



(51) International Patent Classification:

G06K 9/62 (2022.01)

G06N 3/04 (2006.01)

(21) International Application Number:

PCT/US2021/049714

(22) International Filing Date:

09 September 2021 (09.09.2021)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

17/020,649

14 September 2020 (14.09.2020) US

(71) Applicant: **NVIDIA CORPORATION** [US/US]; 2788 San Tomas Expressway, Santa Clara, California 95051 (US).

(72) Inventors: **ZHANG, Yuxuan**; 2788 San Tomas Expressway, Santa Clara, California 95051 (US). **LING, Huan**; 2788 San Tomas Expressway, Santa Clara, California 95051 (US). **GAO, Jun**; 2788 San Tomas Expressway, Santa Clara, California 95051 (US). **CHEN, Wenzheng**; 2788 San Tomas Expressway, Santa Clara, California 95051 (US). **BARRIUSO, Antonio Torralba**; 2788 San Tomas Expressway, Santa Clara, California 95051 (US). **FIDLER,**

Sanja; 2788 San Tomas Expressway, Santa Clara, California 95051 (US).

(74) Agent: **KIMES, Benjamin A.** et al.; LOWENSTEIN SANDLER LLP, One Lowenstein Drive, Roseland, New Jersey 07068 (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, 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,

(54) Title: GENERATING LABELS FOR SYNTHETIC IMAGES USING ONE OR MORE NEURAL NETWORKS

