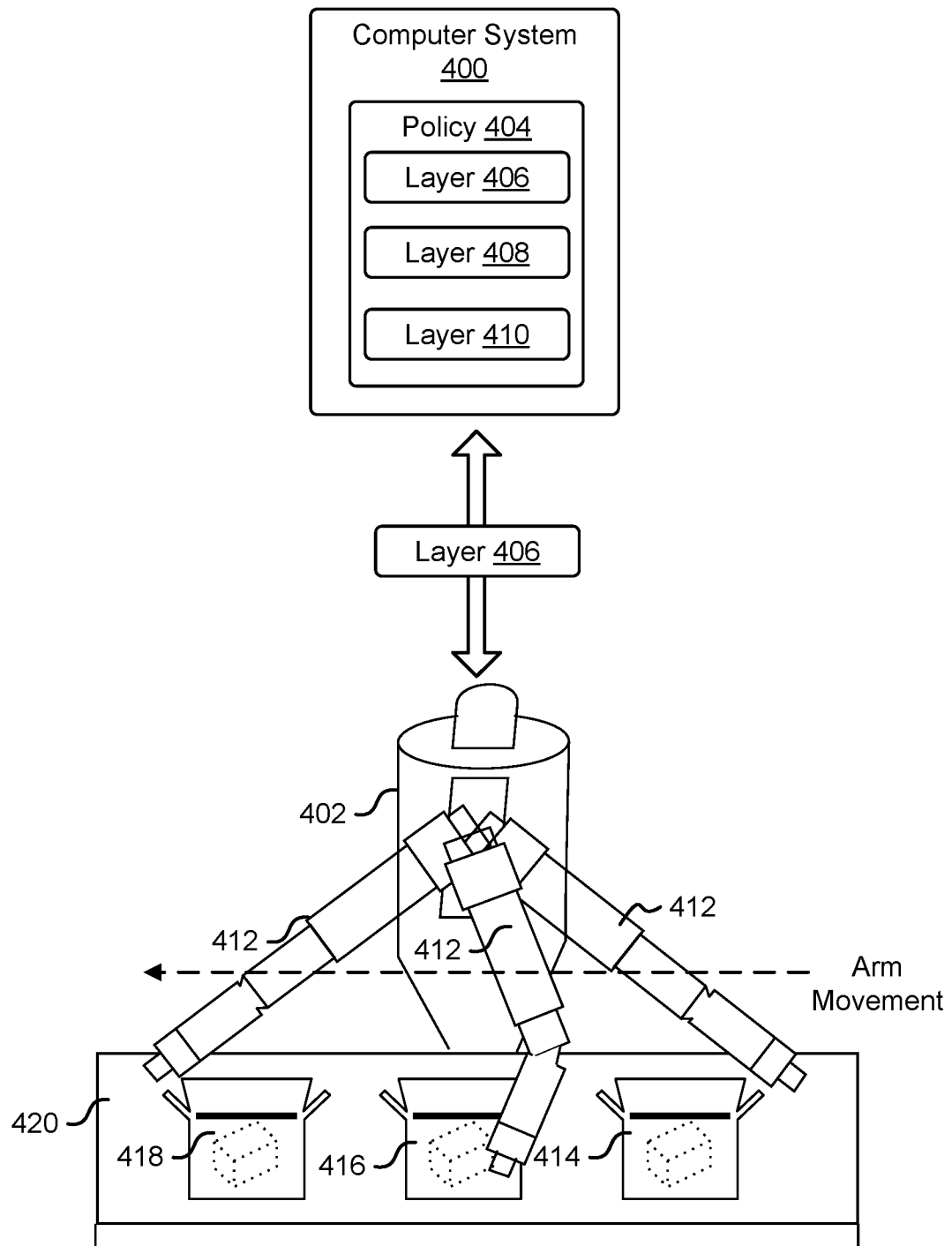
policy layer is a first geometric fabric comprising a nonlinear second-order differential equation and the second policy layer is a second geometric fabric comprising another second-order differential equation.

- 5 24. The computer-implemented method according to claim 19, wherein the first policy layer comprises at least first data comprising a one or more limits associated with a joint of the machine and the second policy layer comprises at least second data comprising a target position to be reached by the machine, the target position located in an Euclidean space associated with the area in which a portion of the machine is to operate.

**FIG. 1**

**FIG. 2**

**FIG. 3**

**FIG. 4**

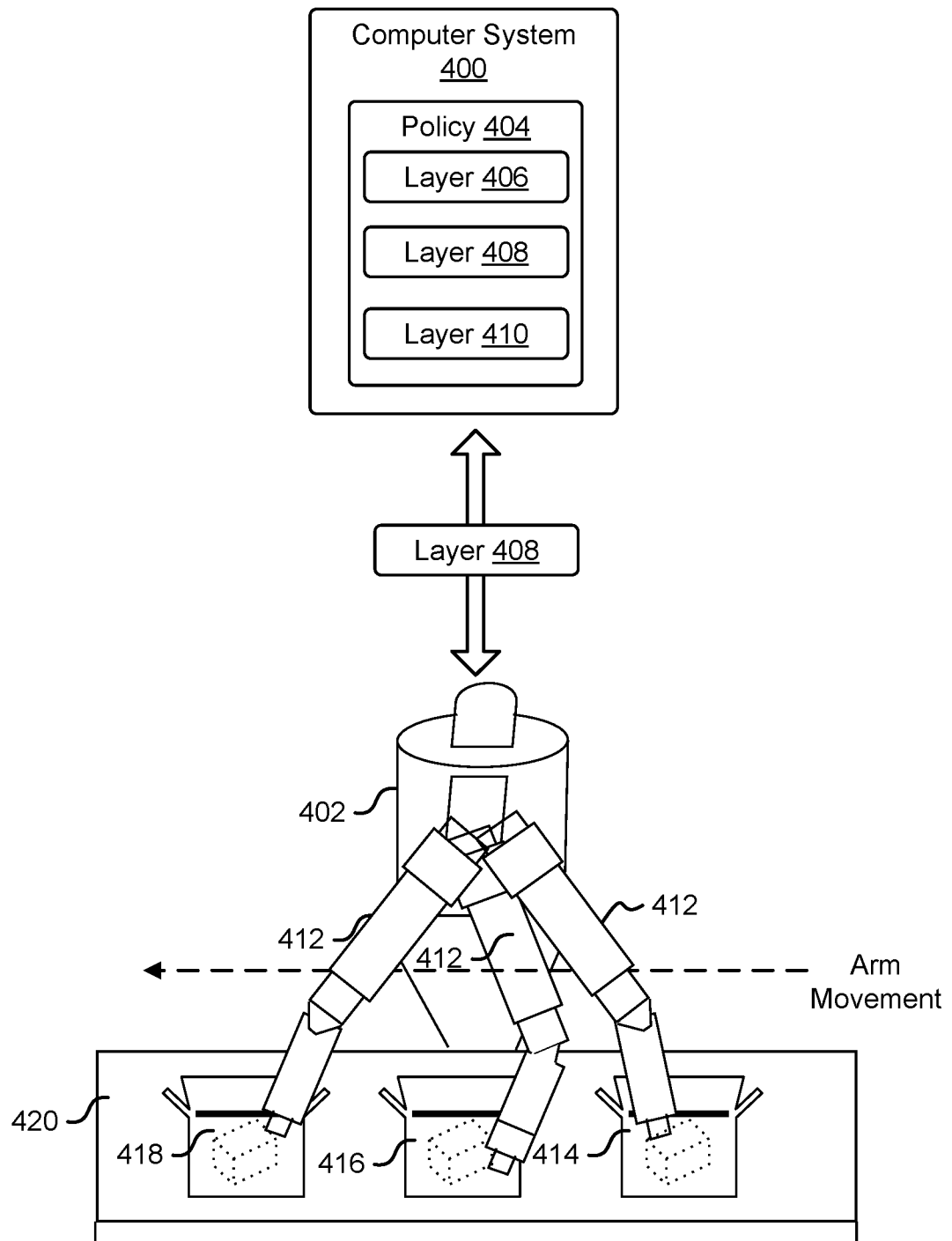
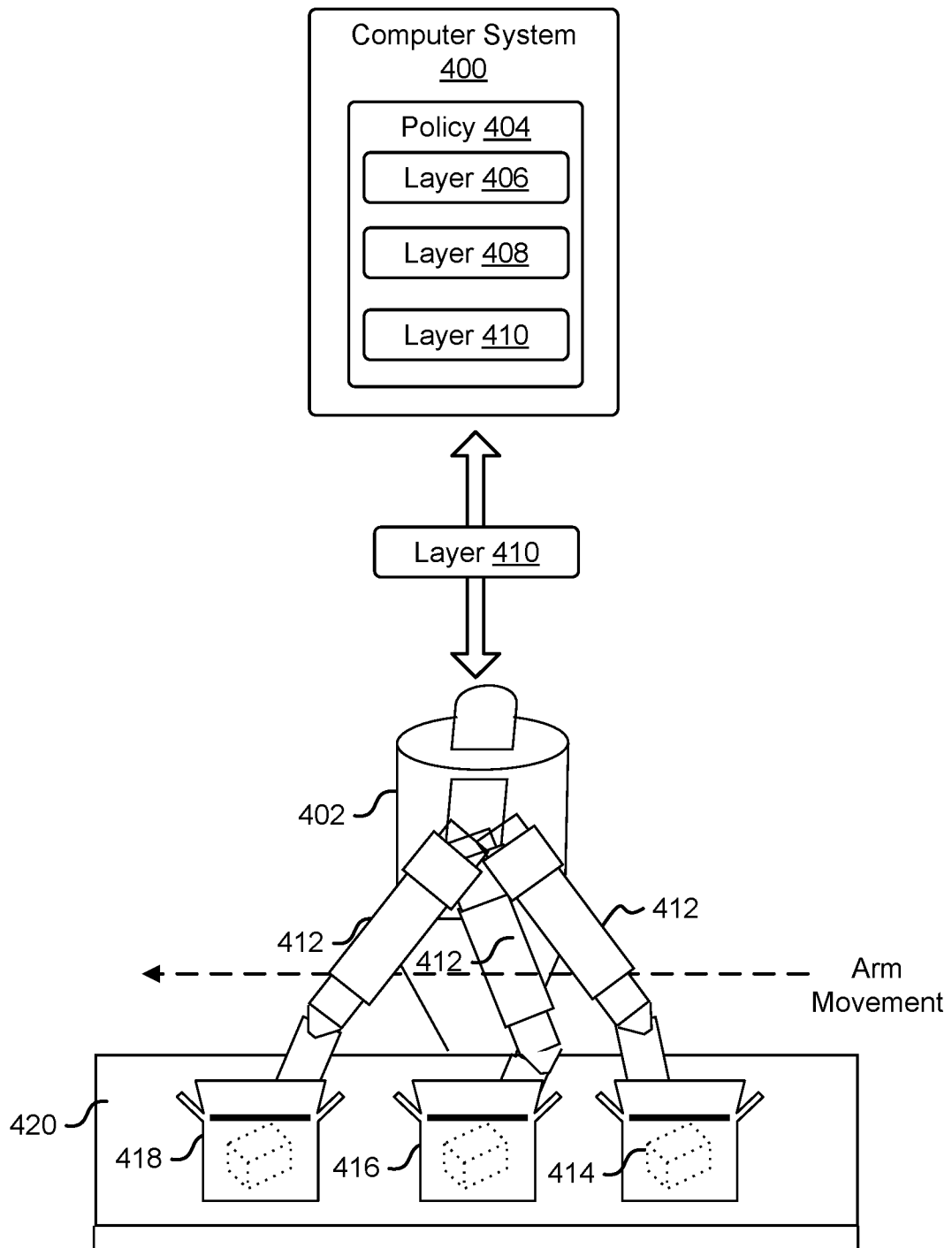
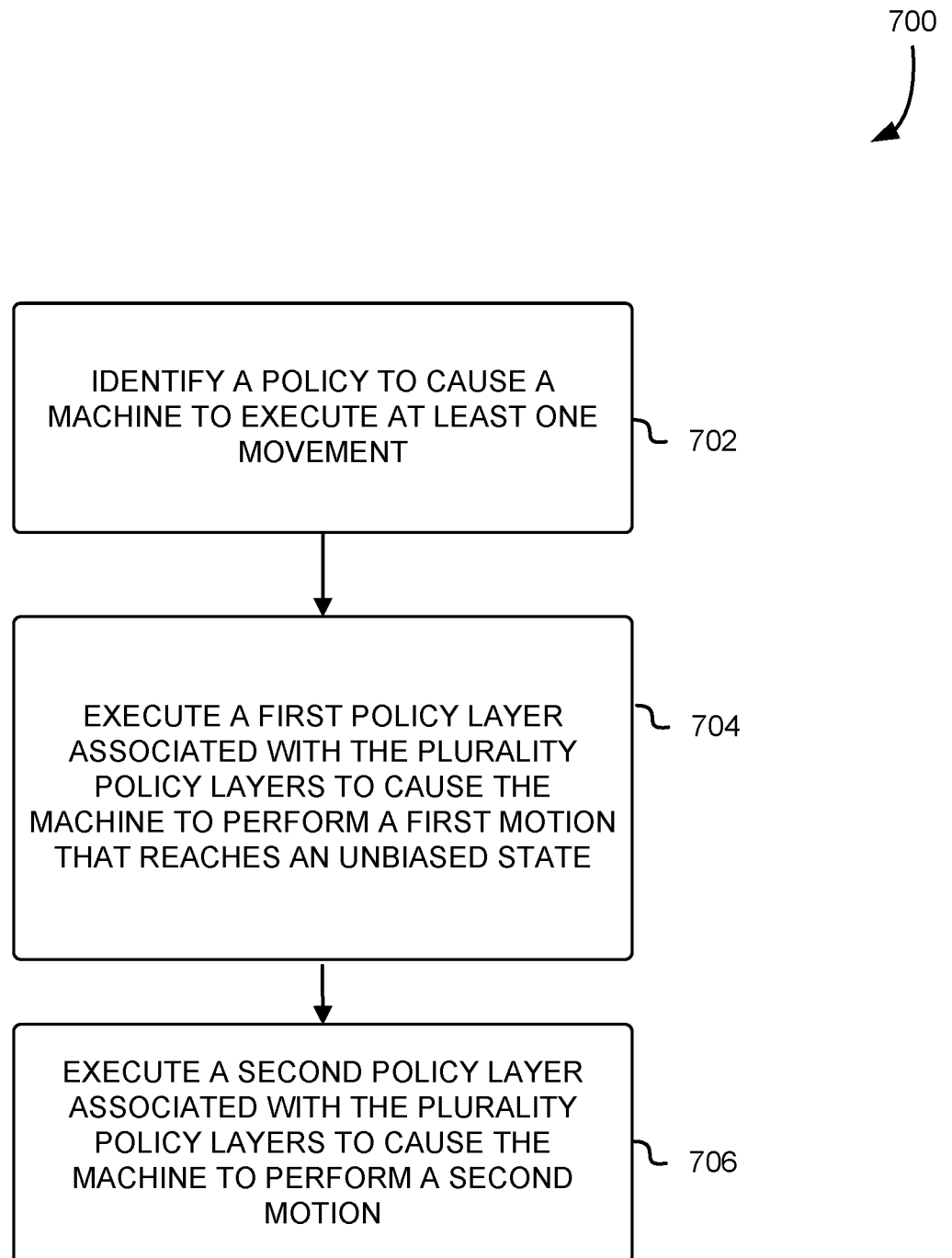
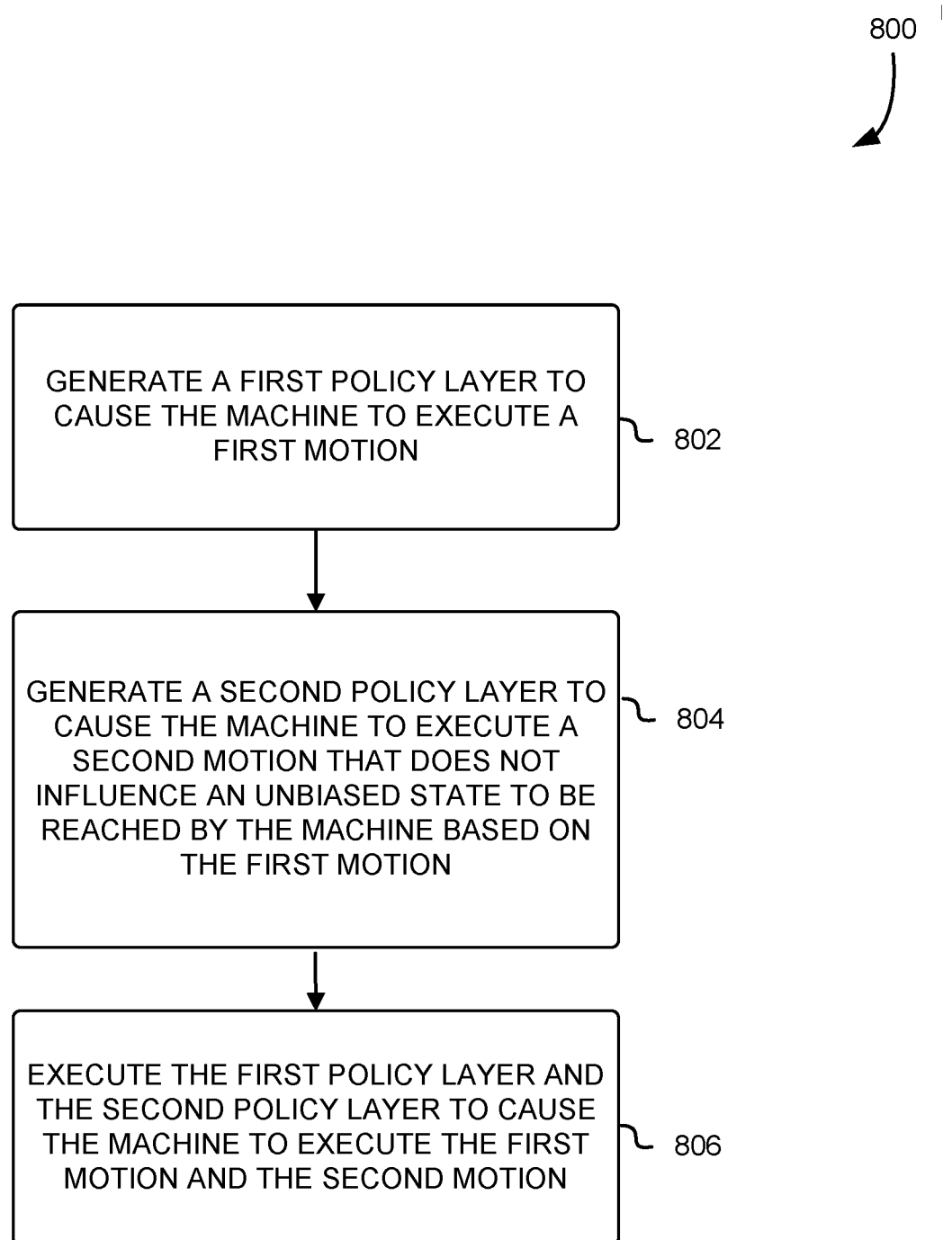
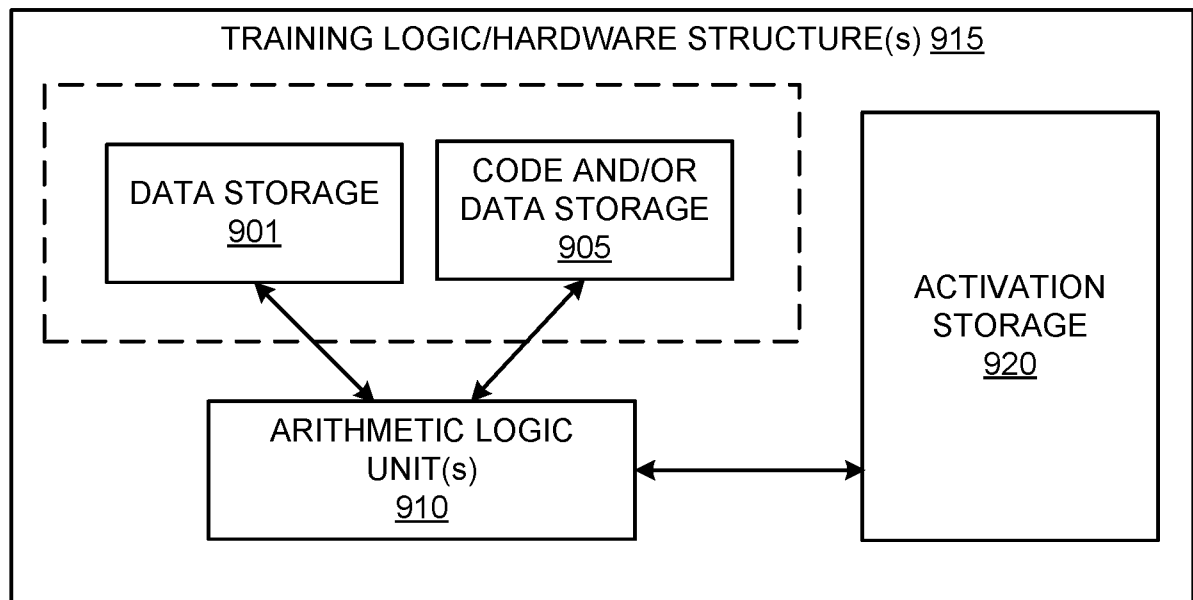
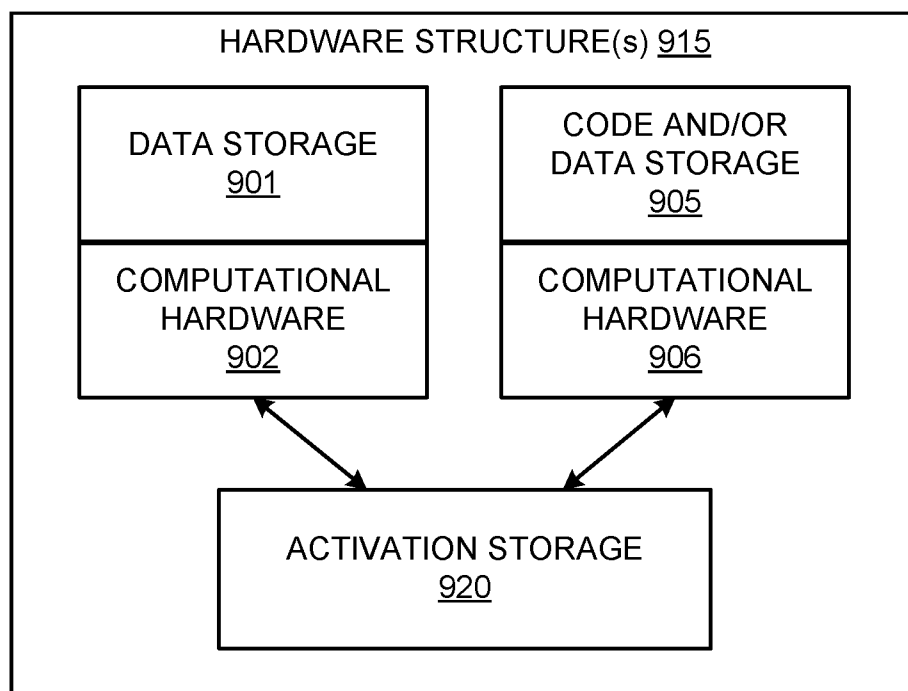


FIG. 5

**FIG. 6**

**FIG. 7**

**FIG. 8**

**FIG. 9A****FIG. 9B**

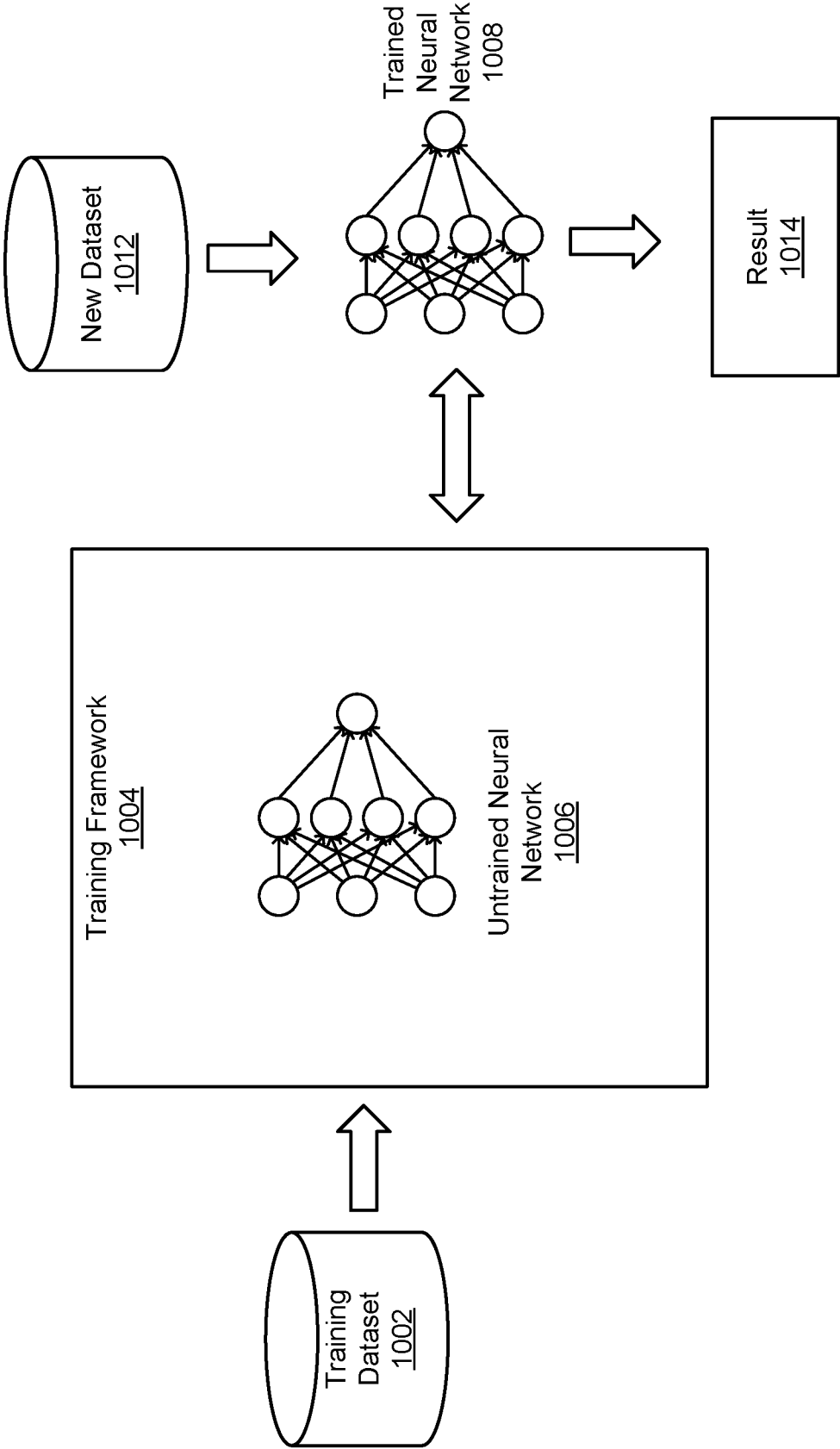


FIG. 10

DATA CENTER
1100

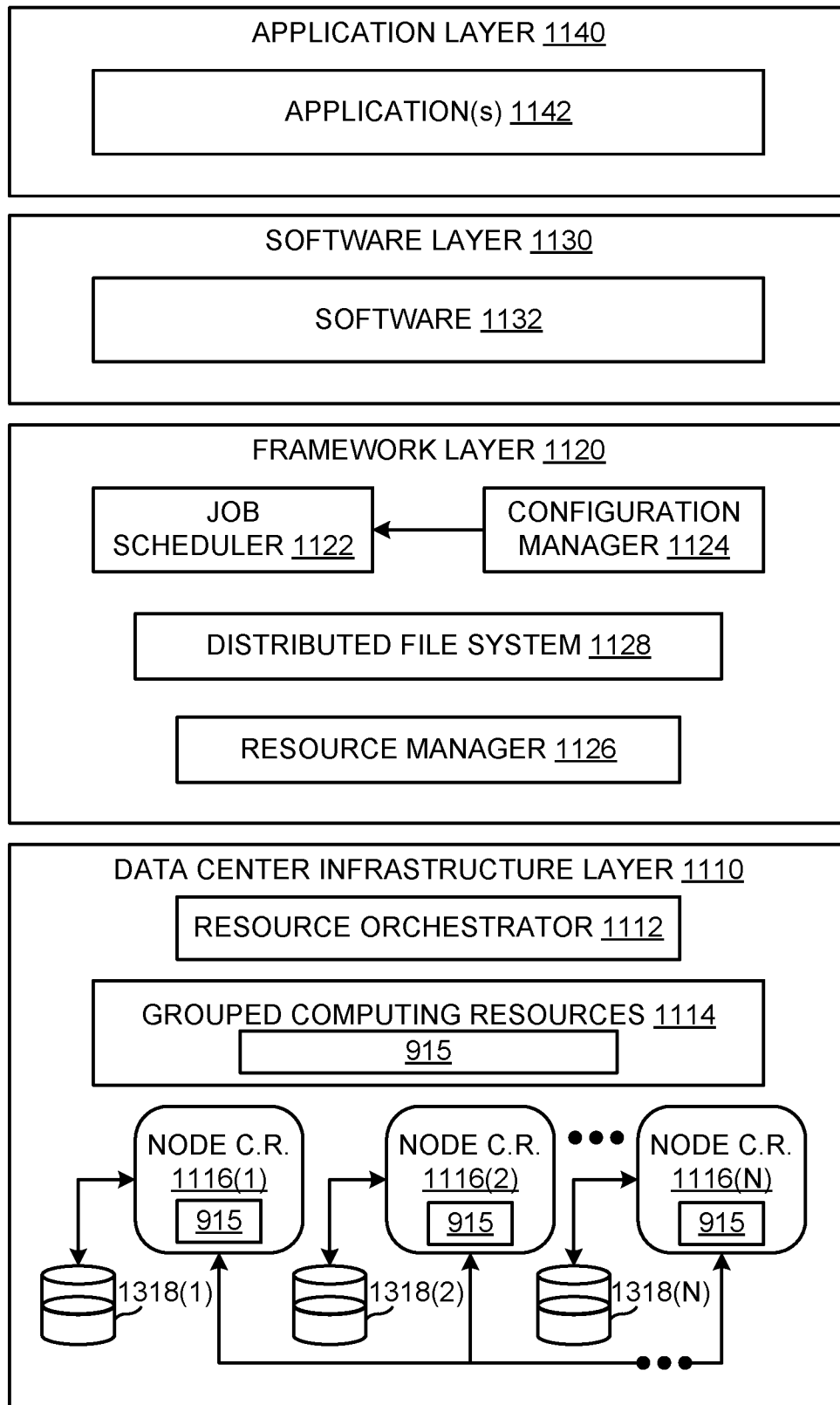


FIG. 11

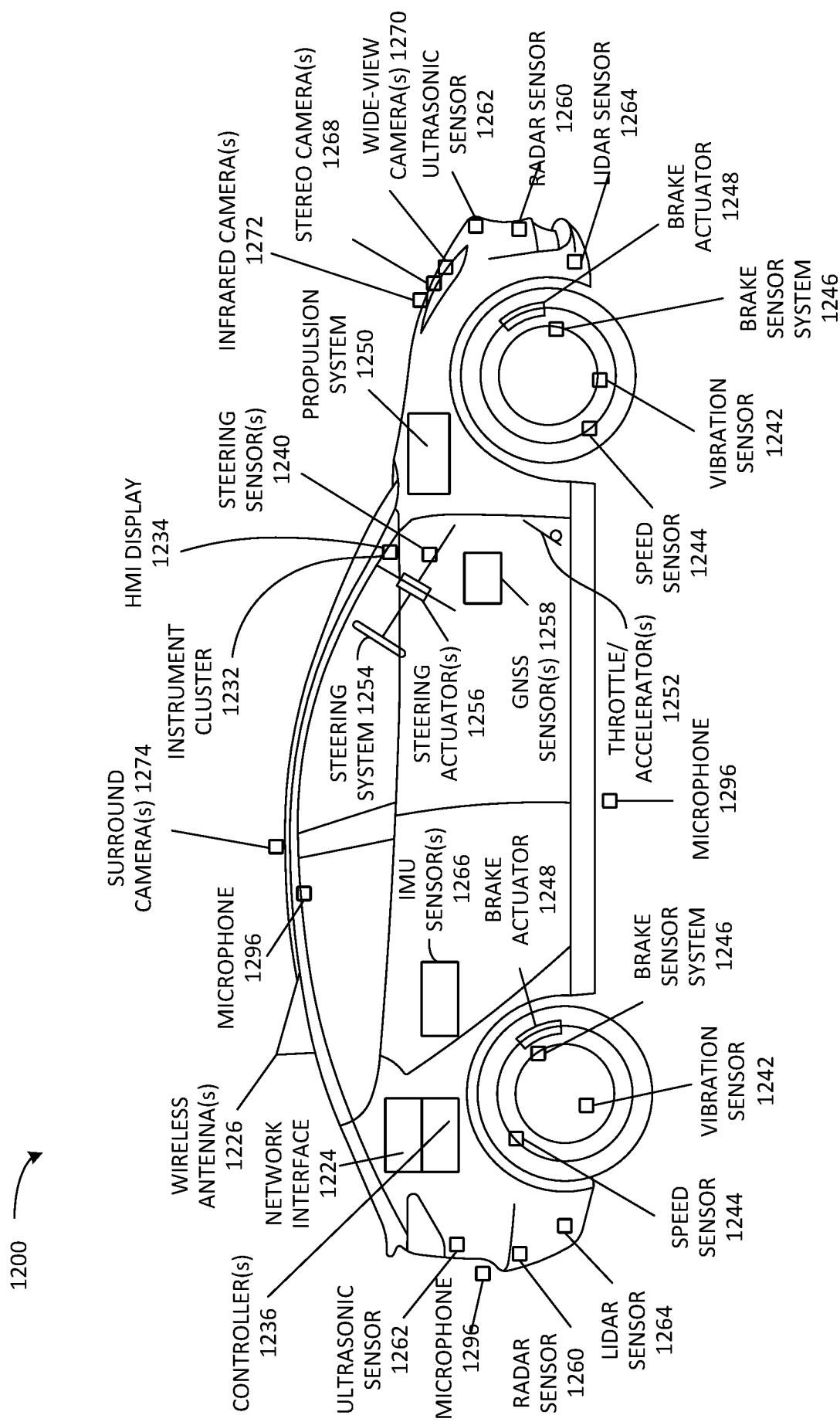


FIG. 12 A

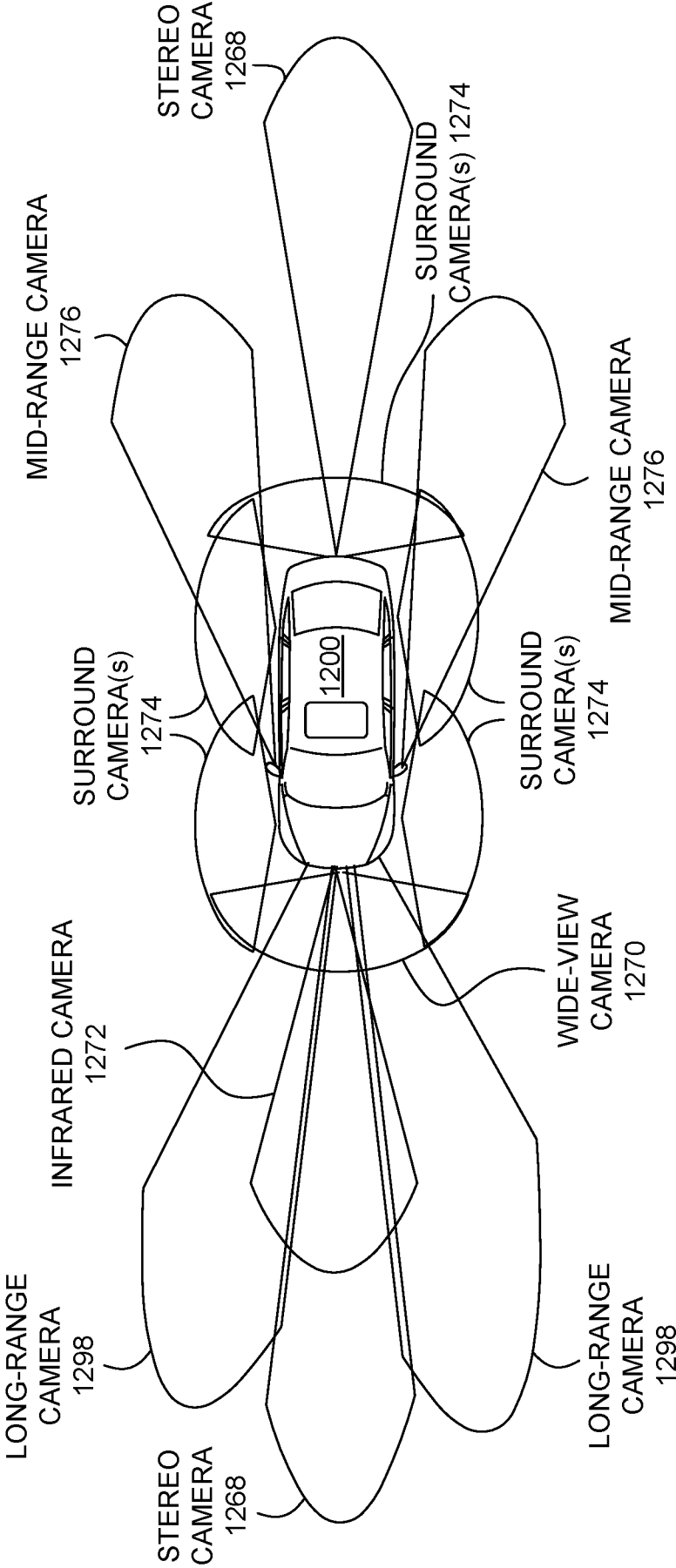
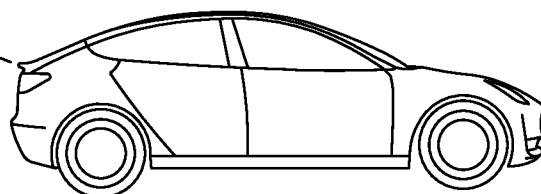
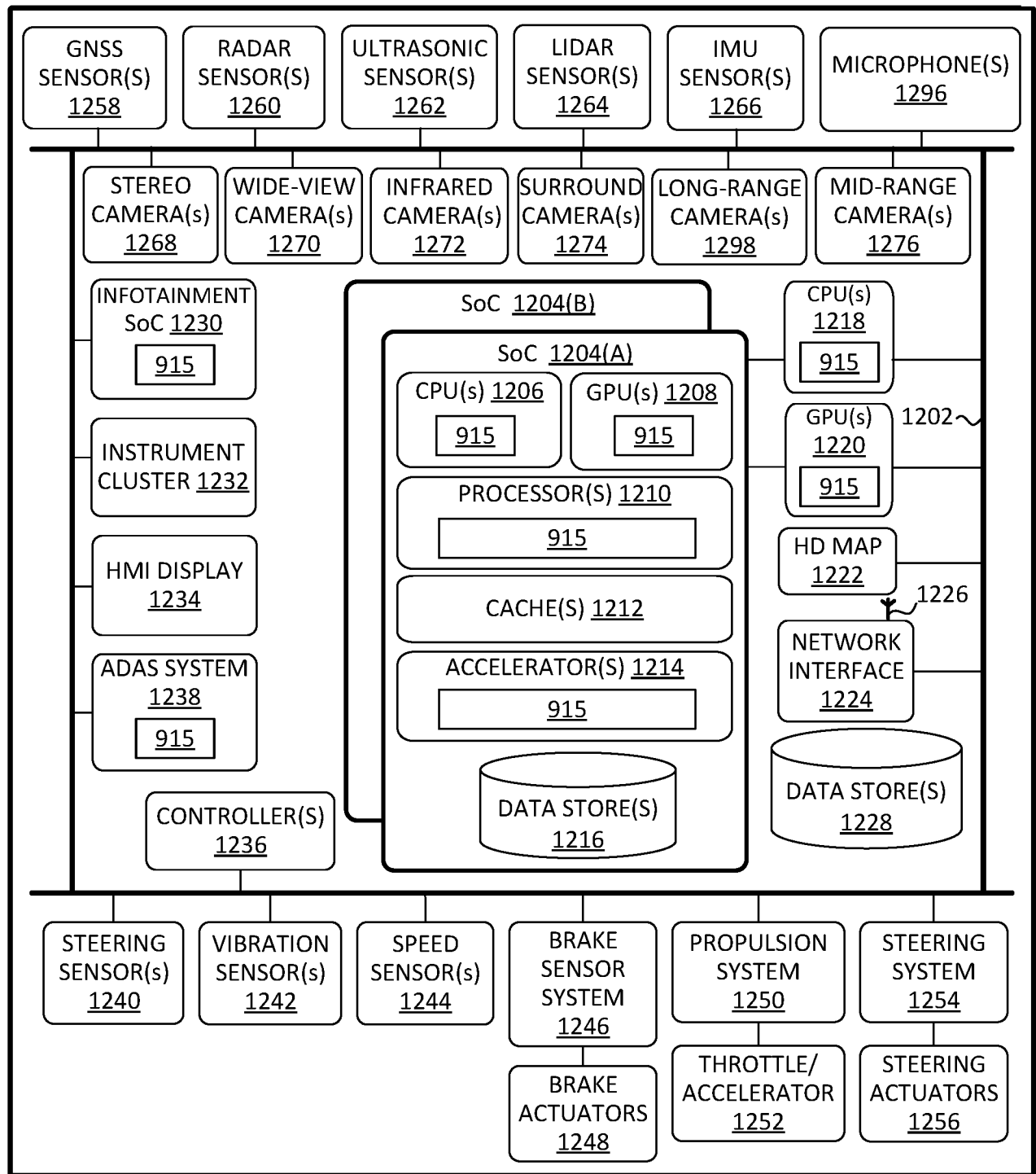
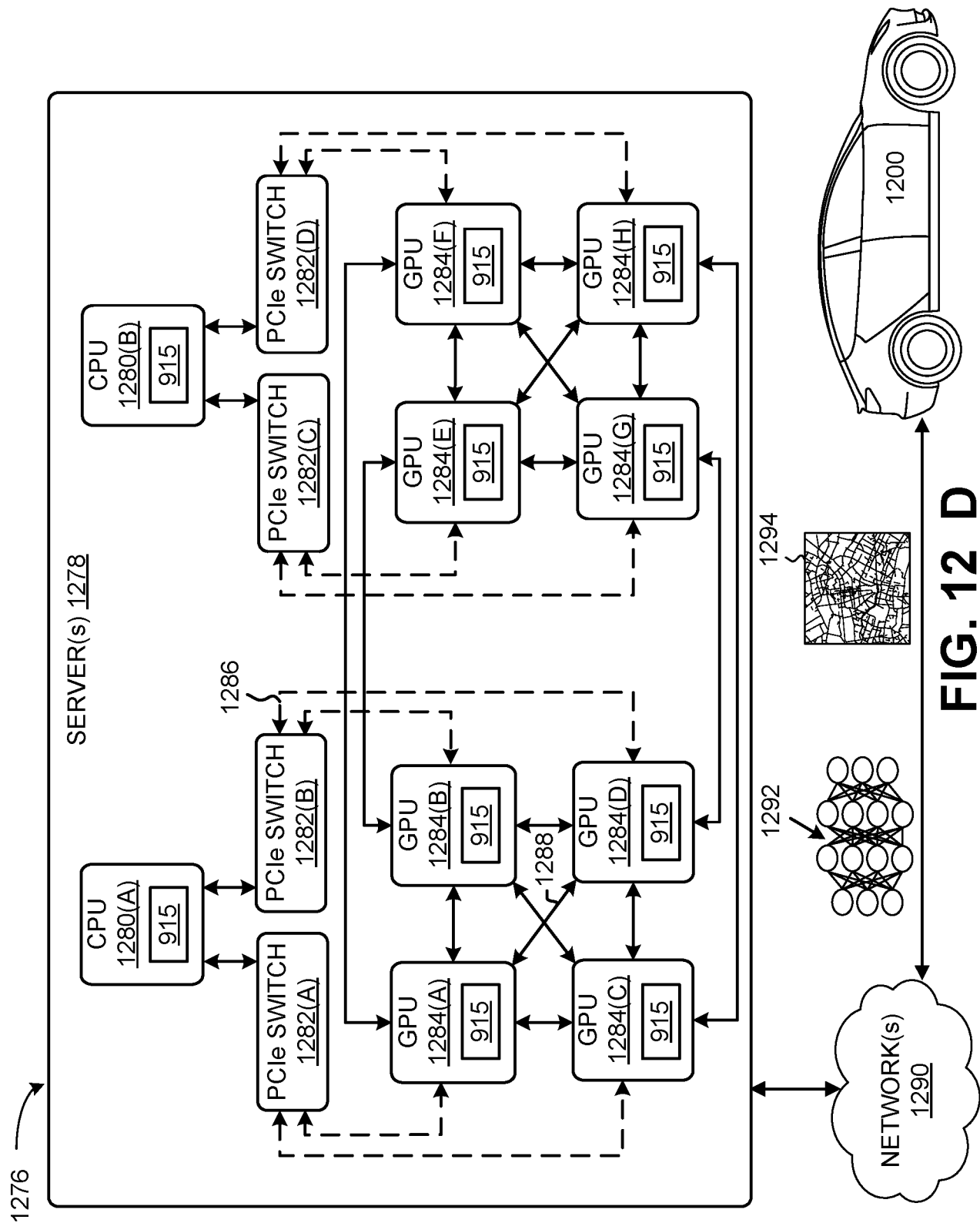


FIG. 12 B

**FIG. 12 C**



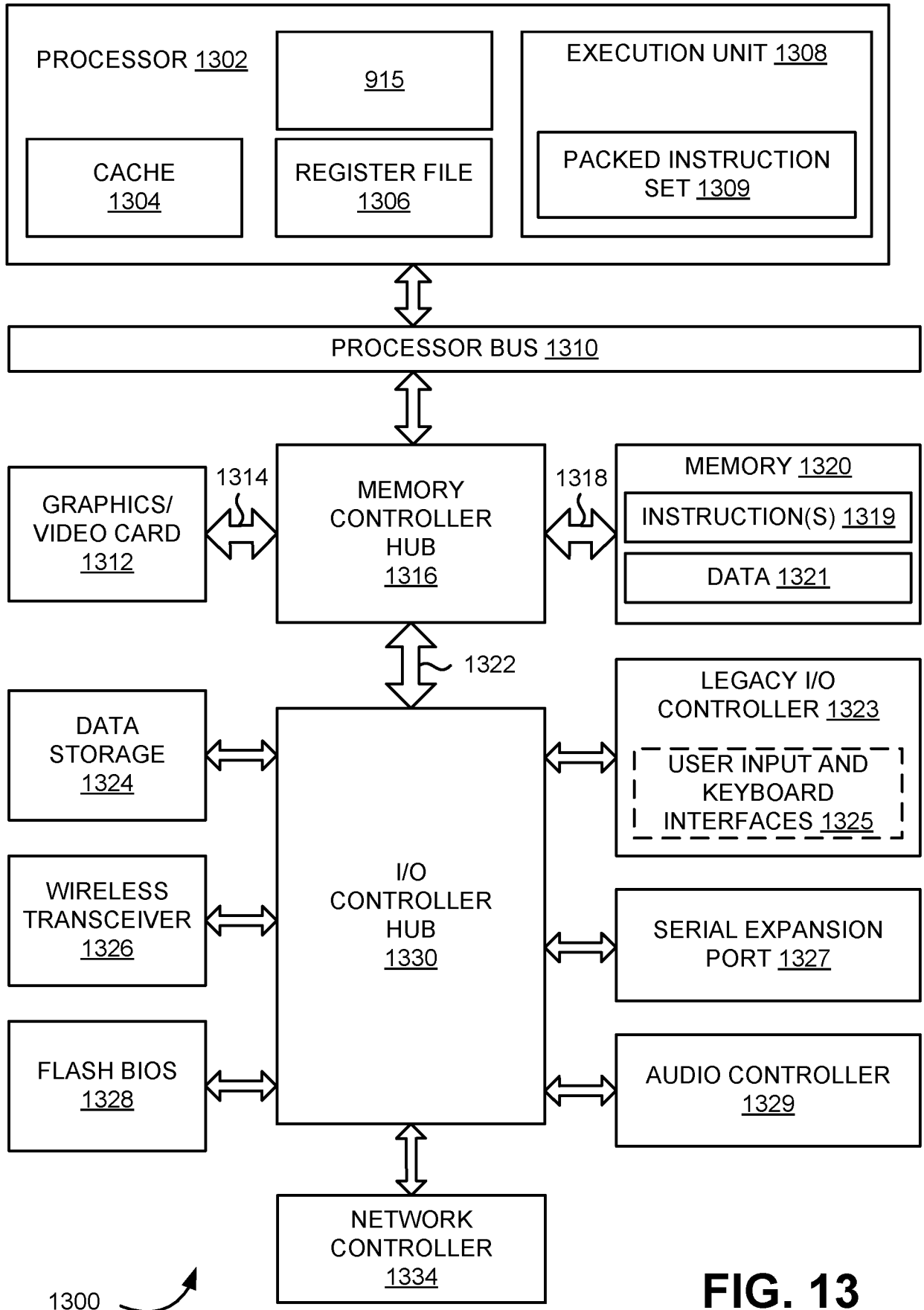
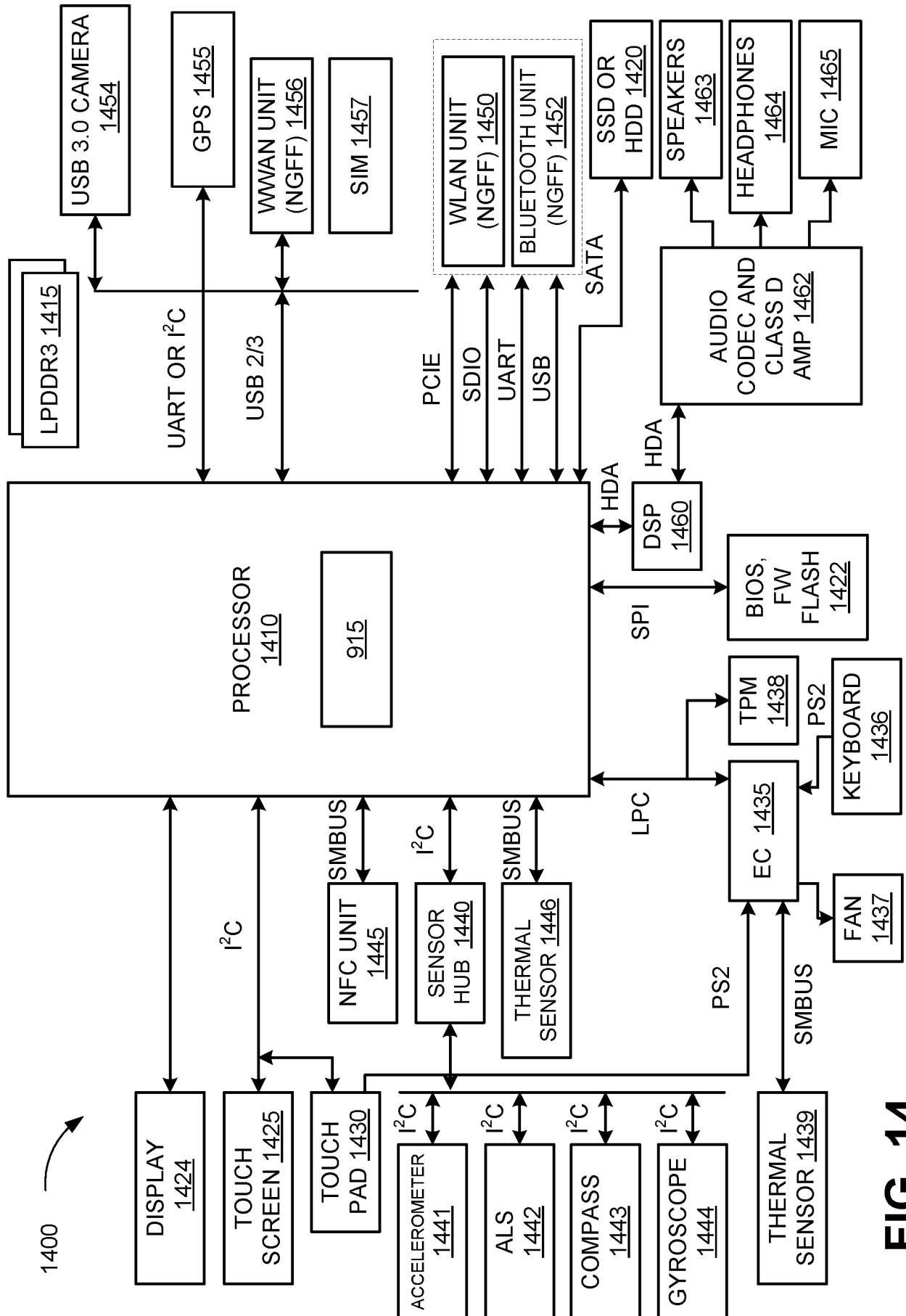
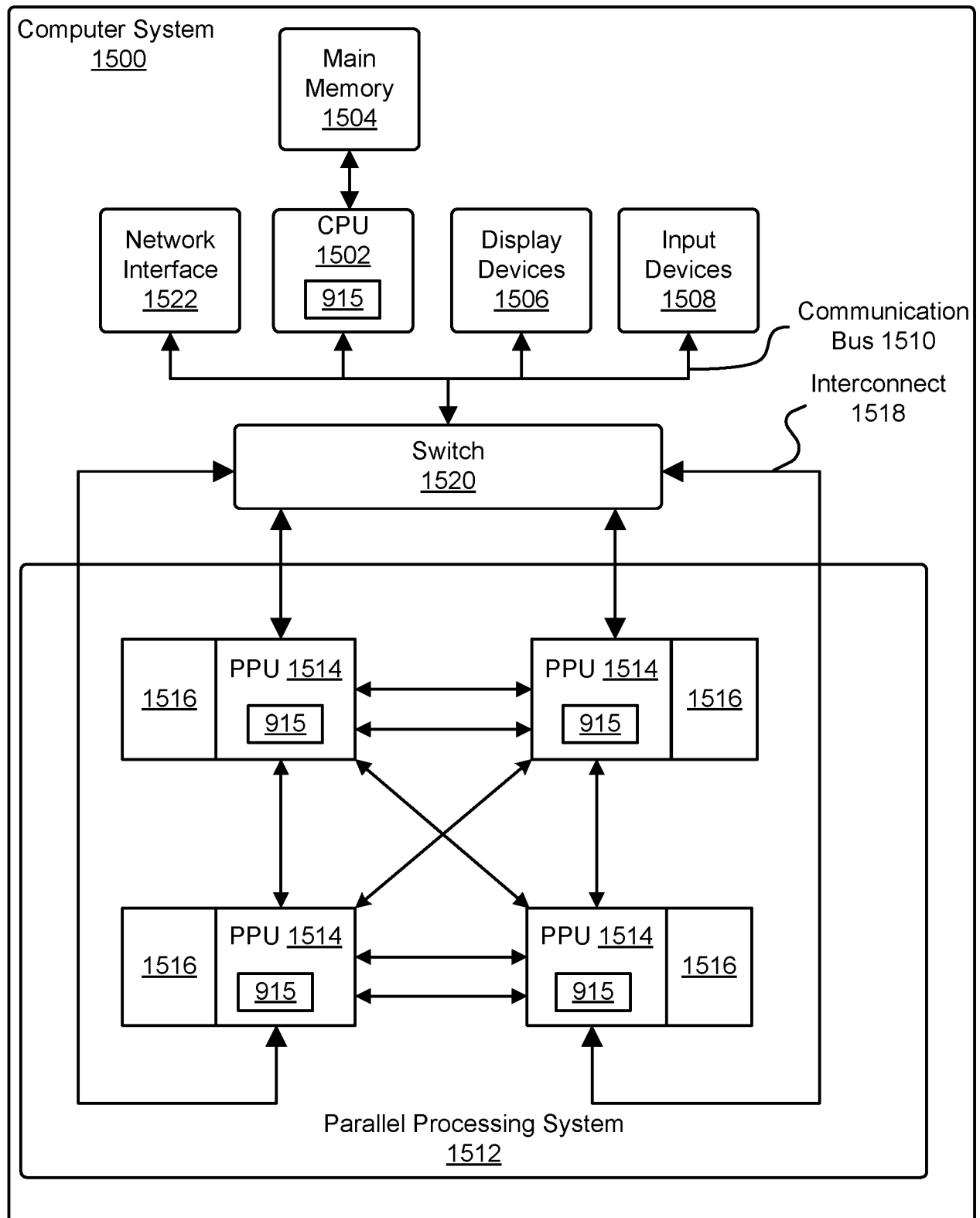


FIG. 13

**FIG. 14**

**FIG. 15**

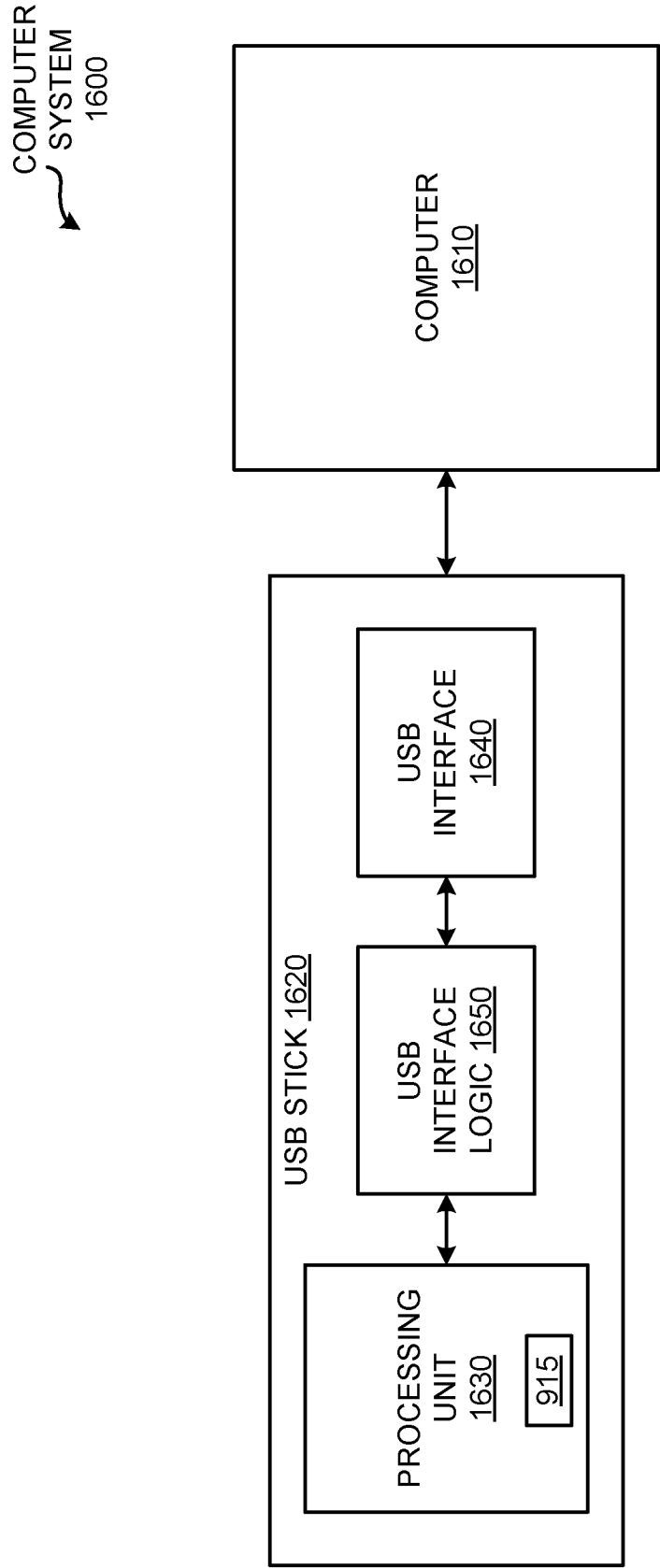


FIG. 16

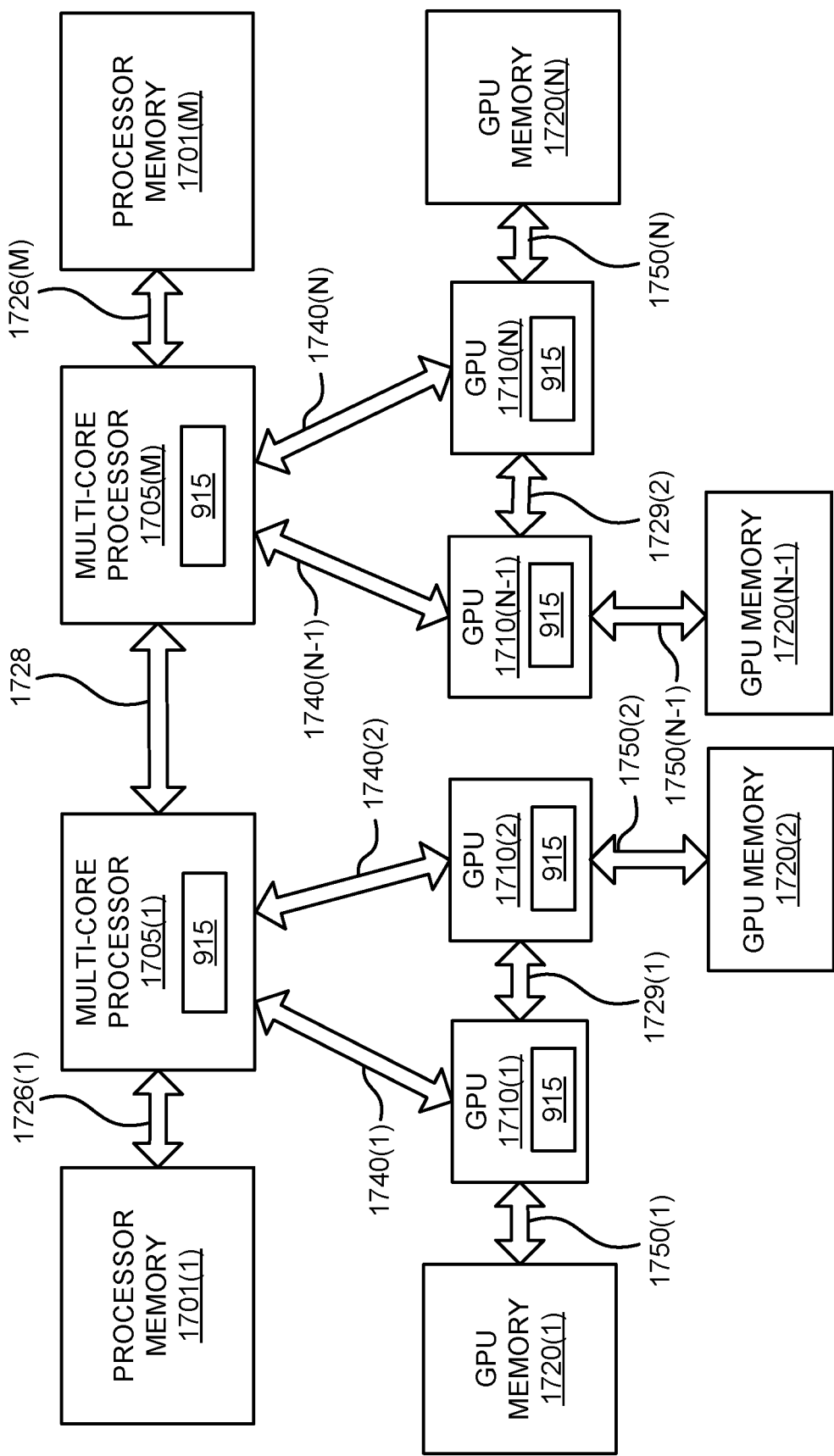


FIG. 17 A

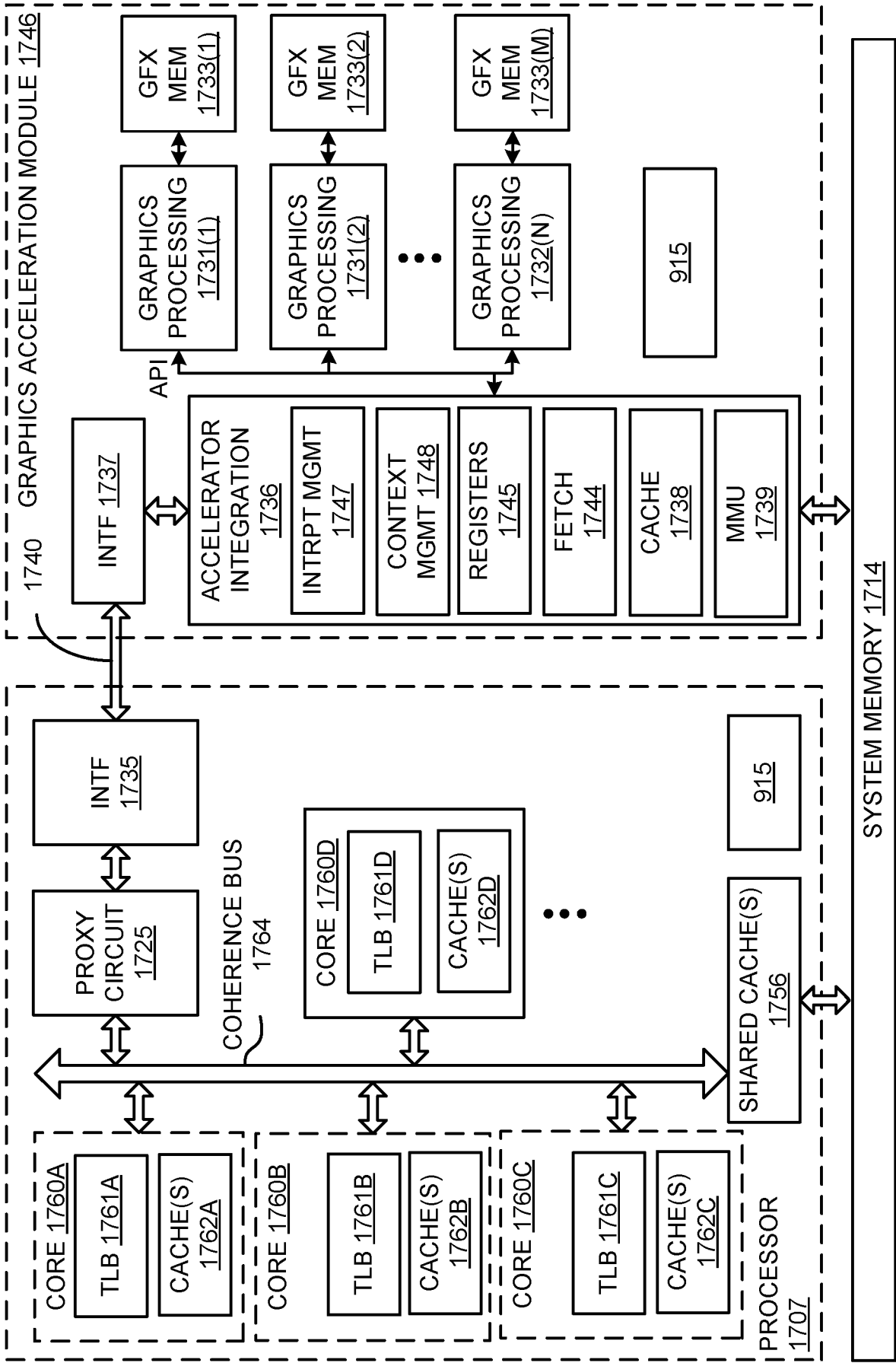


FIG. 17 B

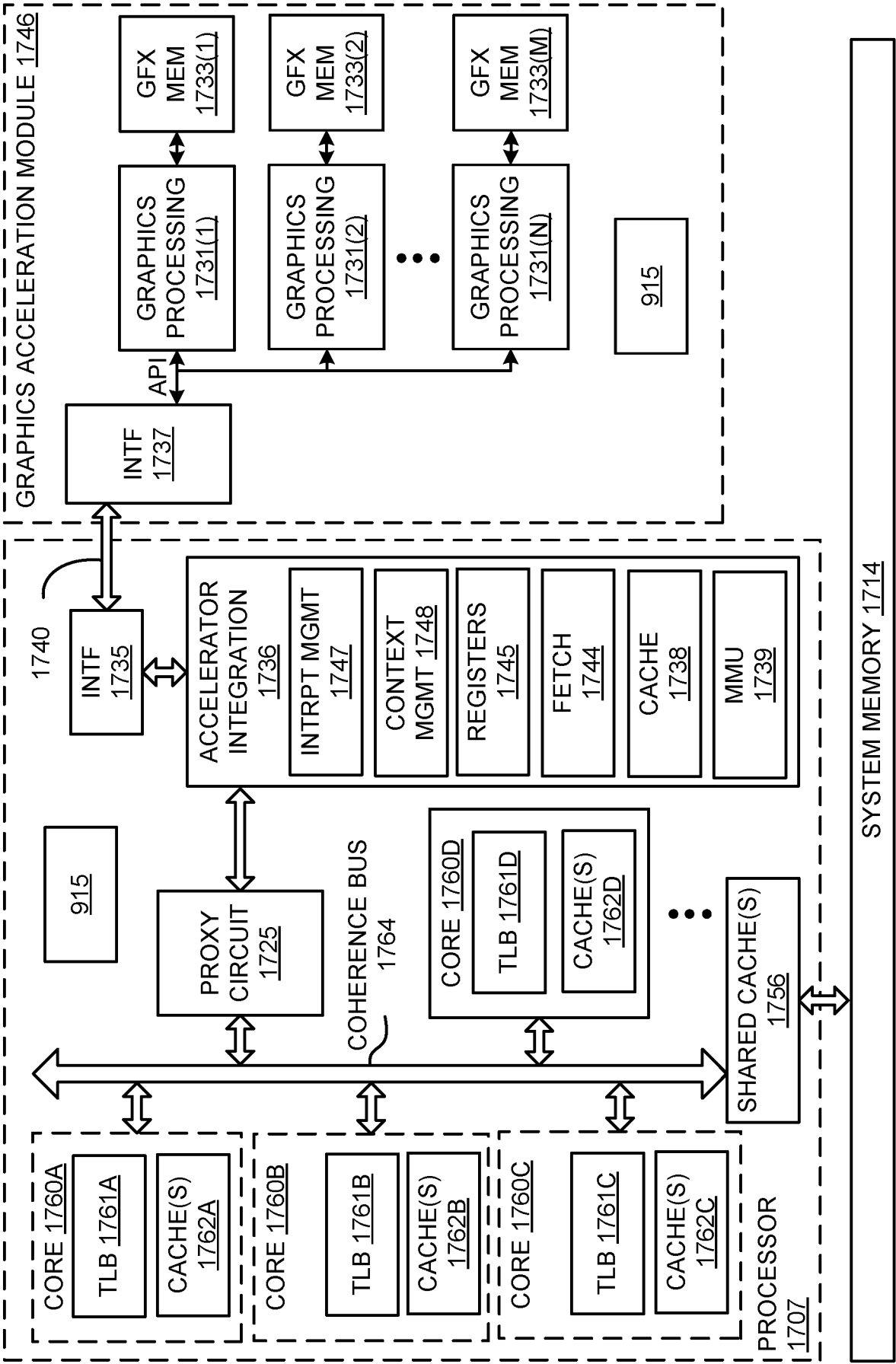


FIG. 17 C

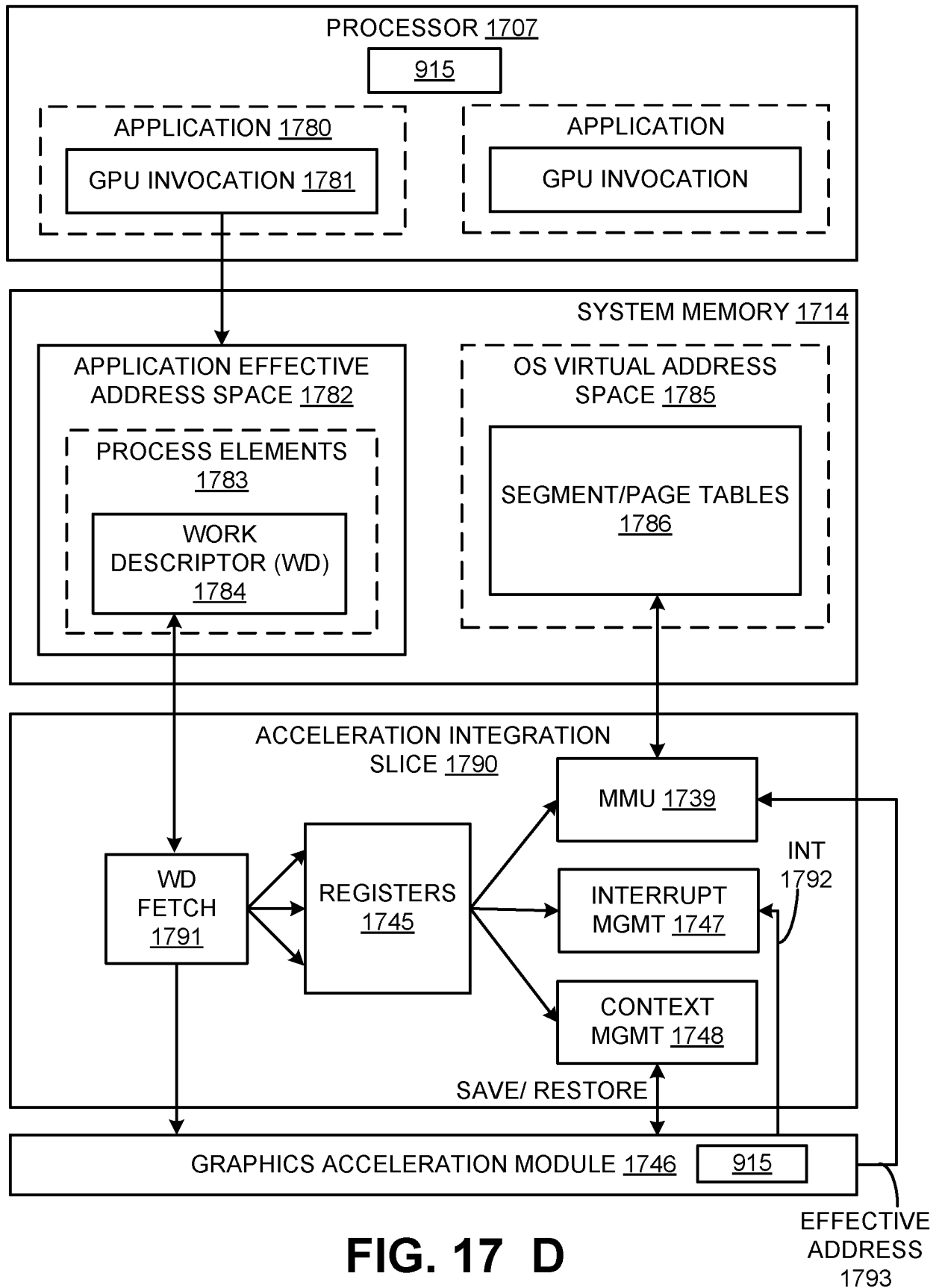


FIG. 17 D

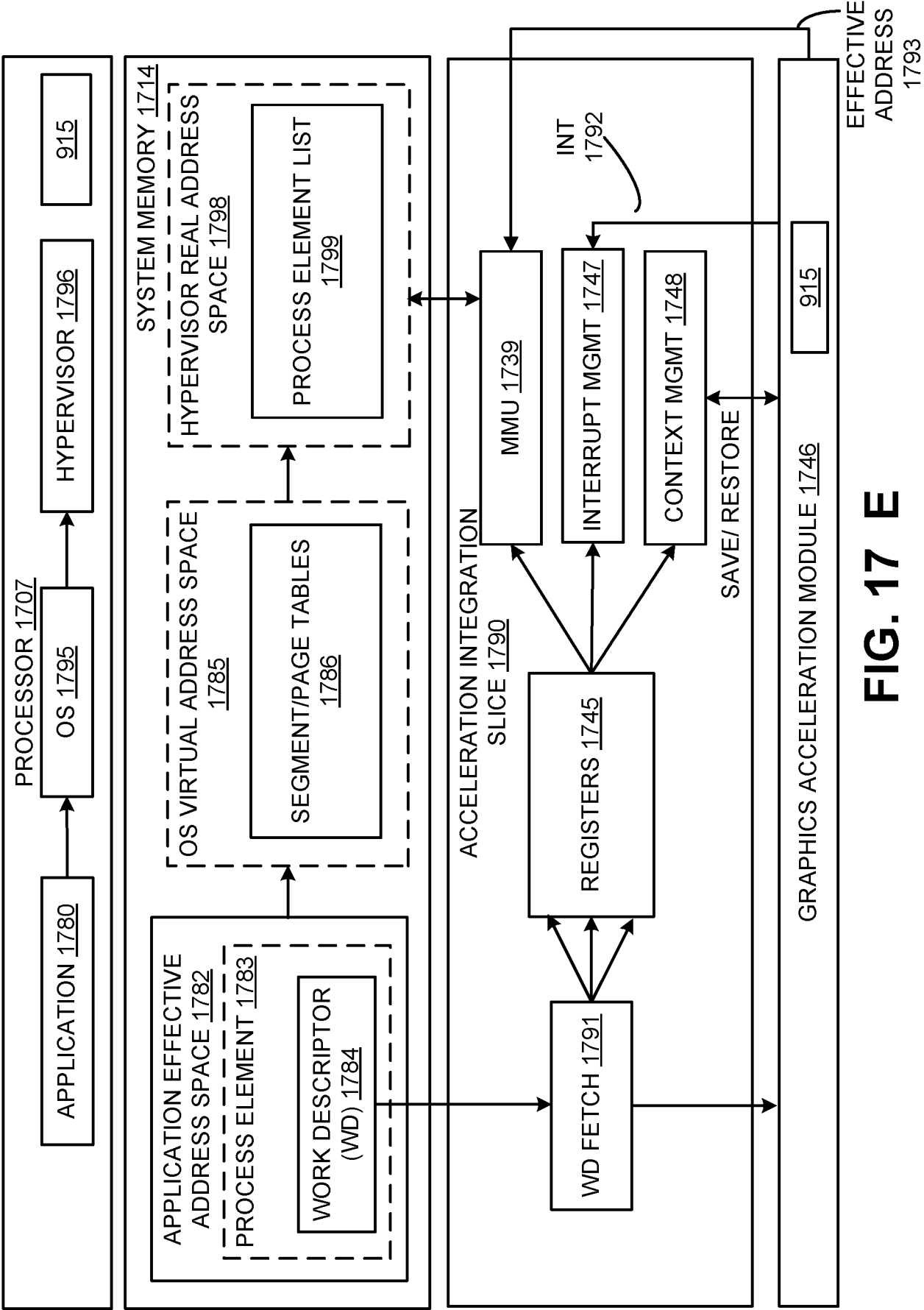


FIG. 17 E

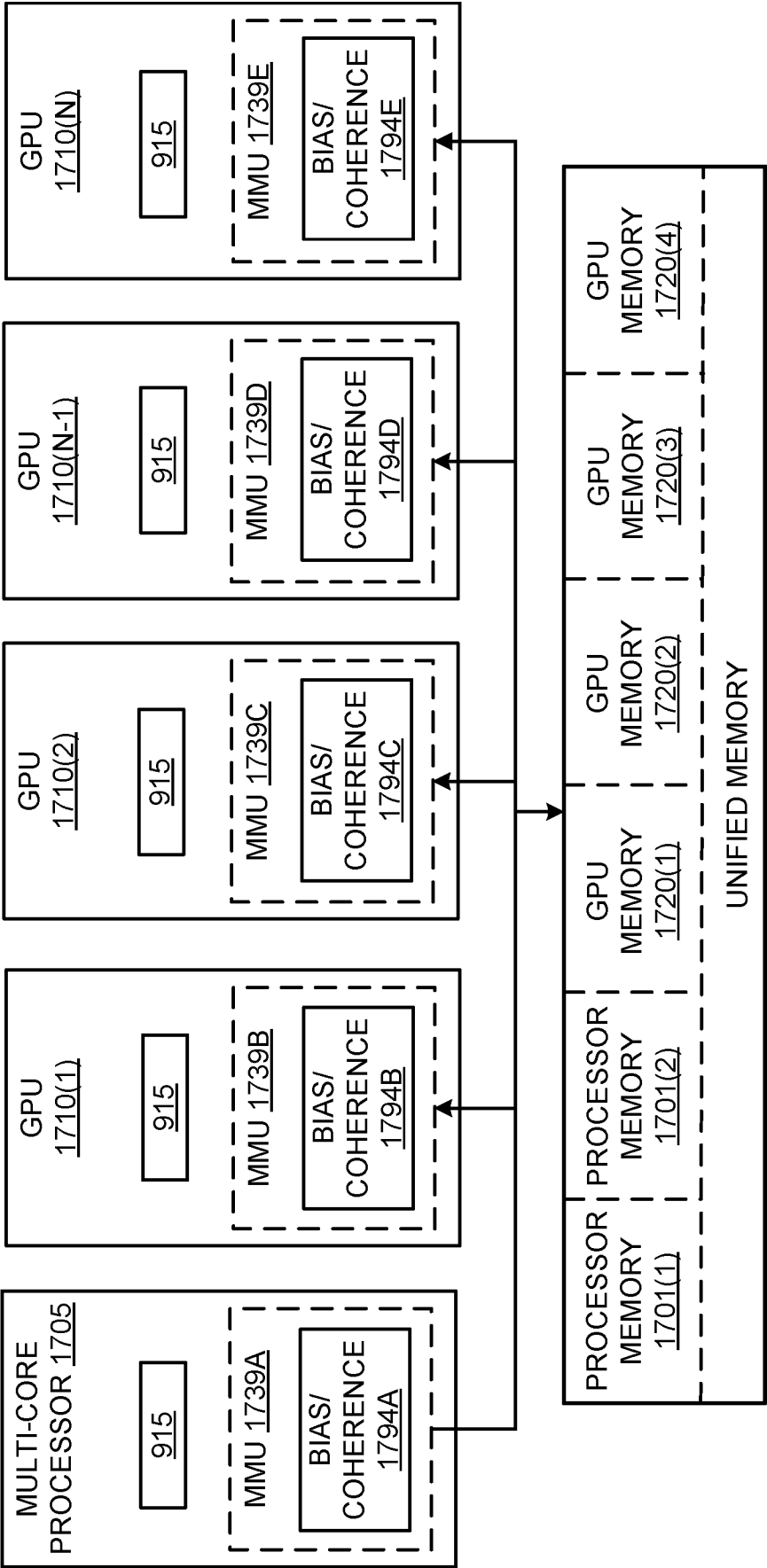


FIG. 17 F

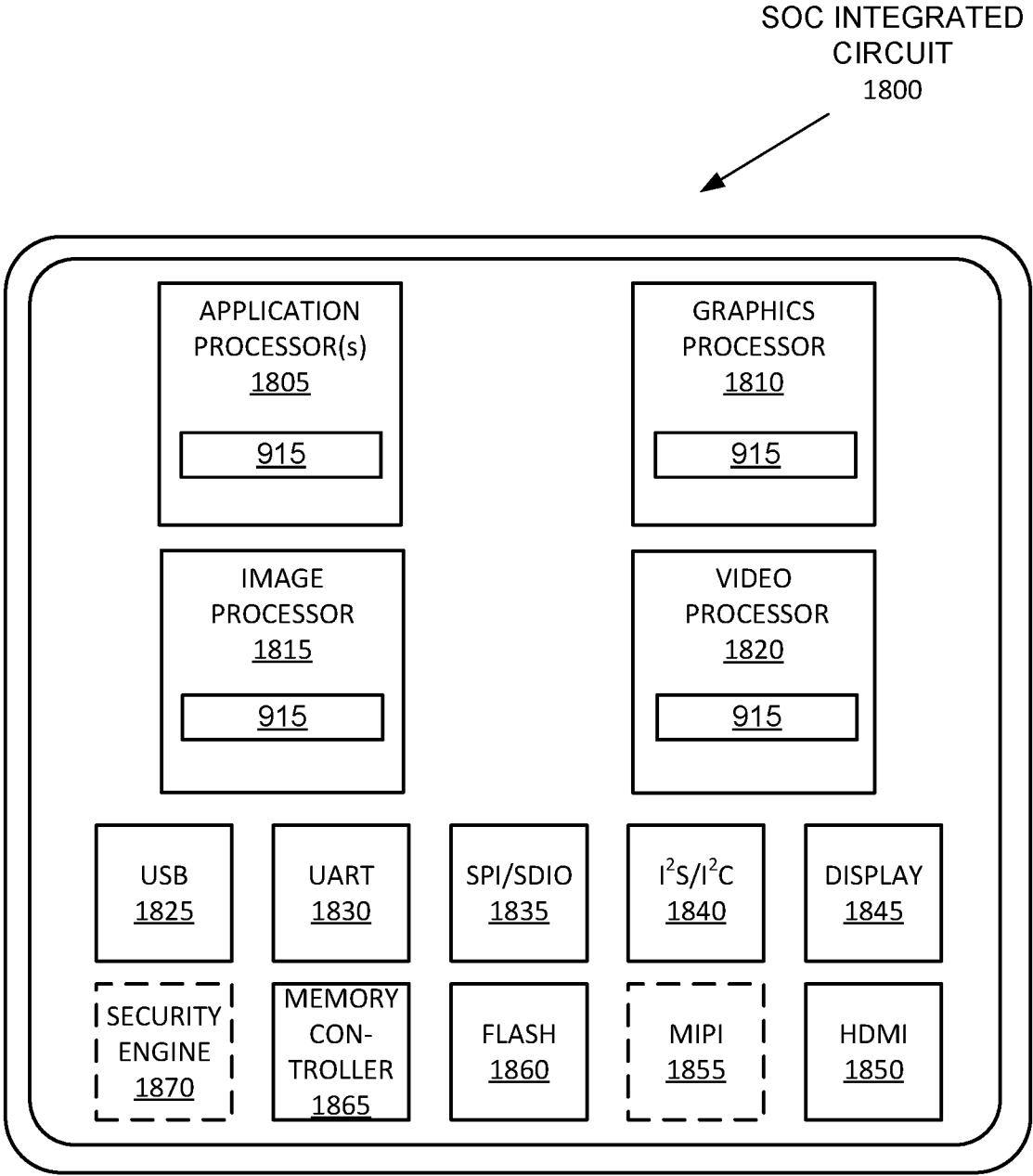


FIG. 18

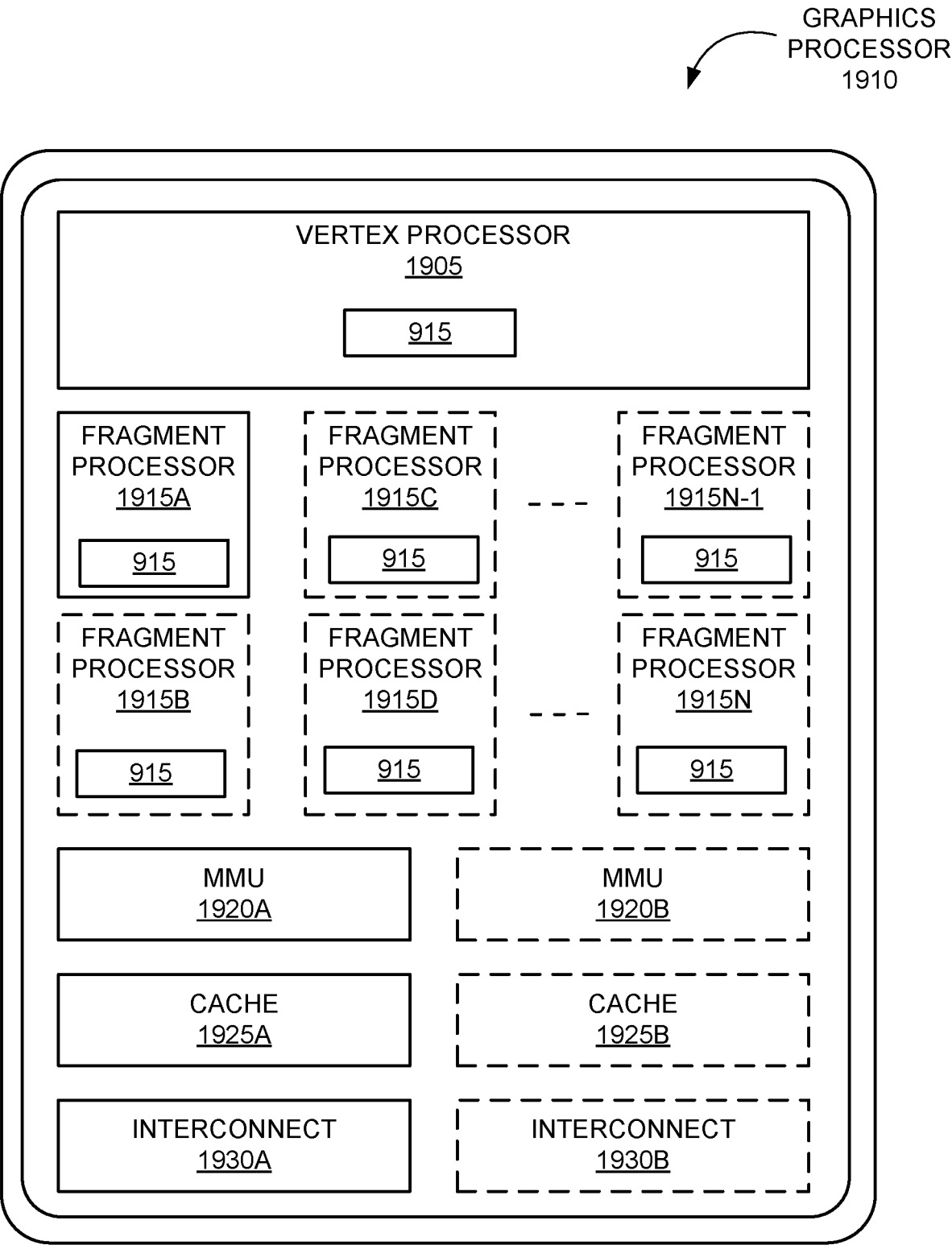


FIG. 19A

GRAPHICS
PROCESSOR
1940

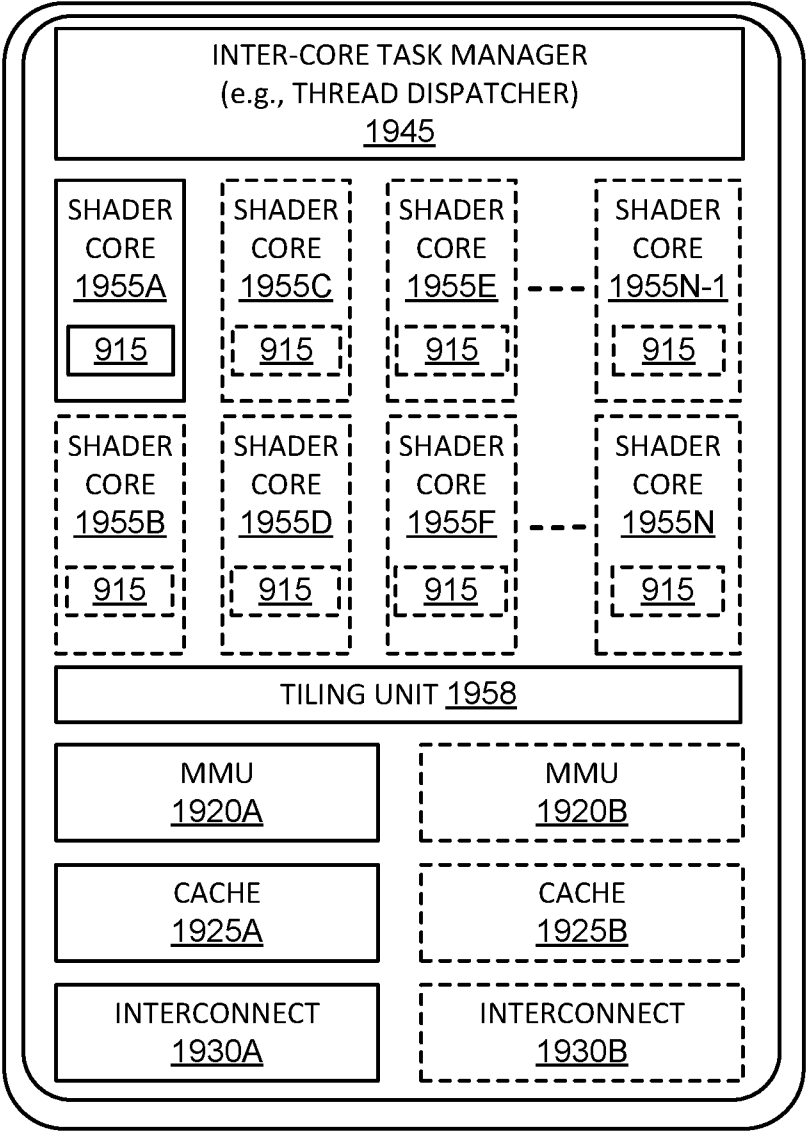
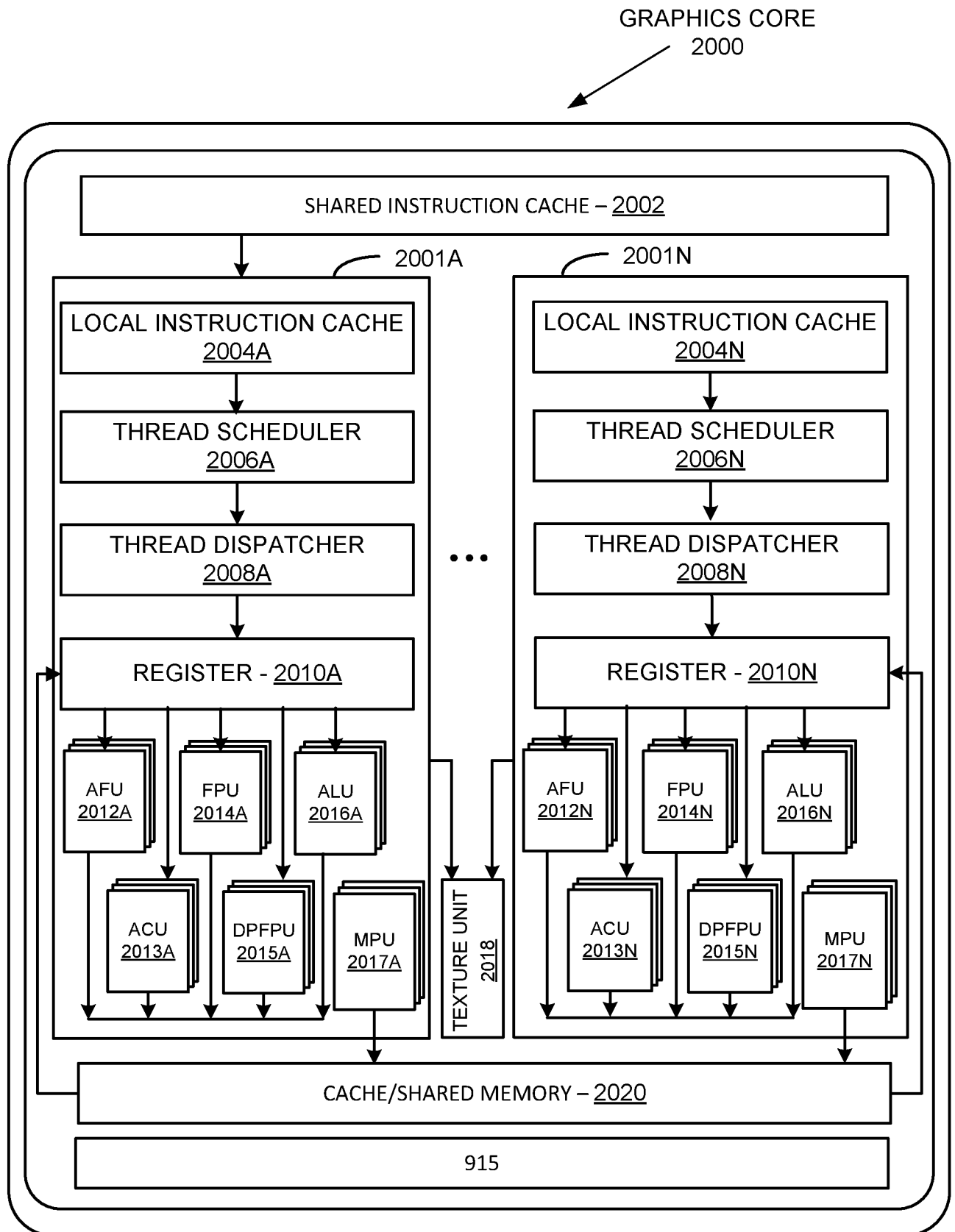


FIG. 19B

**FIG. 20A**

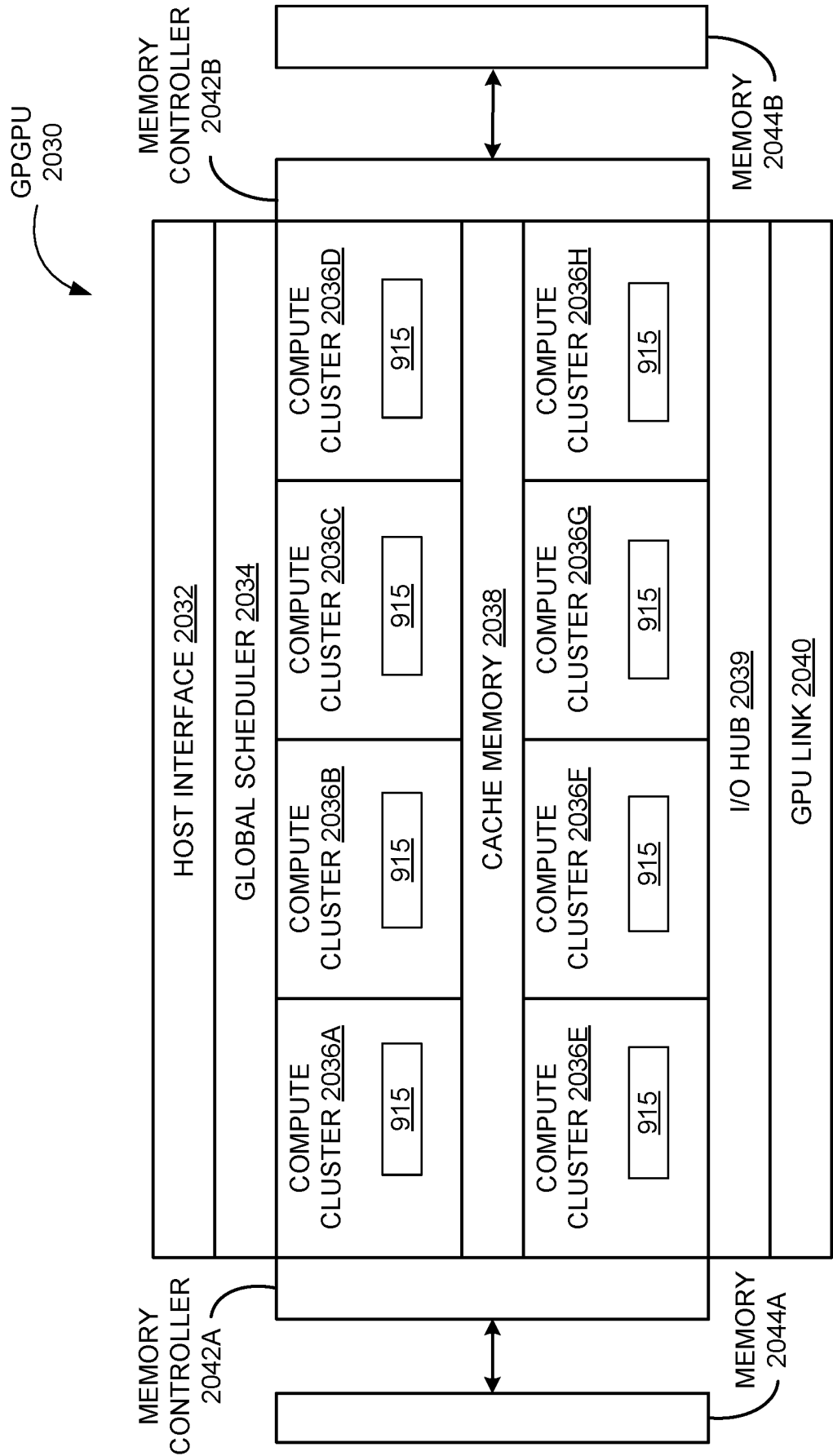
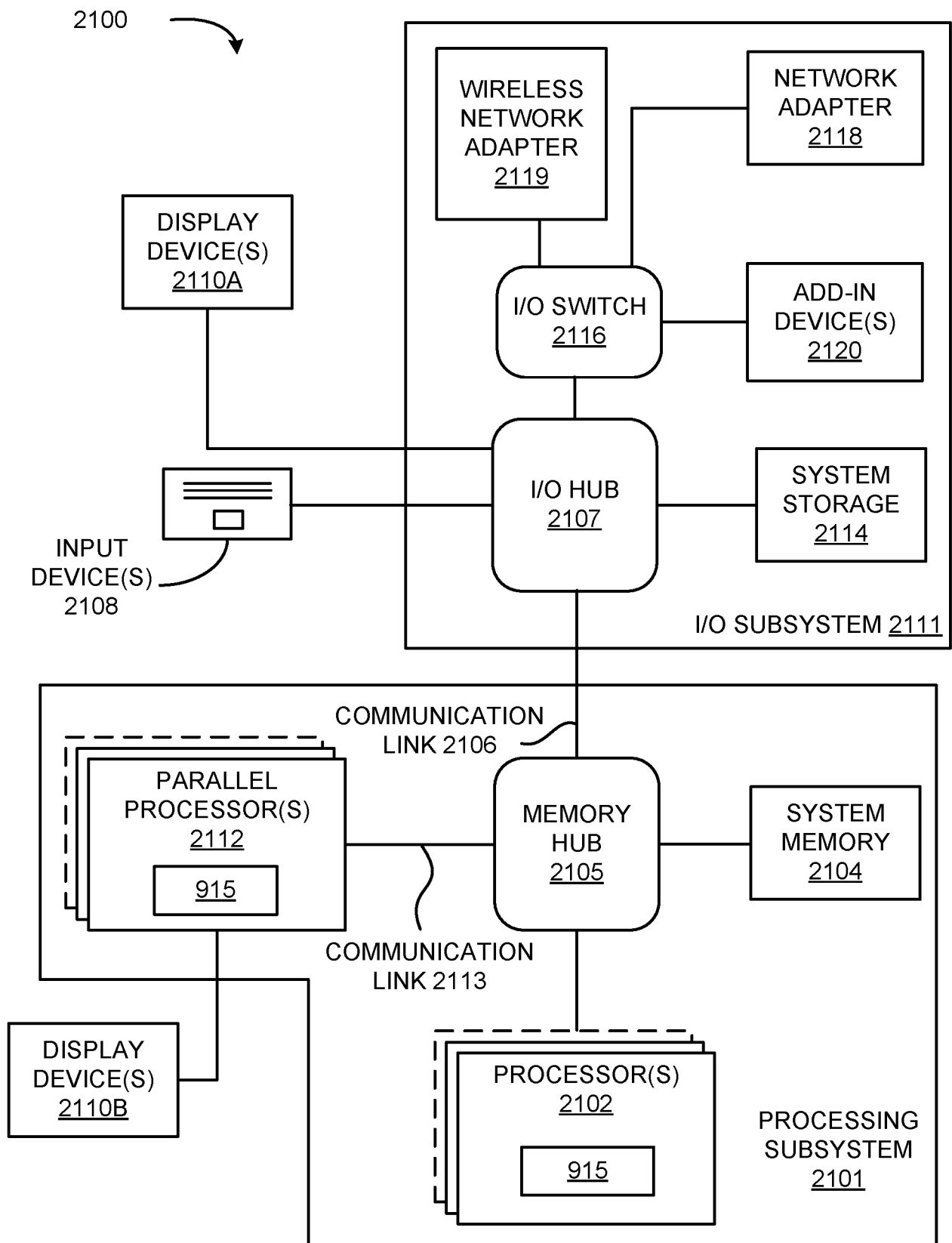
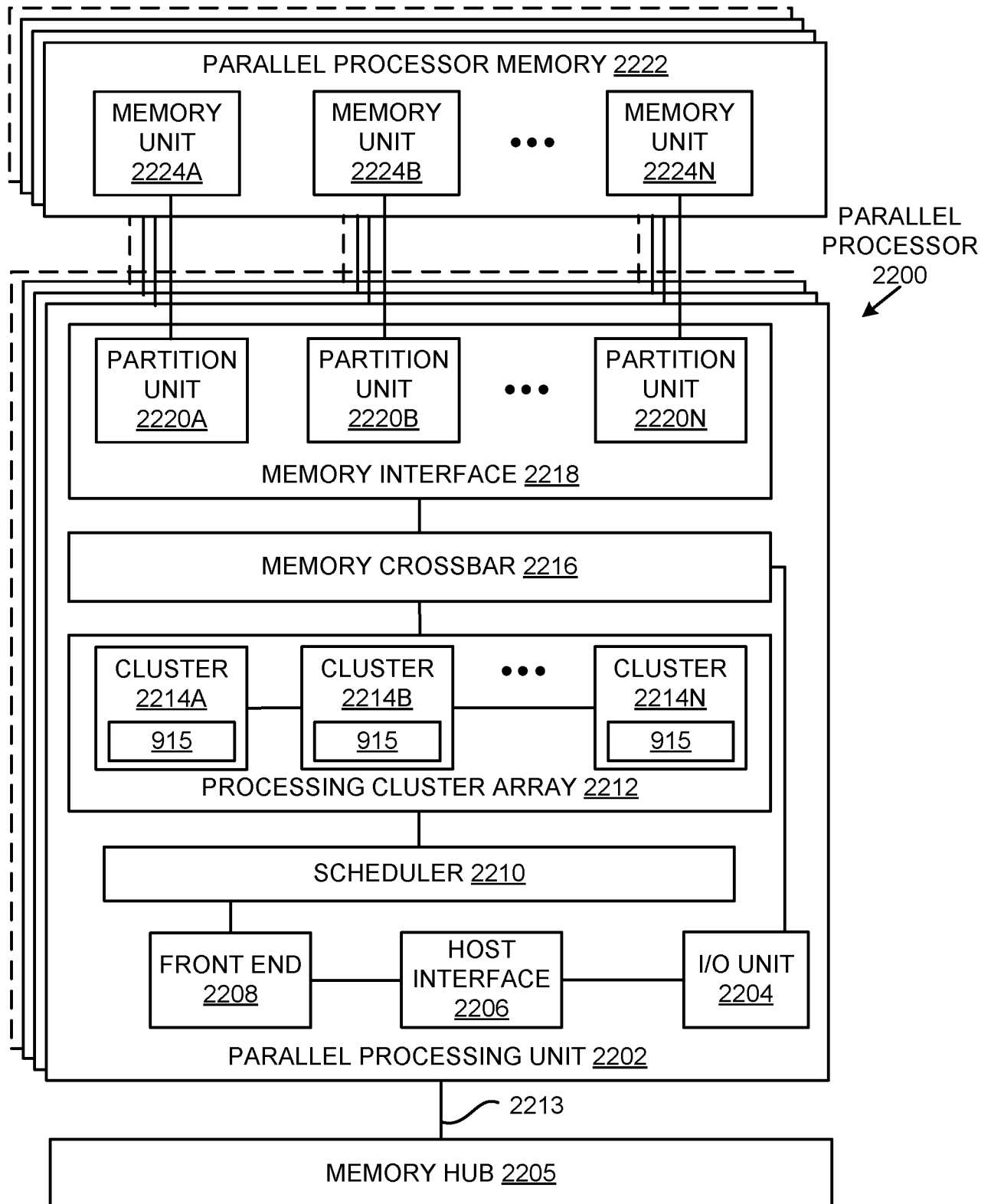
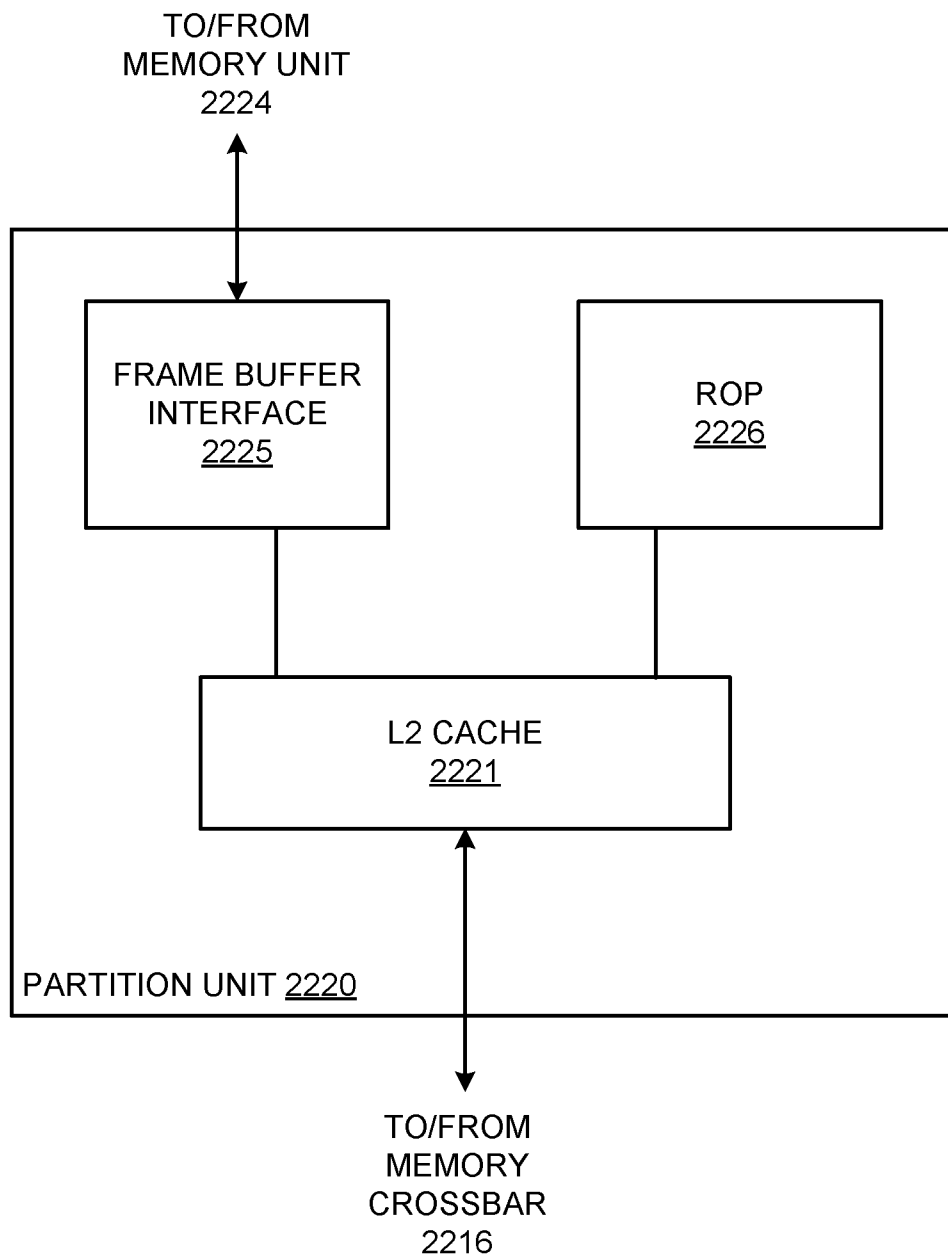
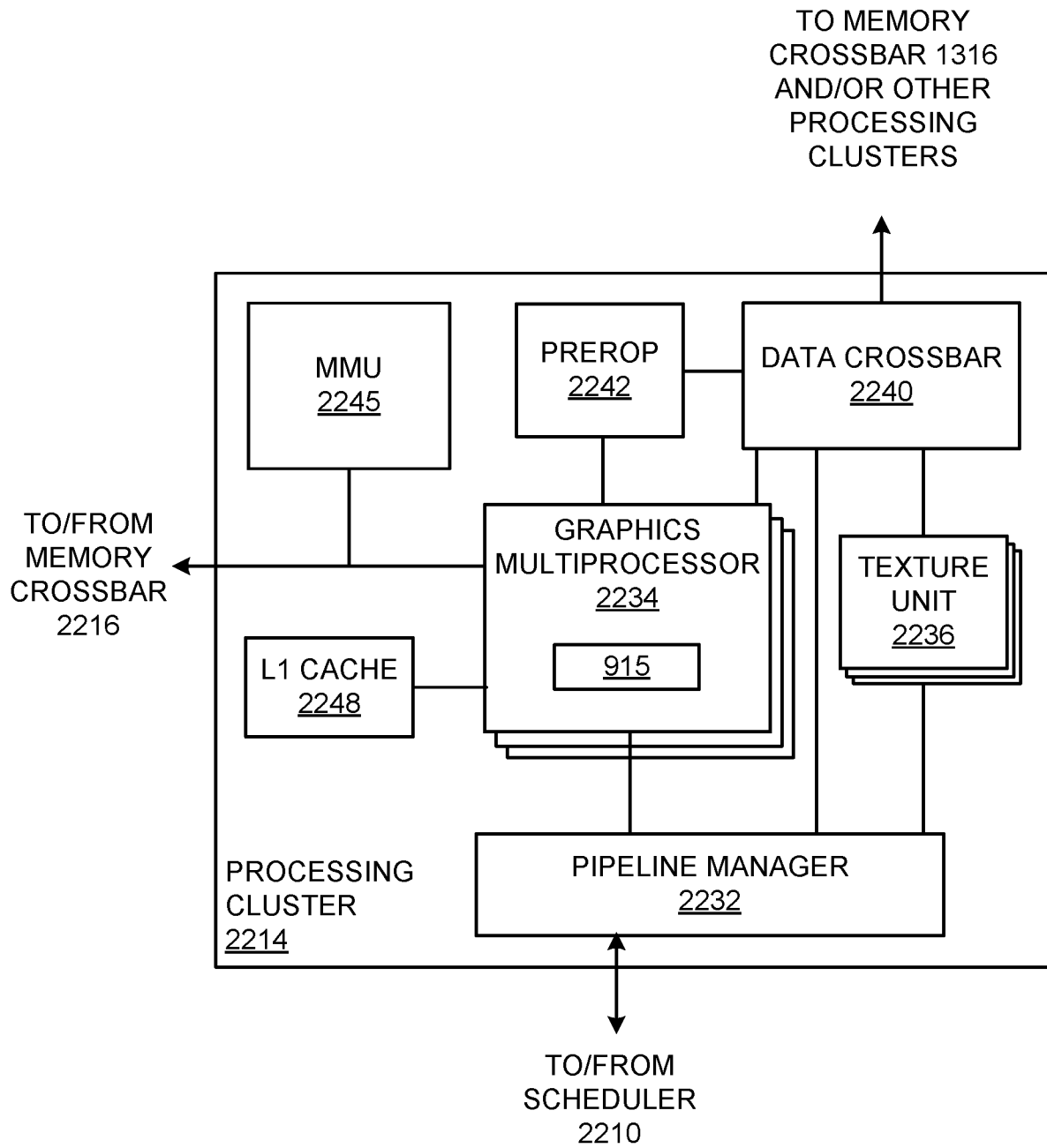


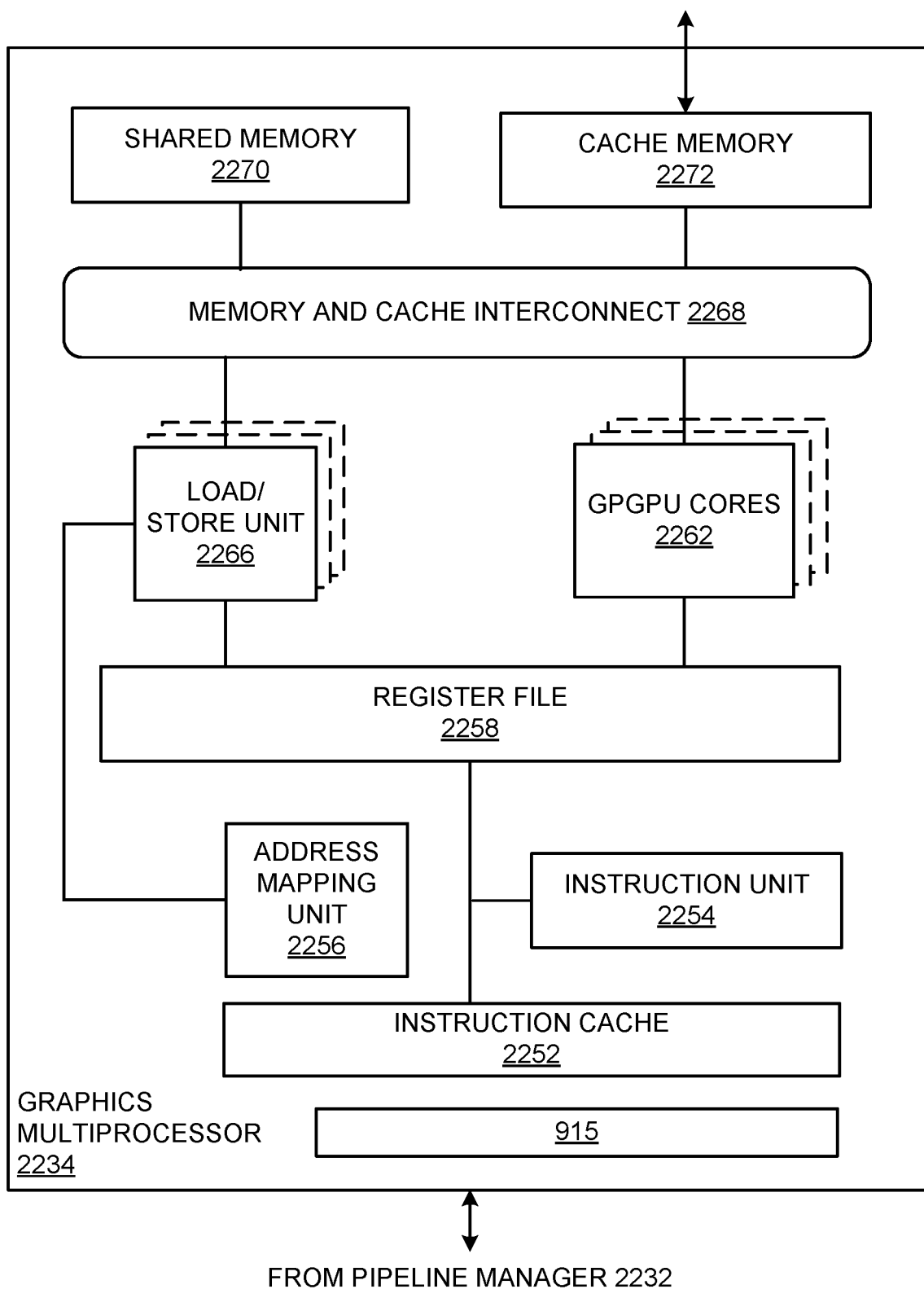
FIG. 20B

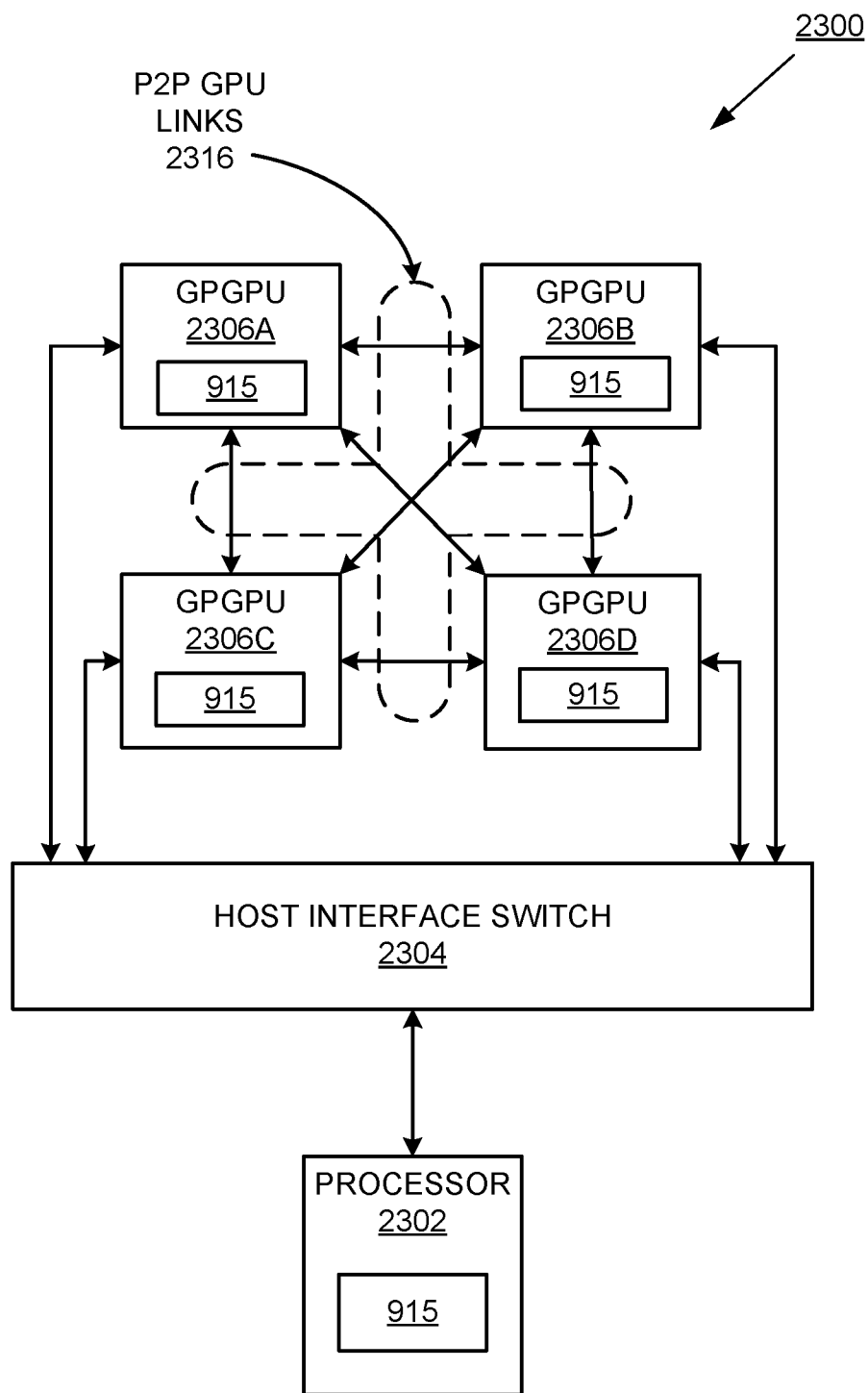
**FIG. 21**

**FIG. 22 A**

**FIG. 22 B**

**FIG. 22 C**

**FIG. 22 D**

**FIG. 23**

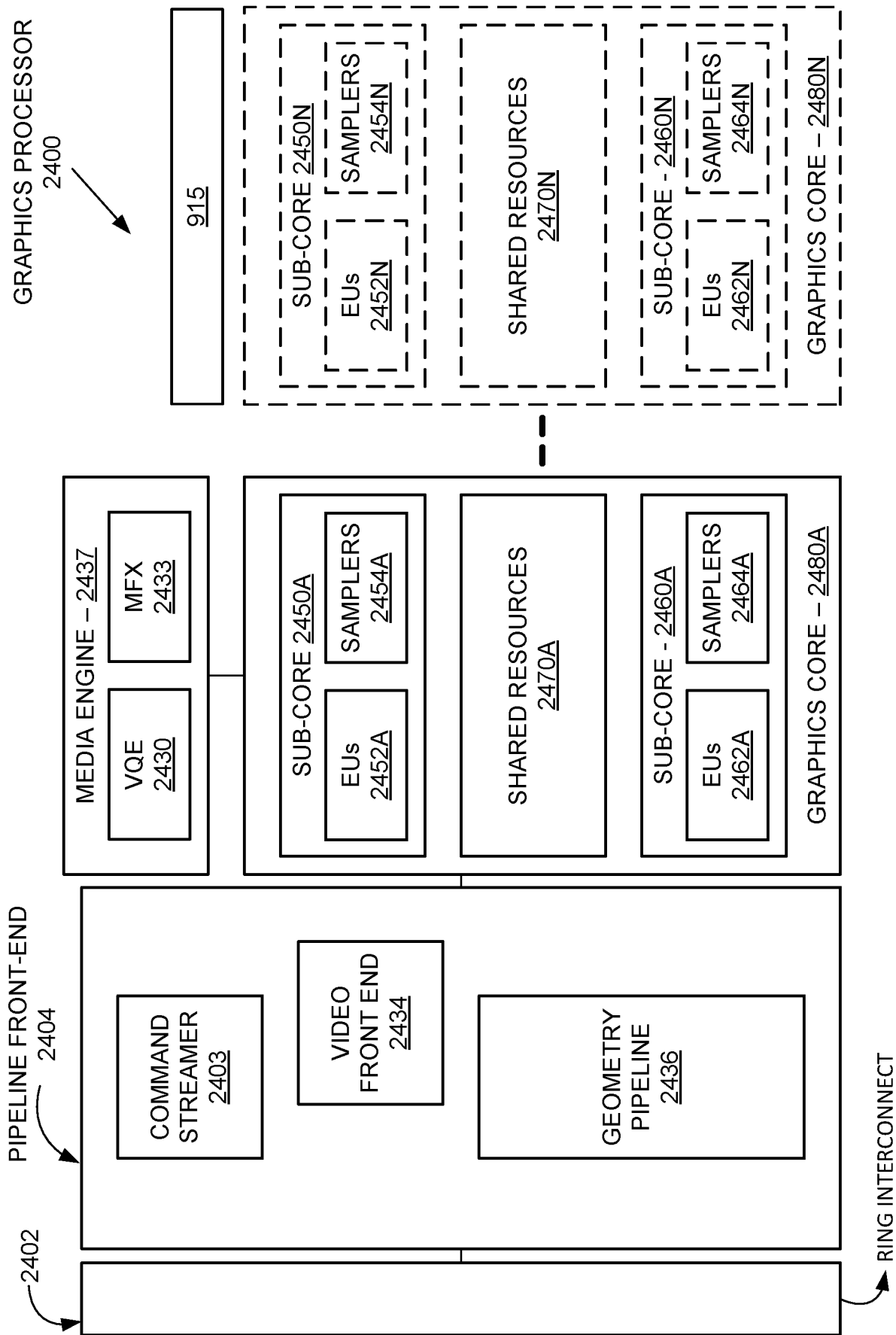


FIG. 24

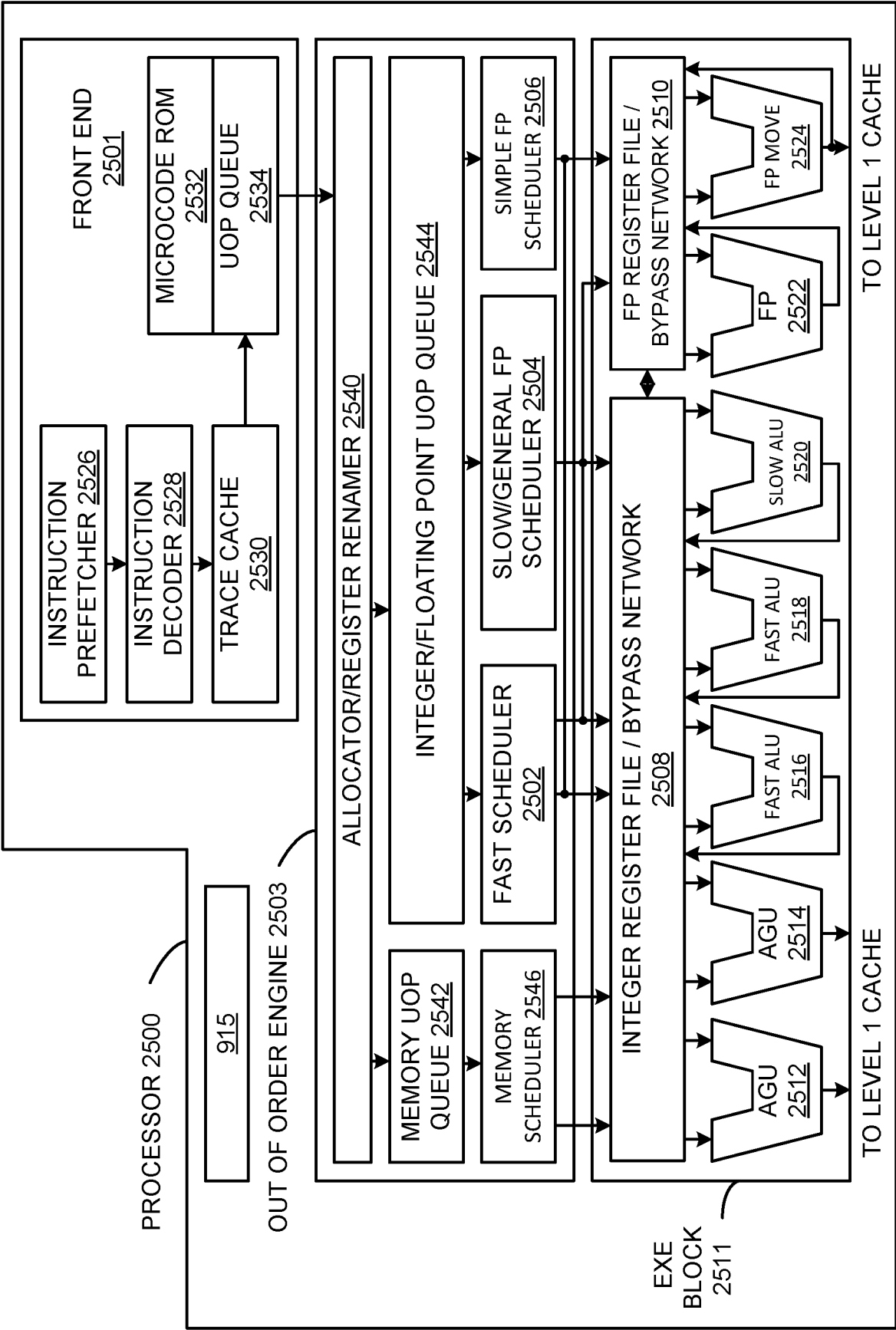


FIG. 25

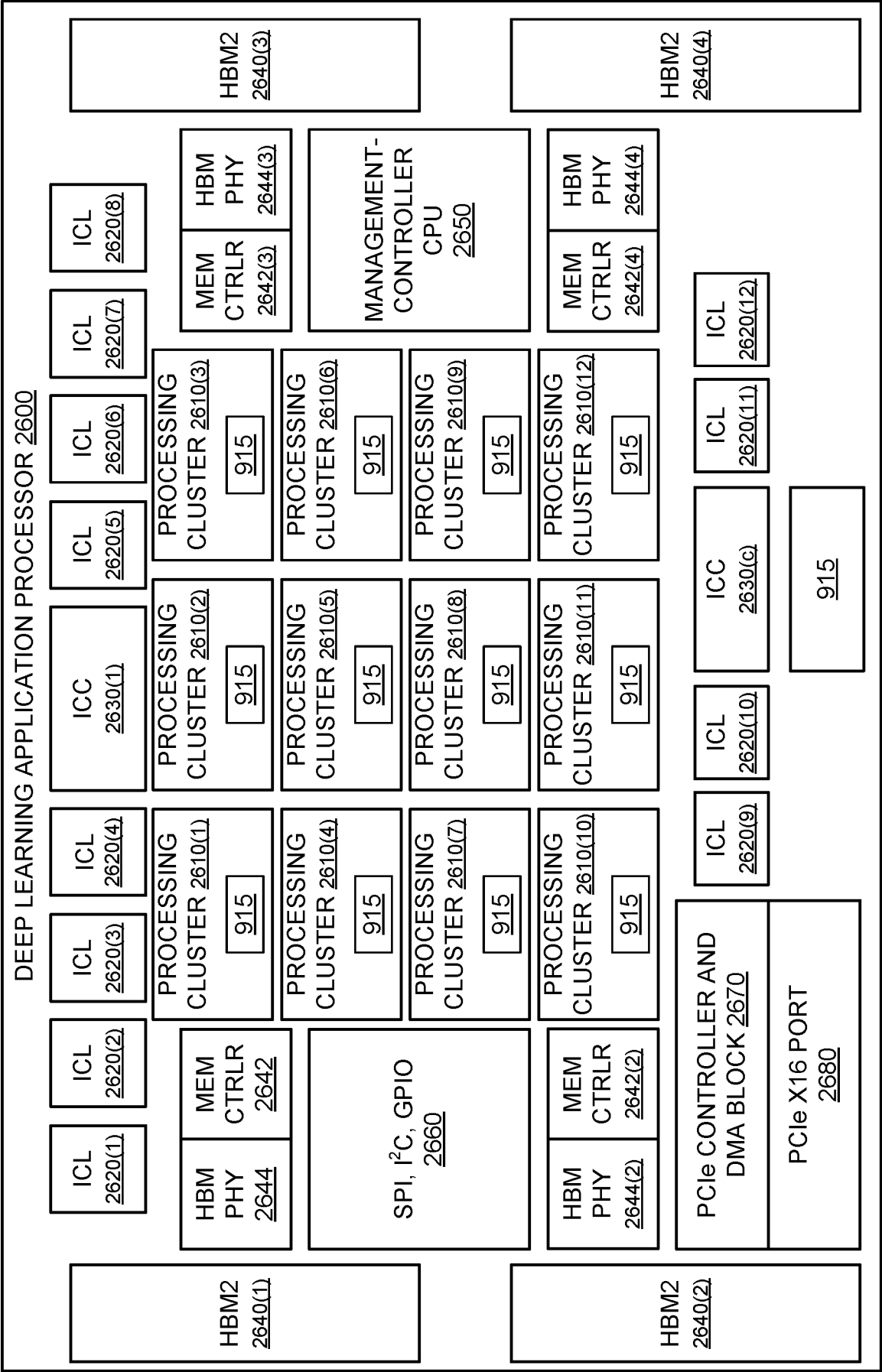


FIG. 26

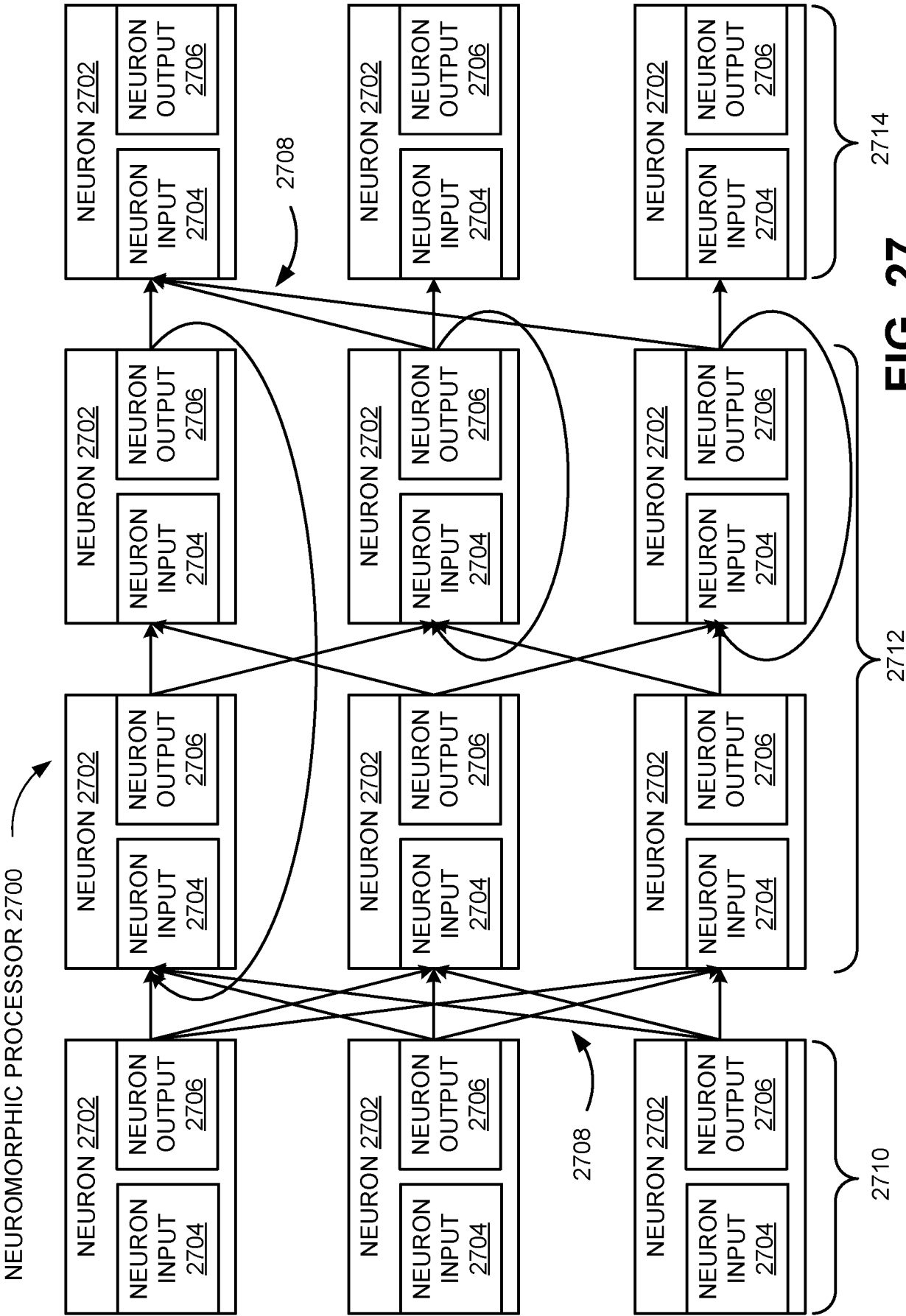
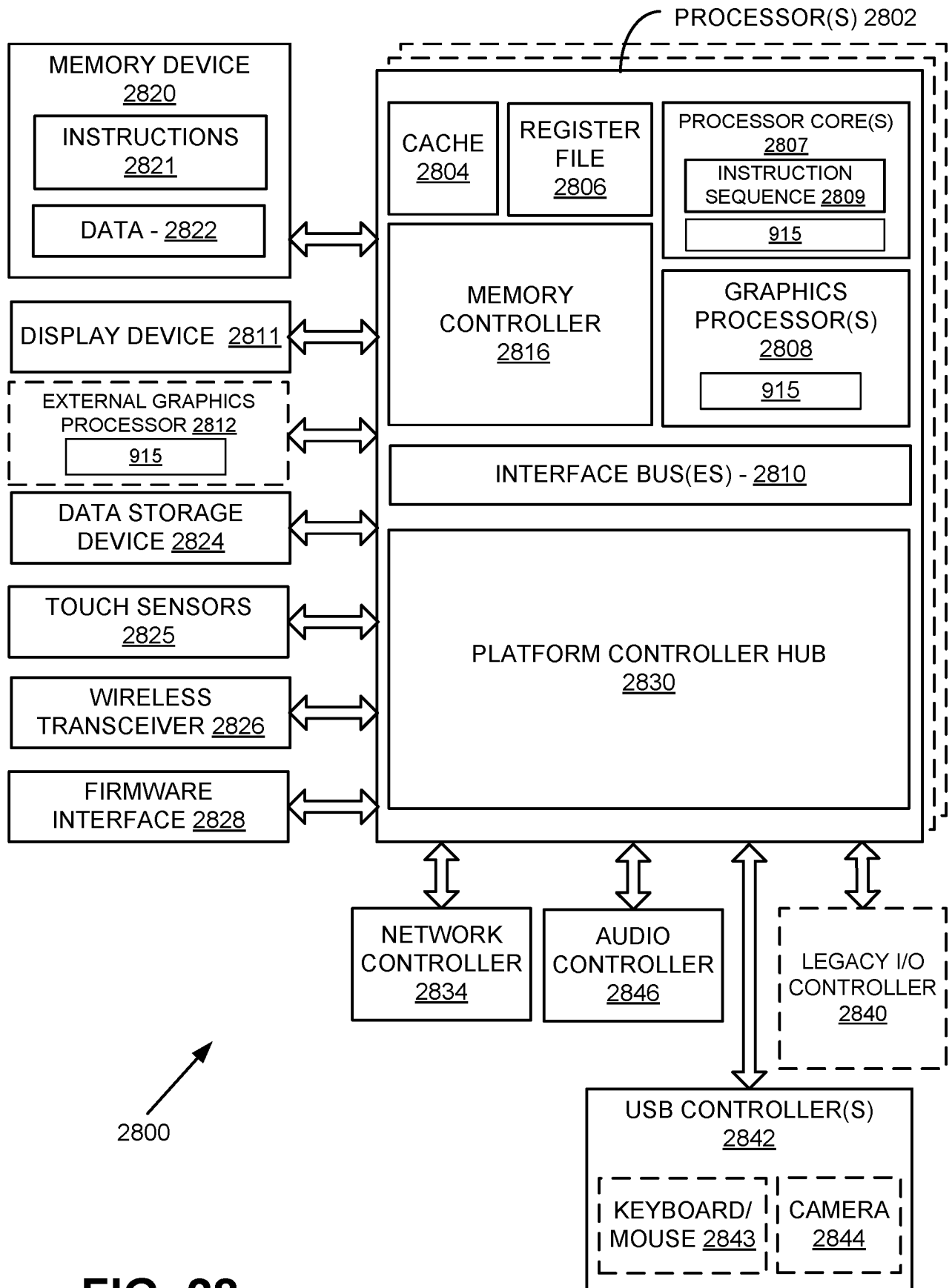


FIG. 27

**FIG. 28**

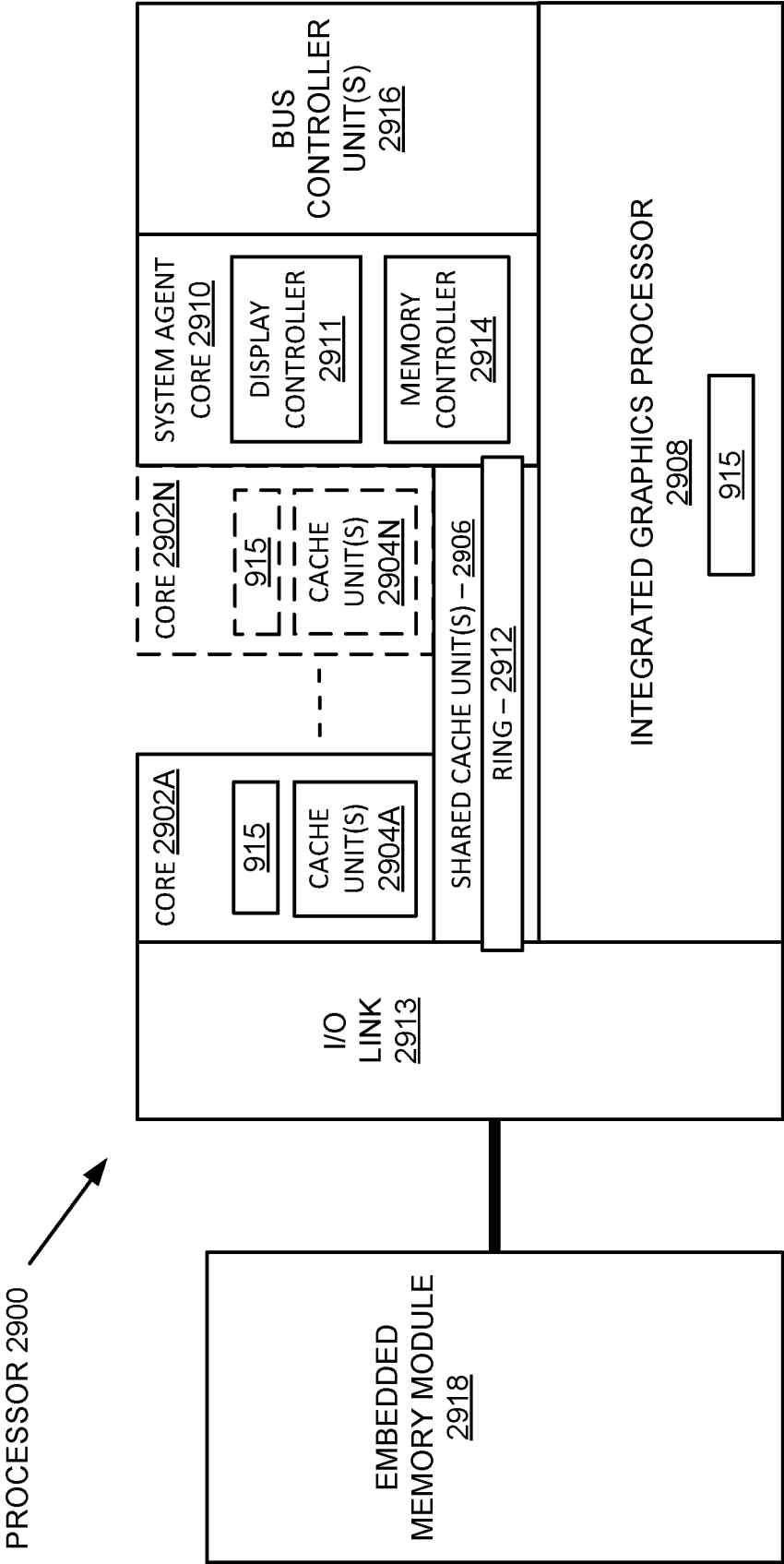


FIG. 29

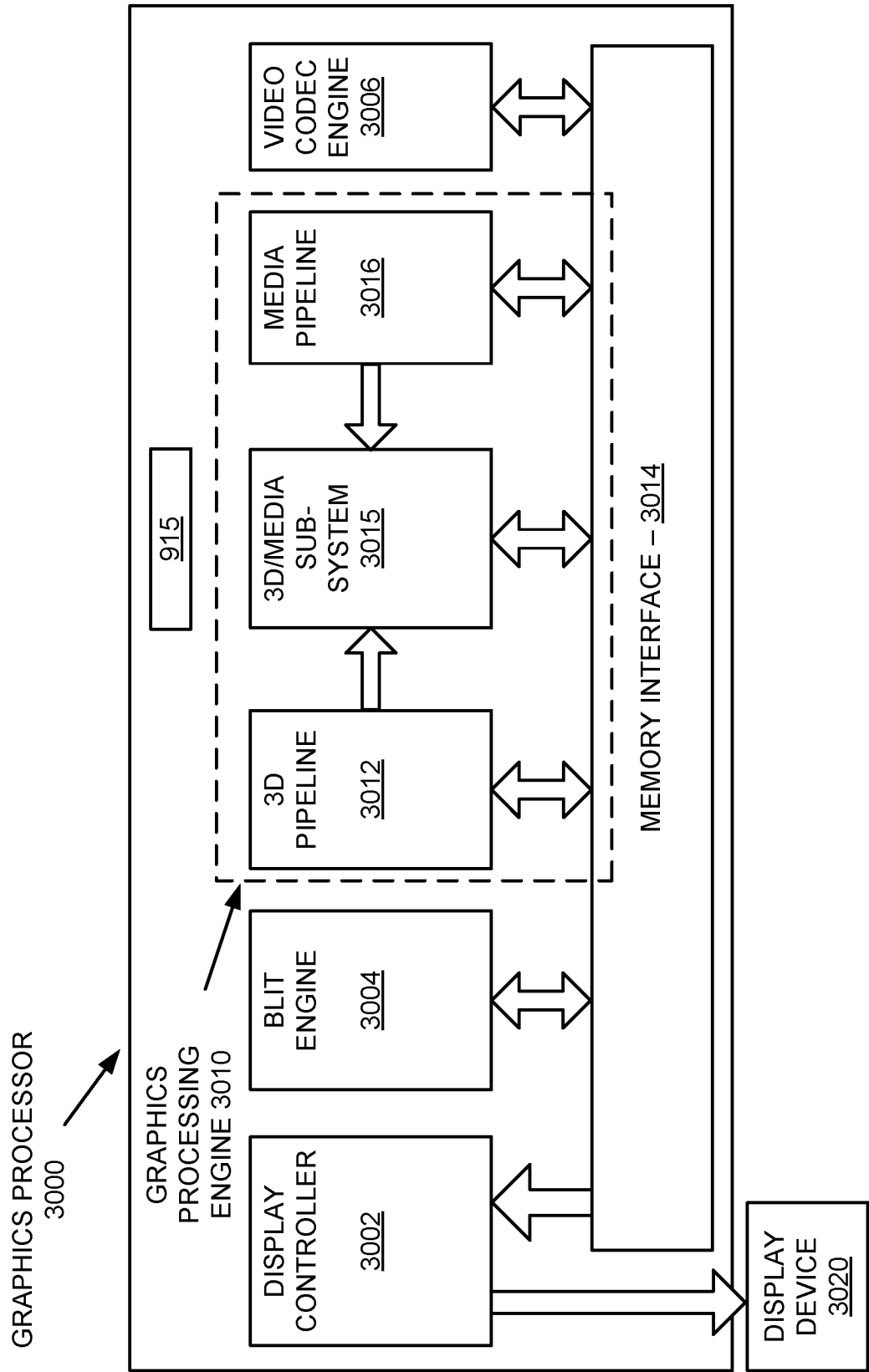


FIG. 30

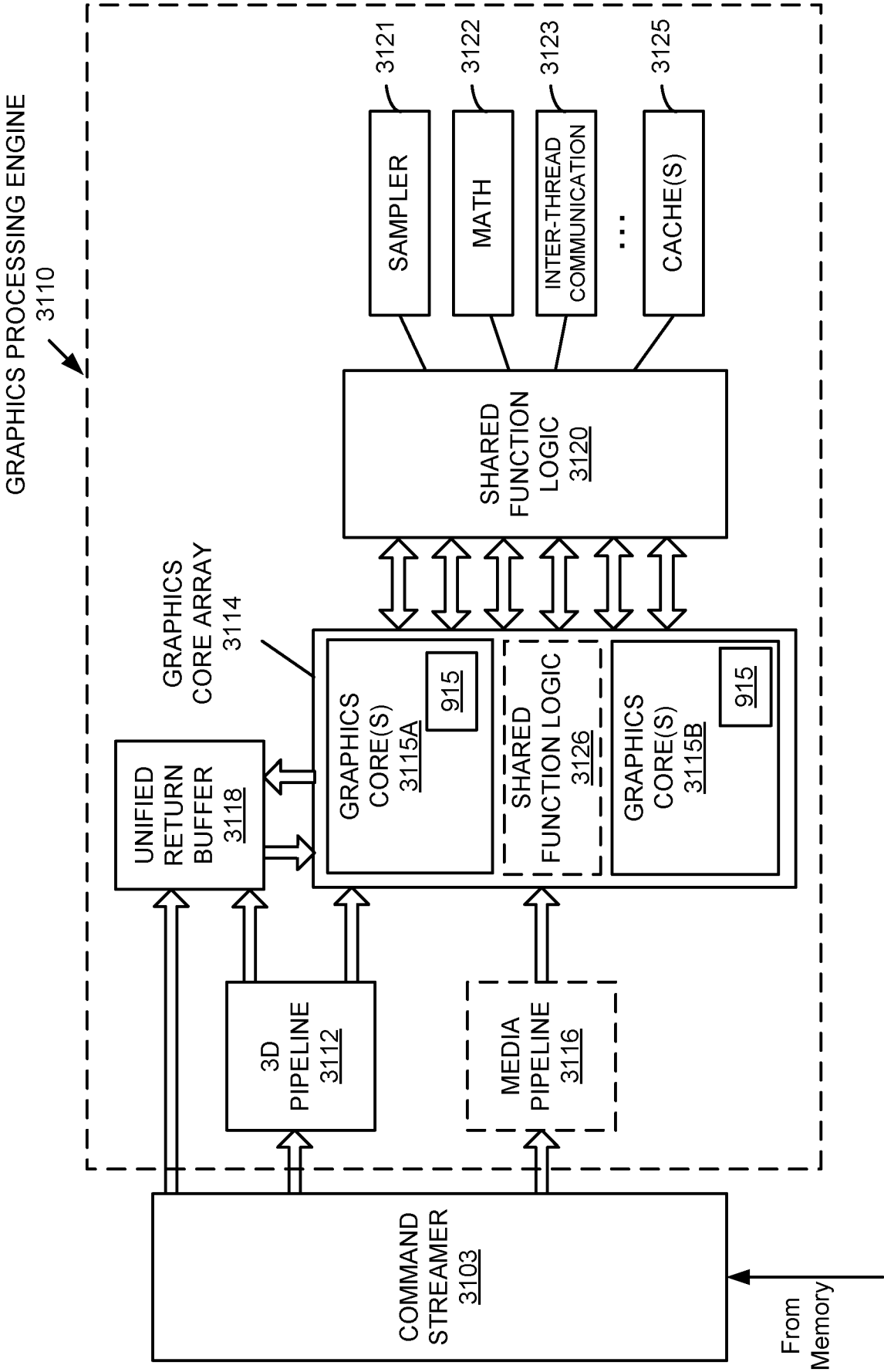


FIG. 31

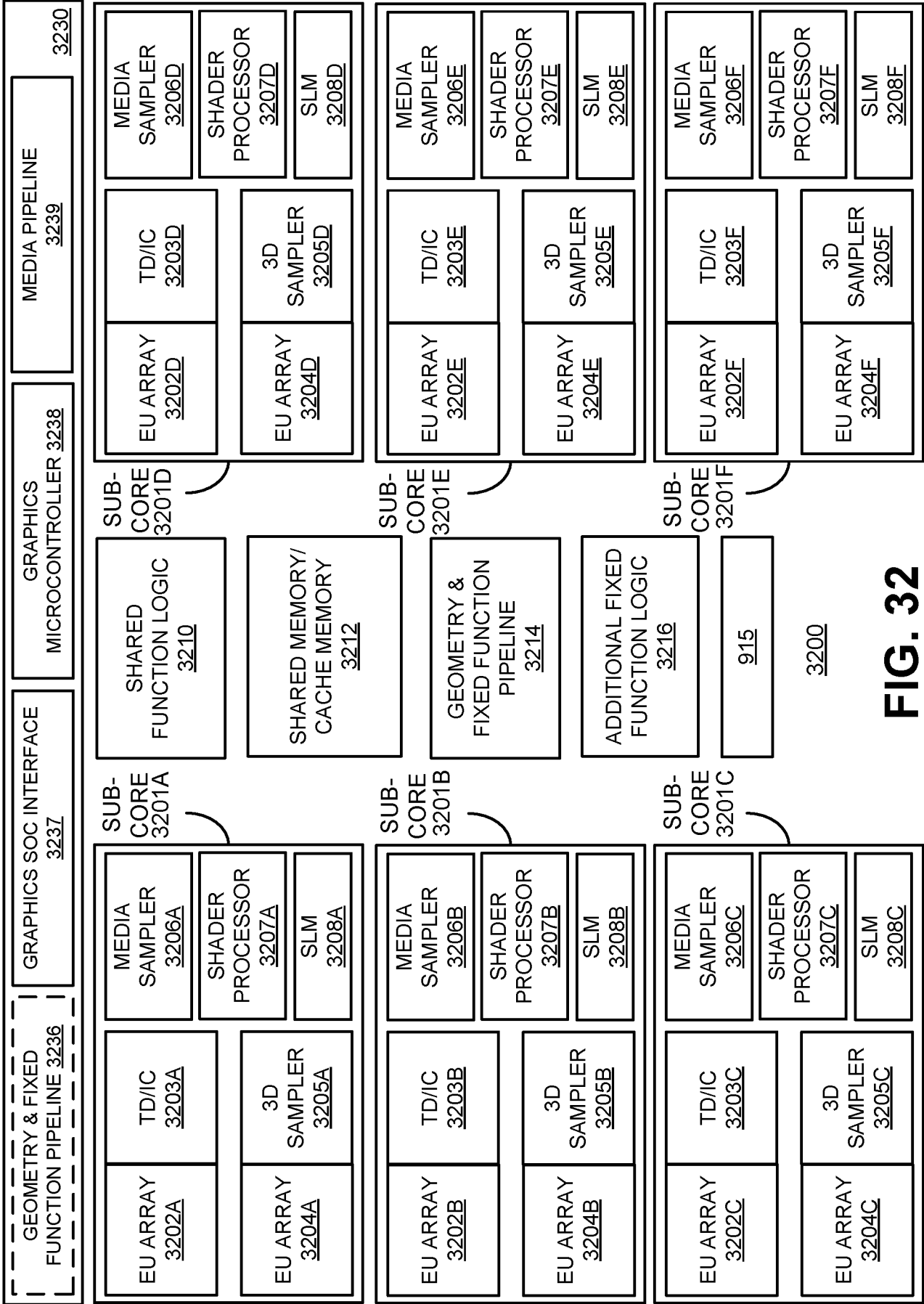


FIG. 32

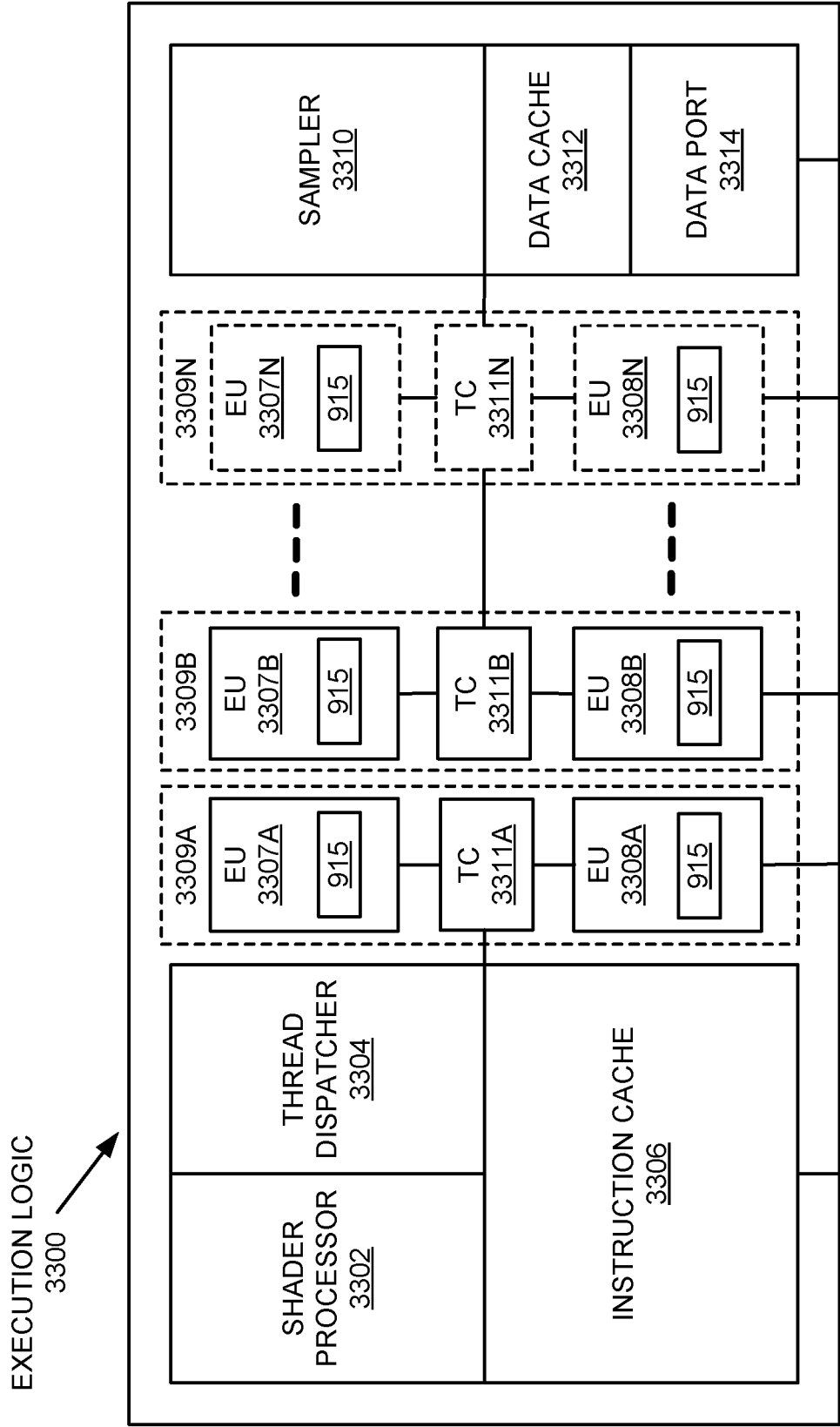


FIG. 33 A

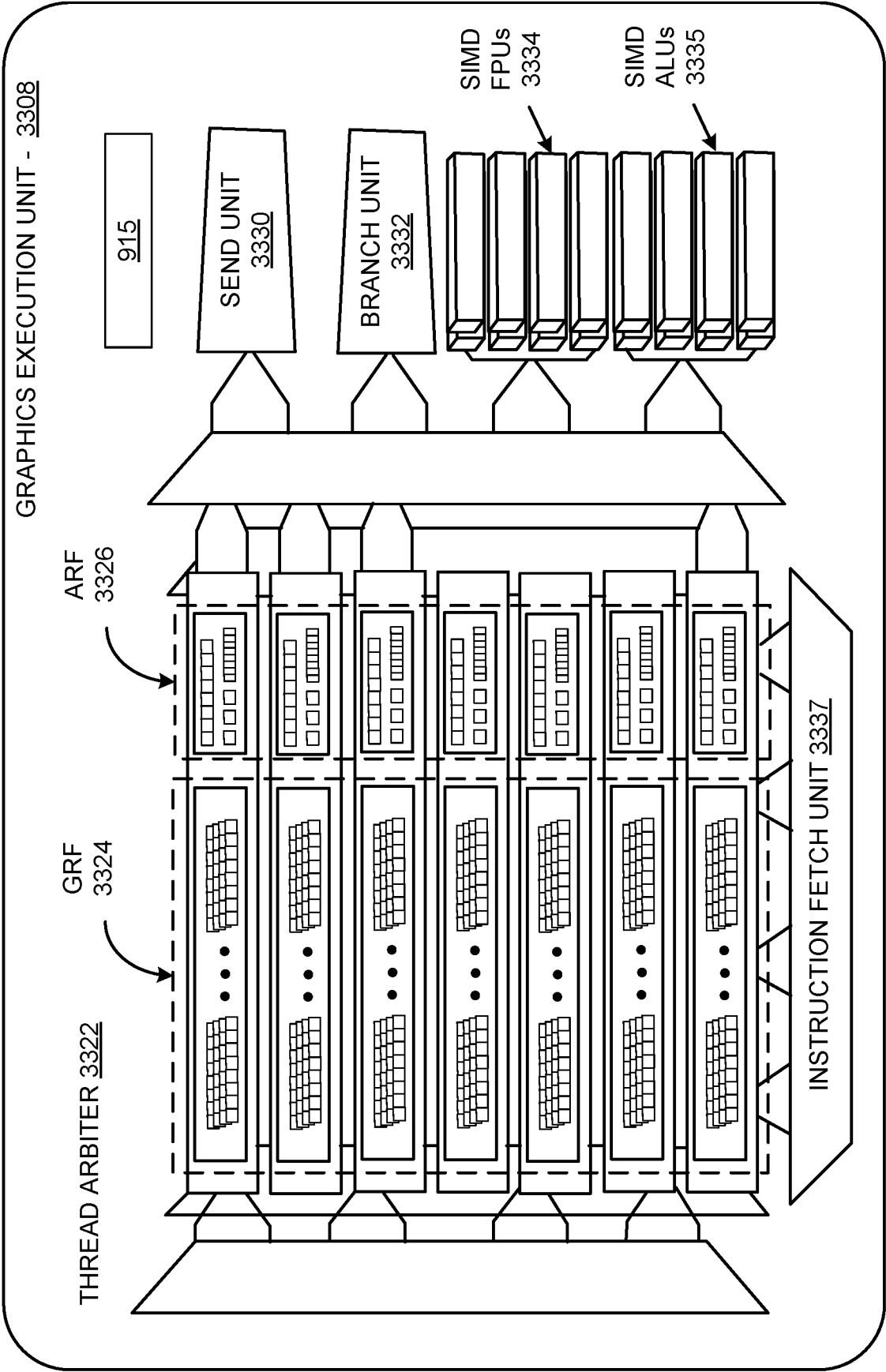
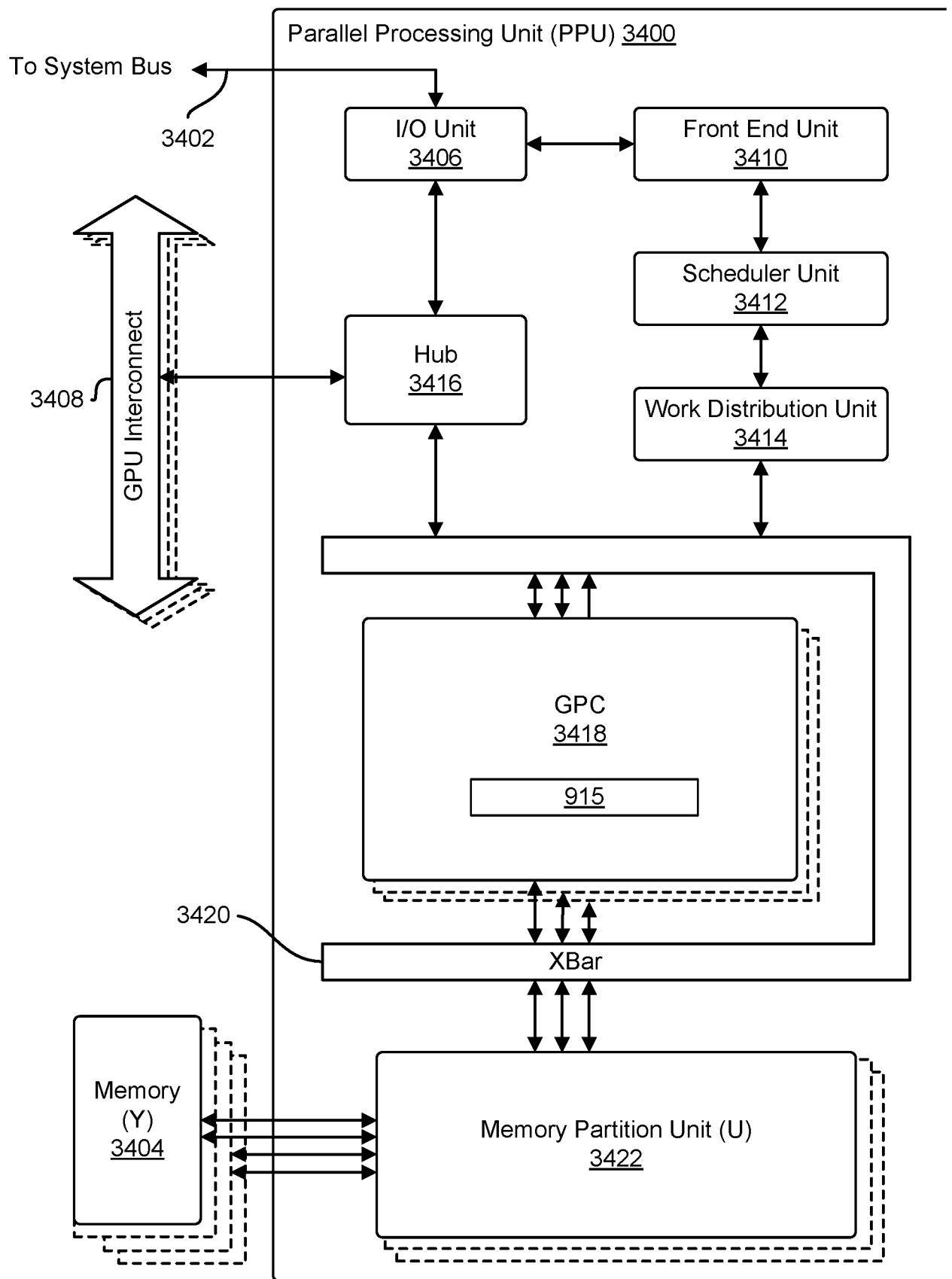
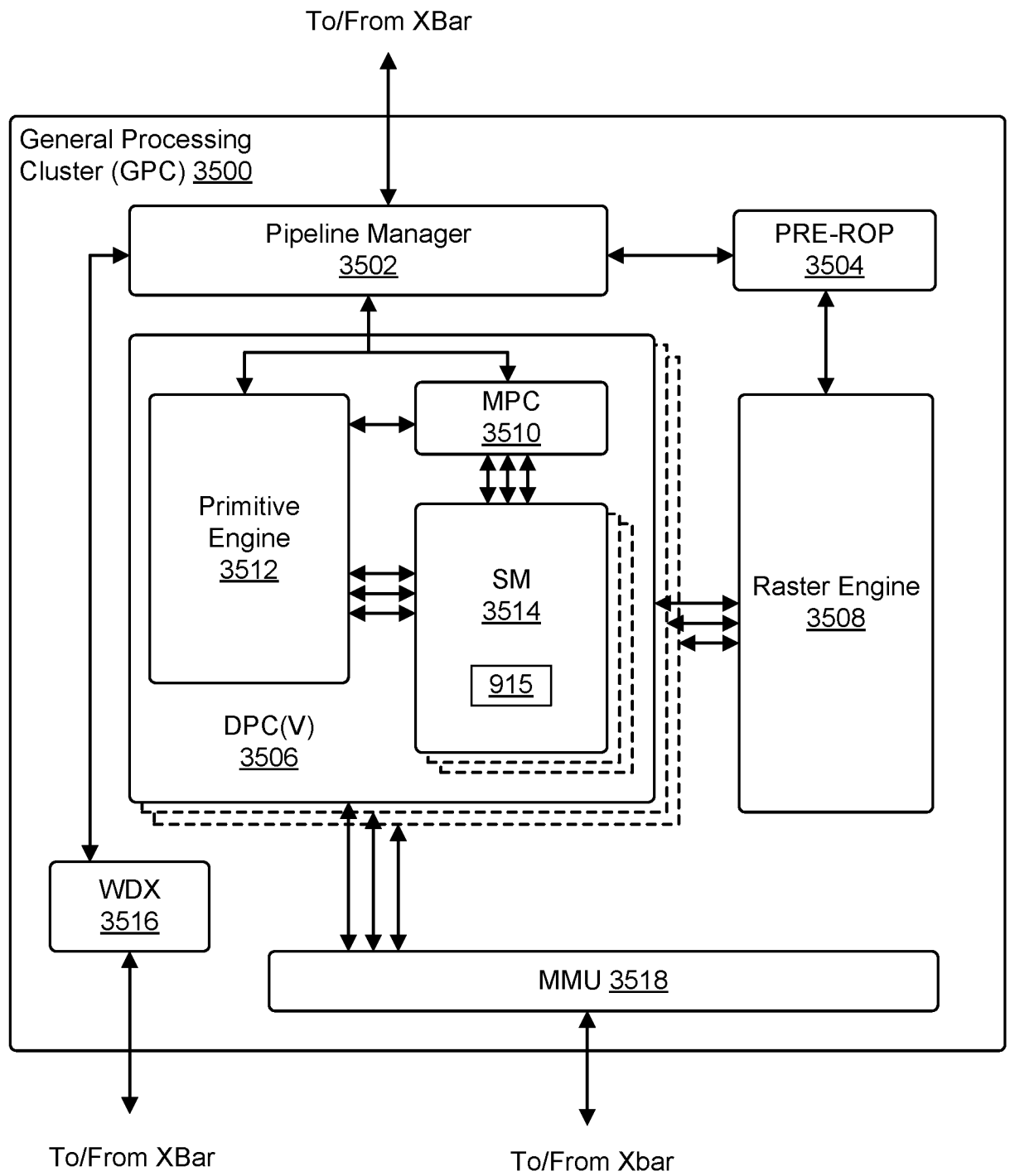
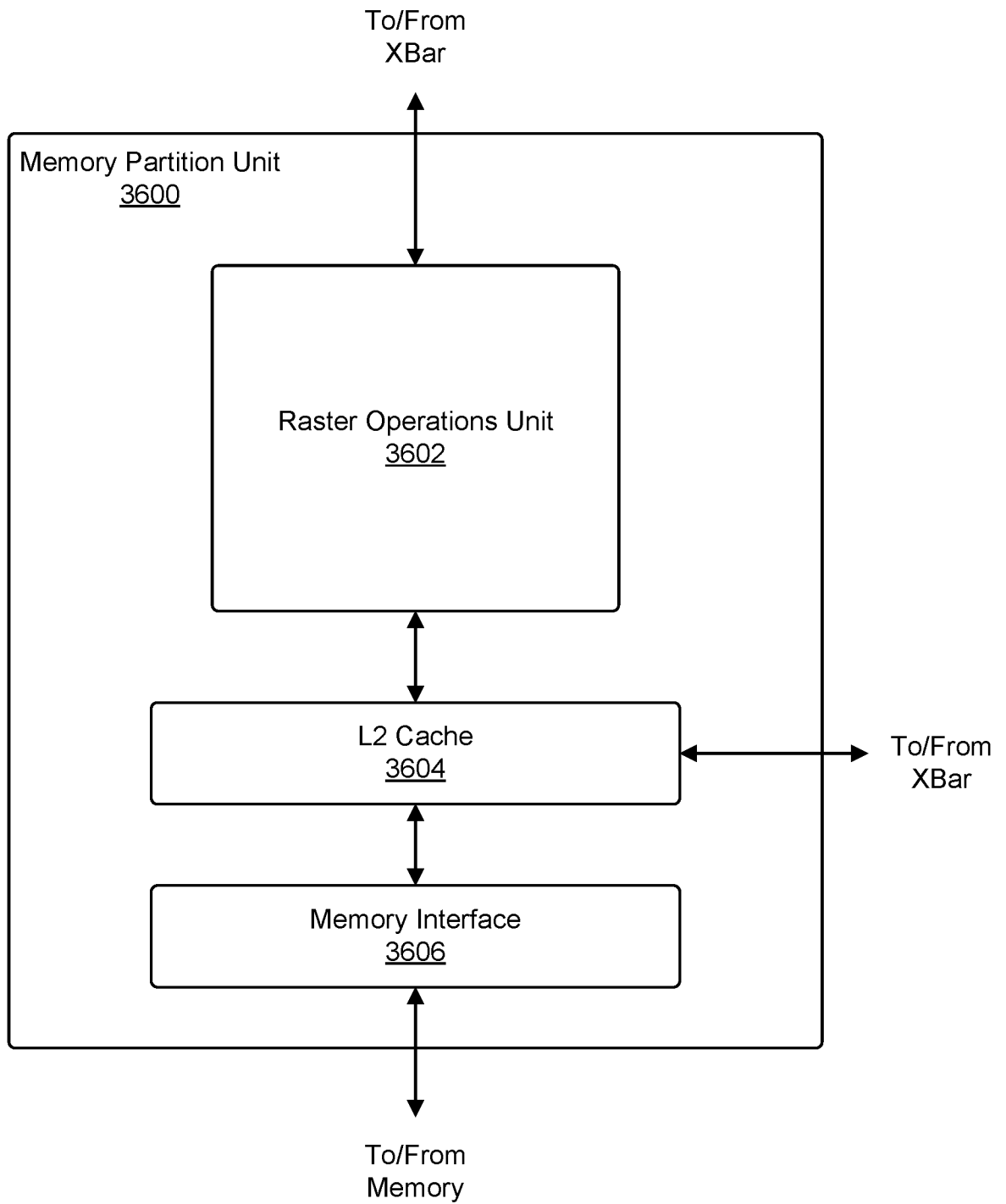
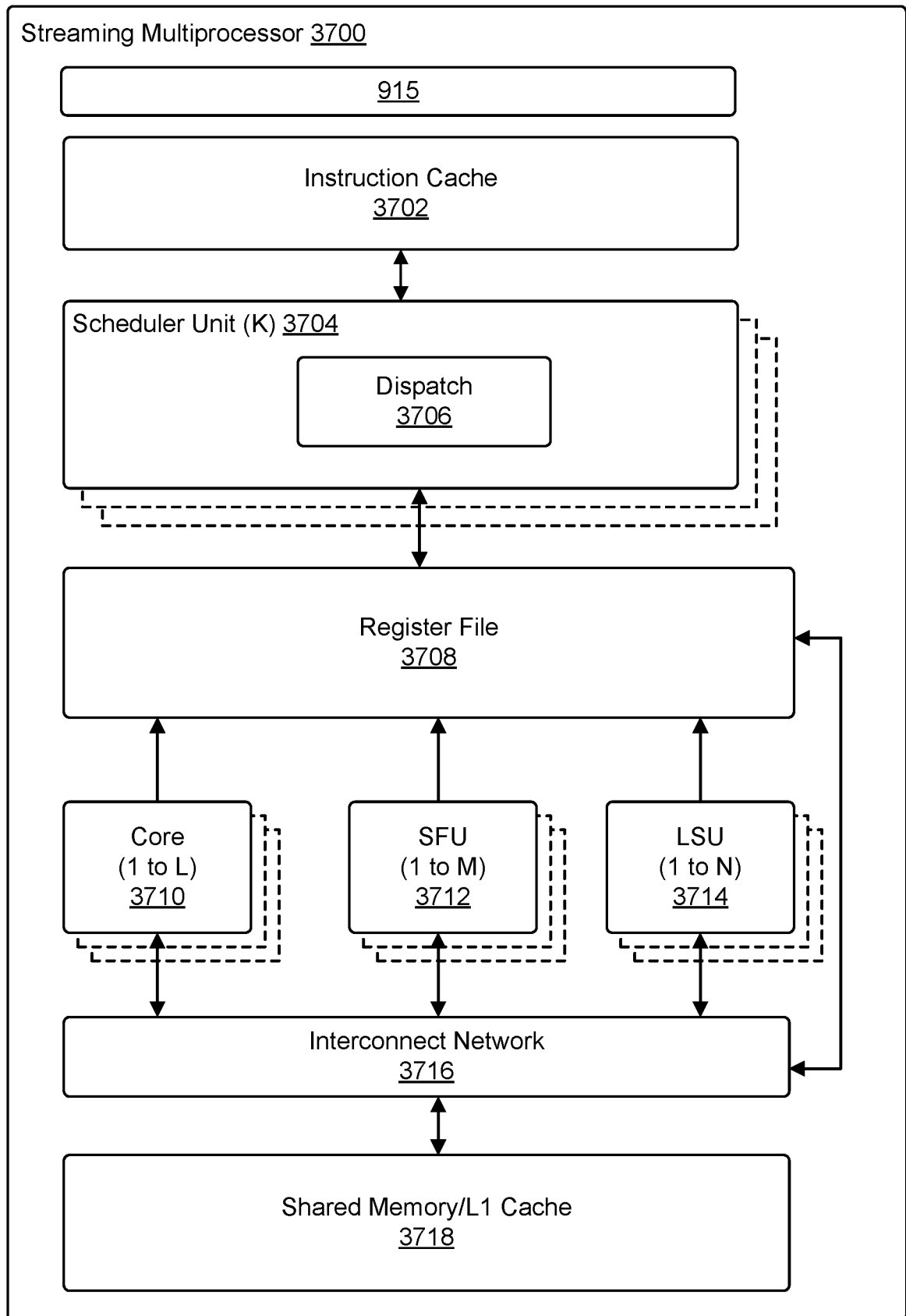


FIG. 33 B

**FIG. 34**

**FIG. 35**

**FIG. 36**

**FIG. 37**

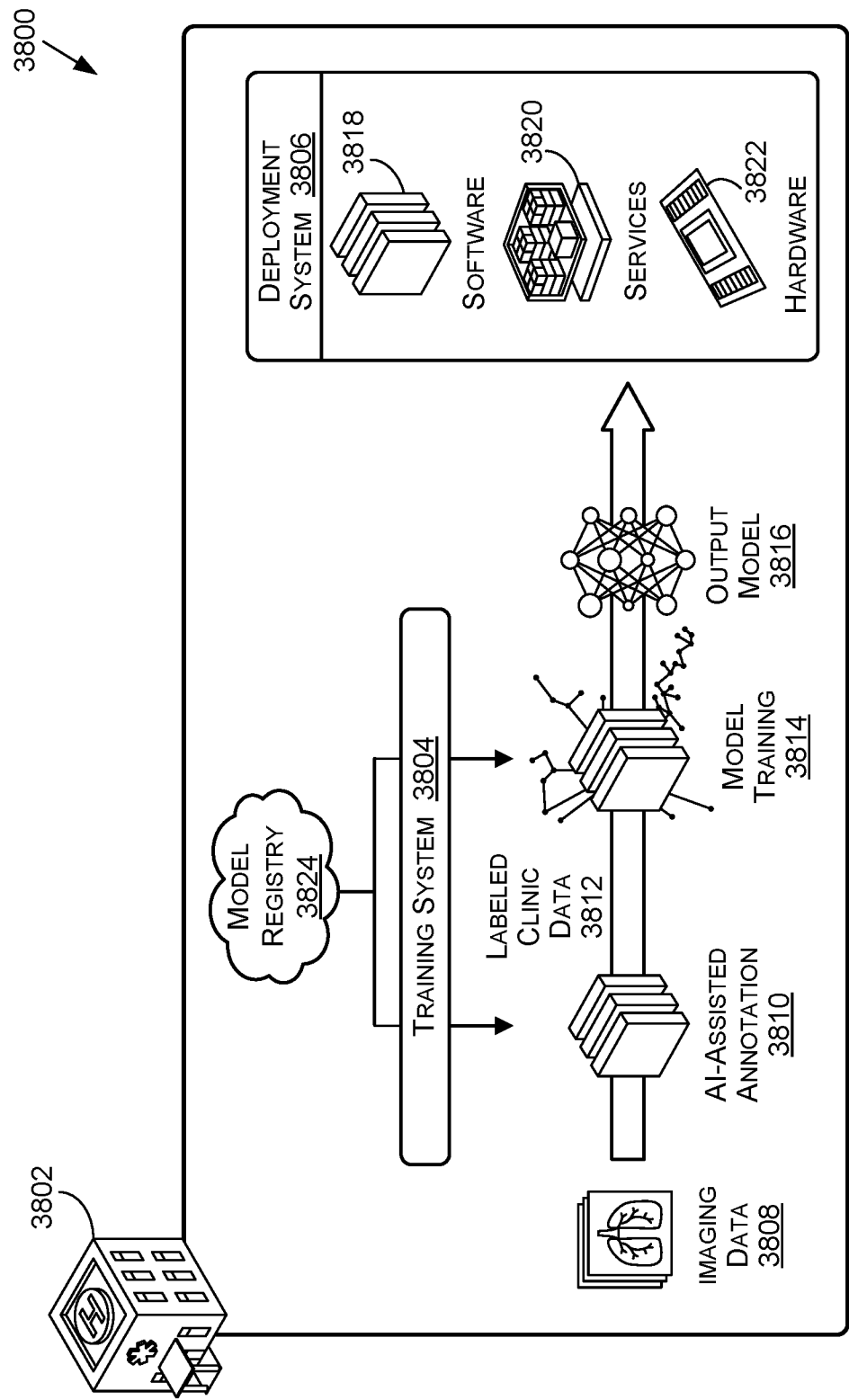


FIG. 38

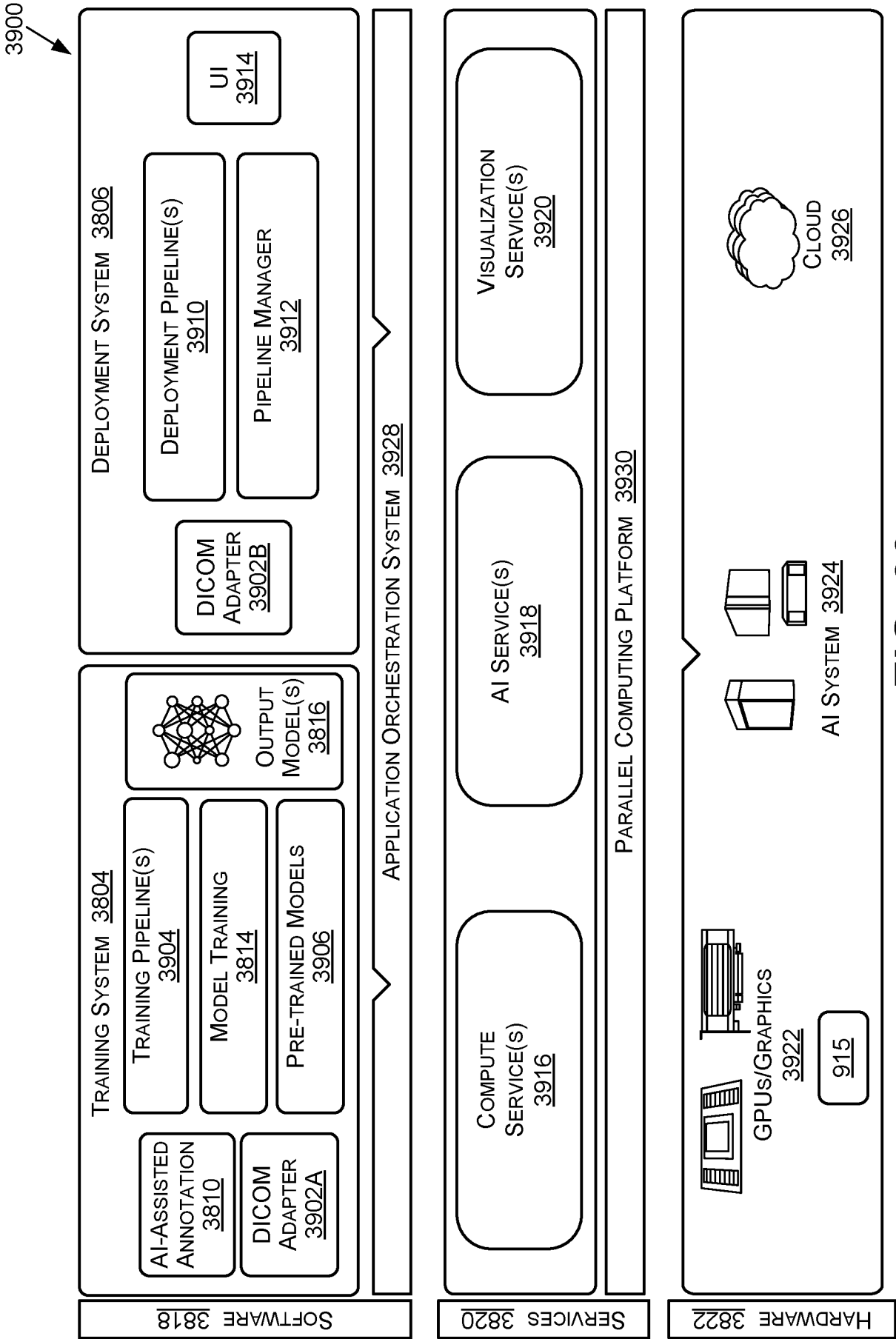


FIG. 39

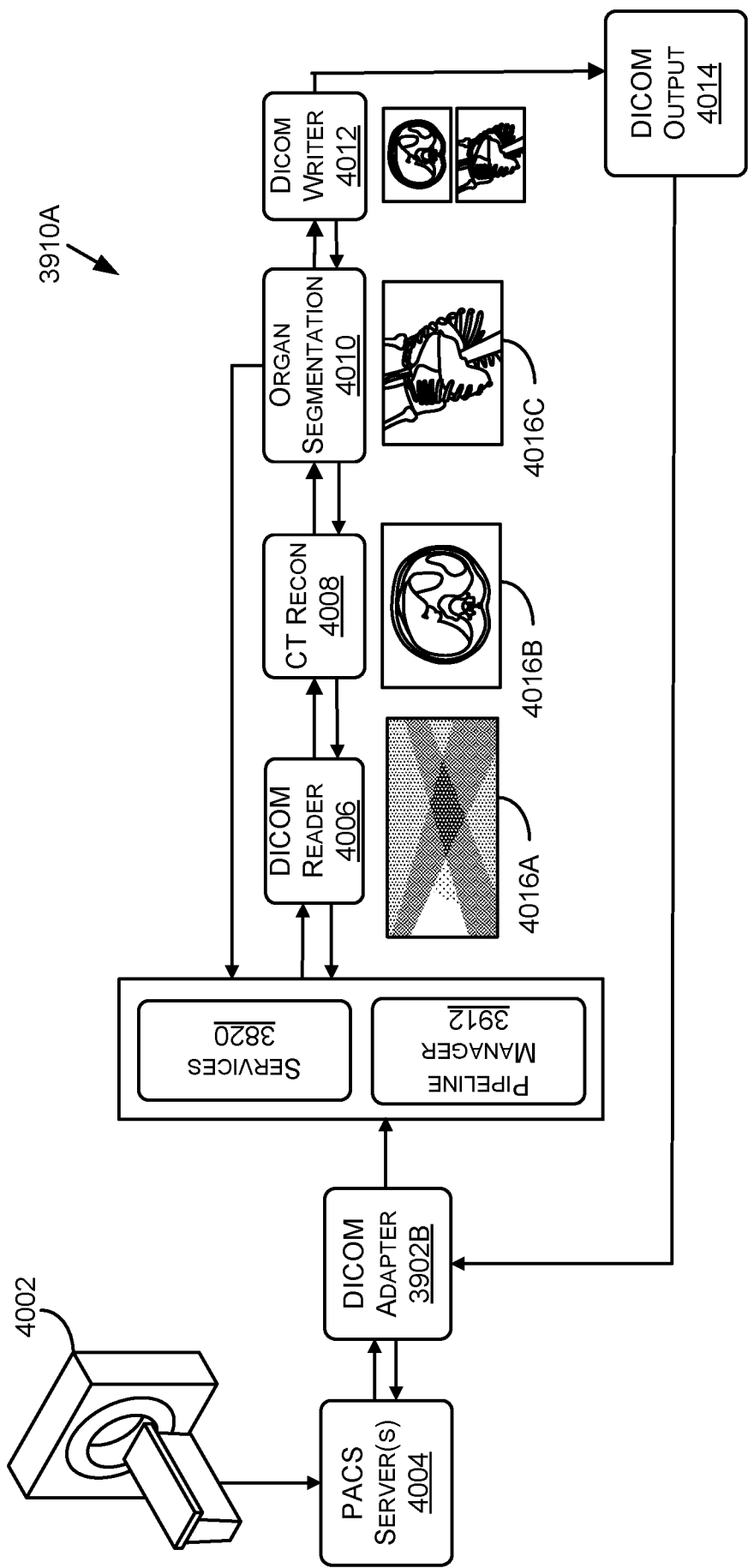


FIG. 40

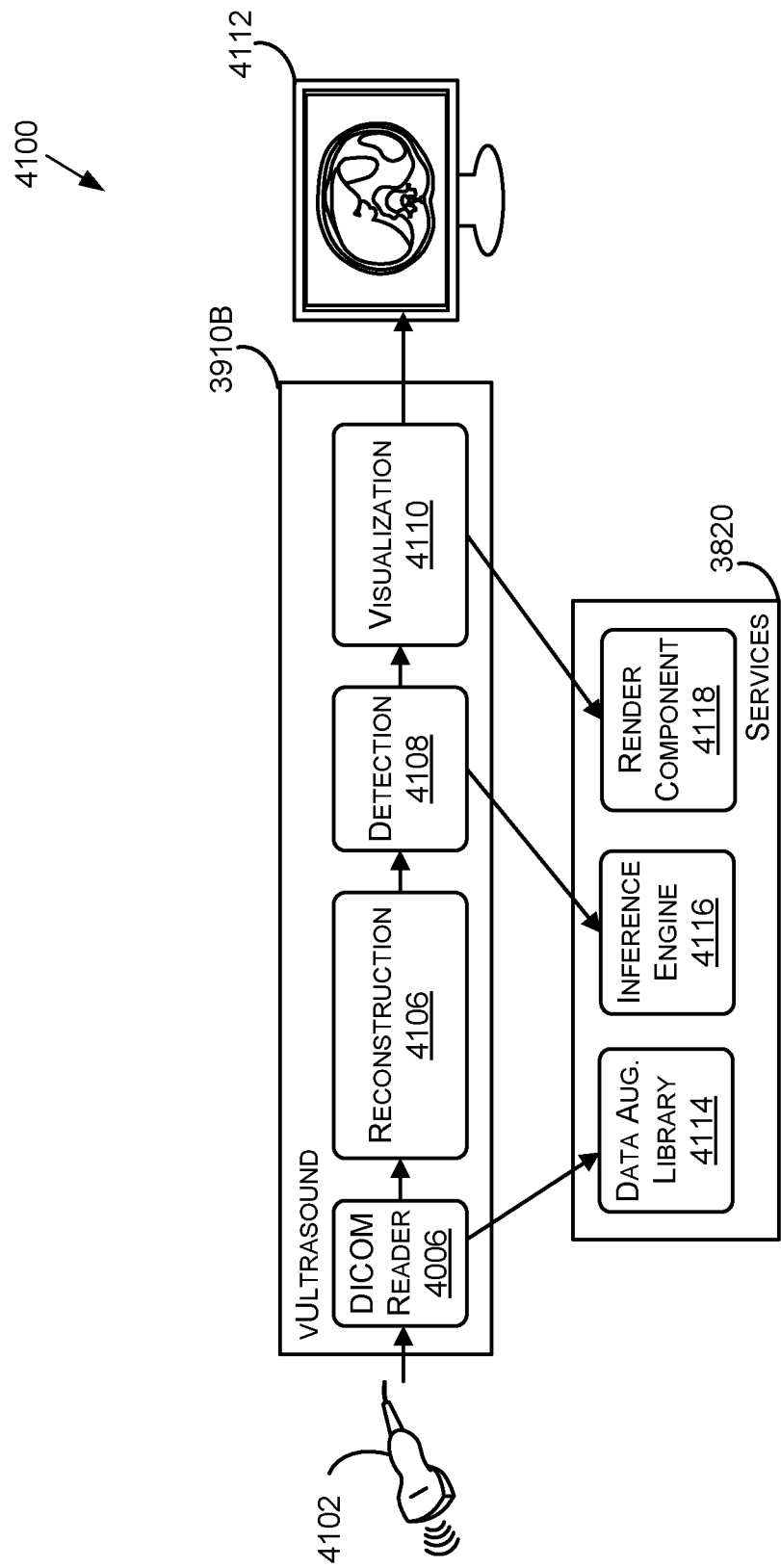


FIG. 41A

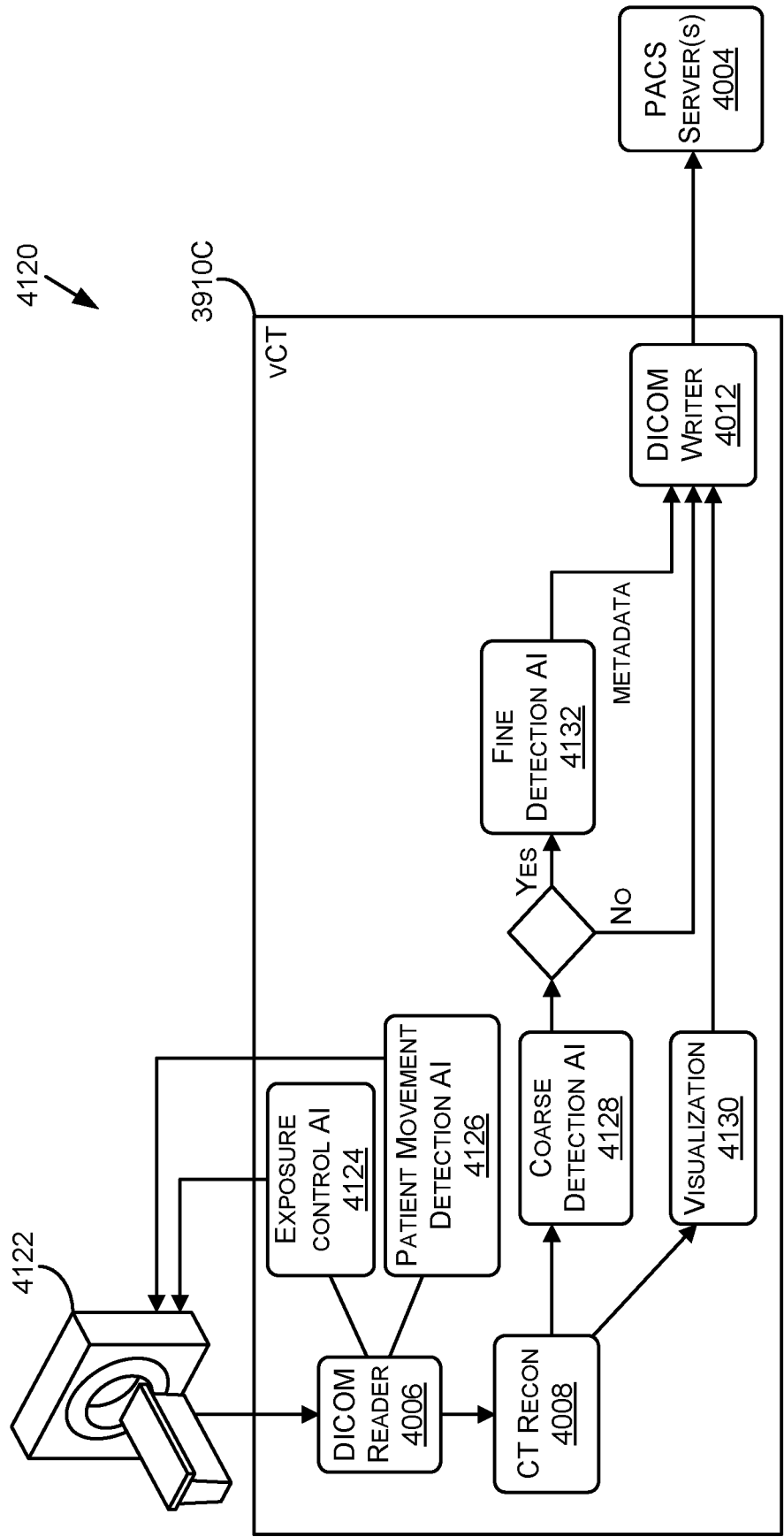


FIG. 41B

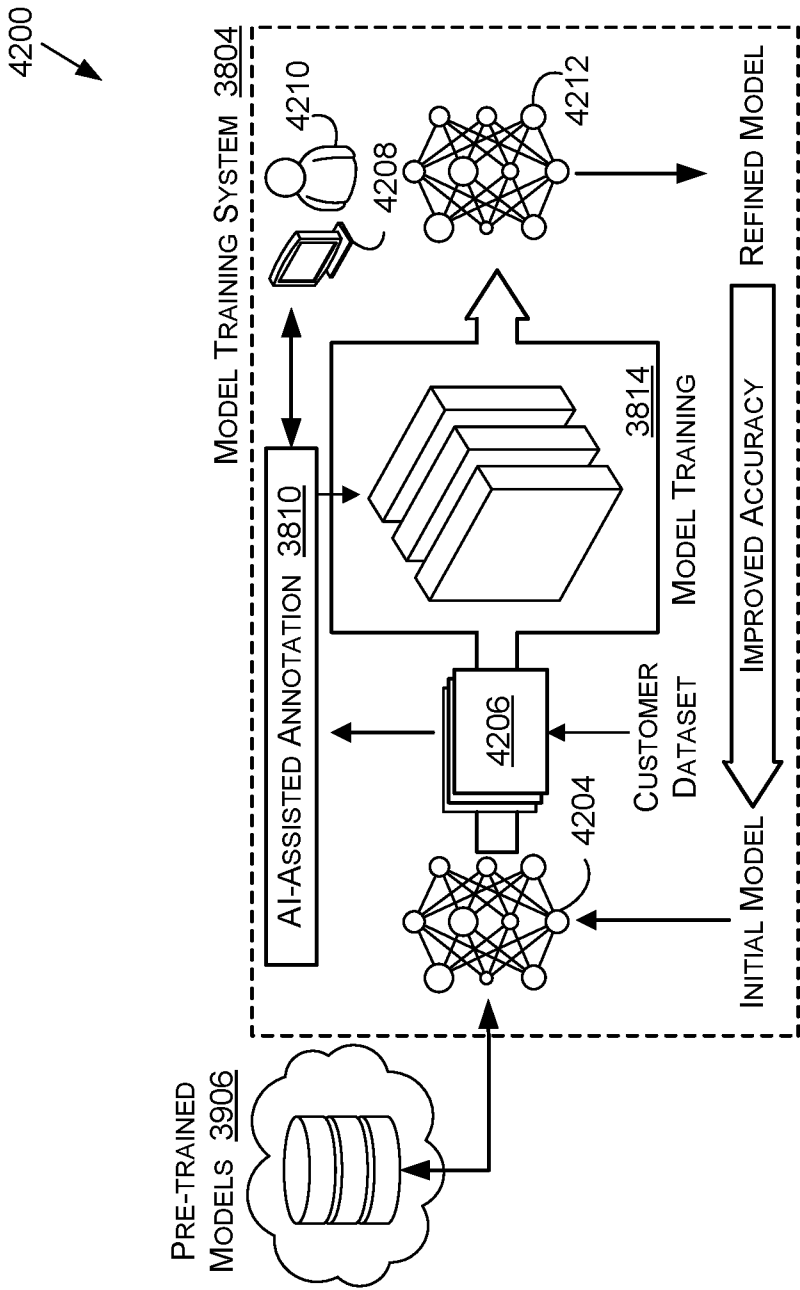


FIG. 42A

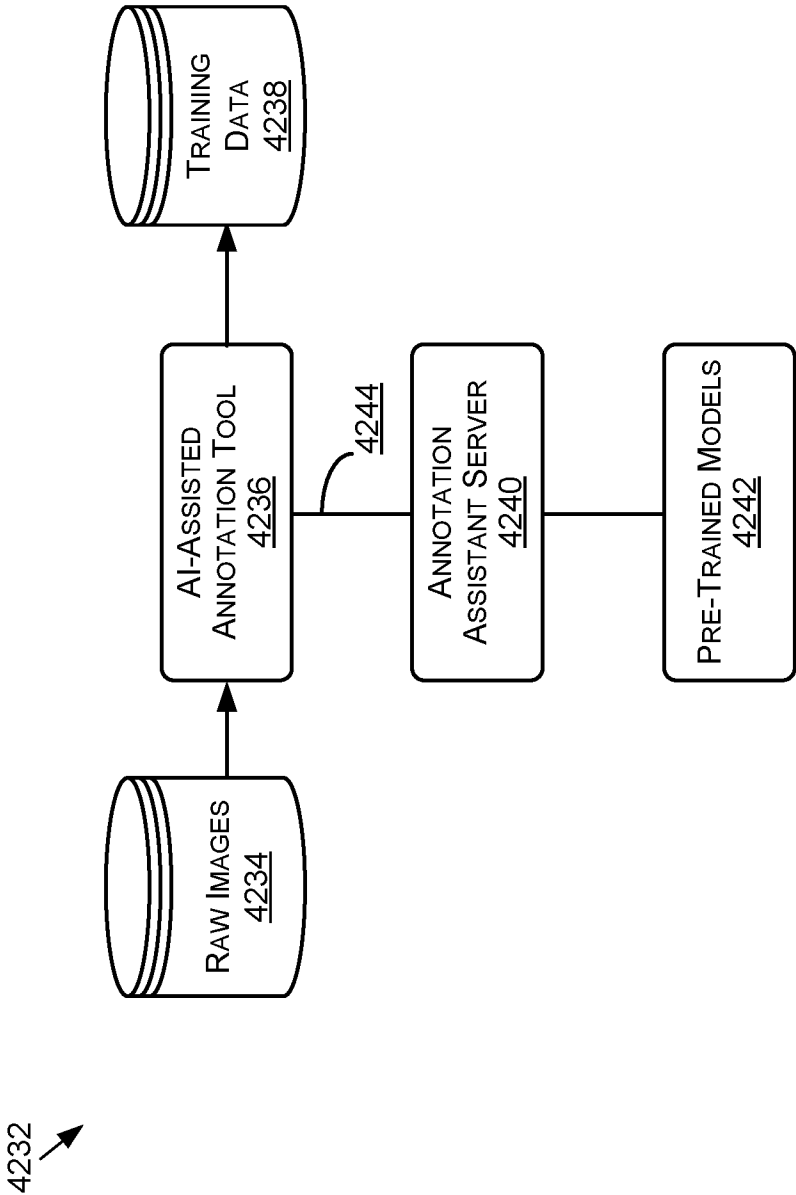


FIG. 42B

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2022/026403

A. CLASSIFICATION OF SUBJECT MATTER

INV. B25J9/16 G05B13/02

ADD.

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)

B25J

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	MANDY XIE ET AL: "Geometric Fabrics for the Acceleration-based Design of Robotic Motion", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 28 October 2020 (2020-10-28), XP081802723, Section VII Section VIII -----	1-24

☐

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

15 July 2022

Date of mailing of the international search report

25/07/2022

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2

NL - 2280 HV Rijswijk

Tel. (+31-70) 340-2040,

Fax: (+31-70) 340-3016

Authorized officer

Lefeure, Guillaume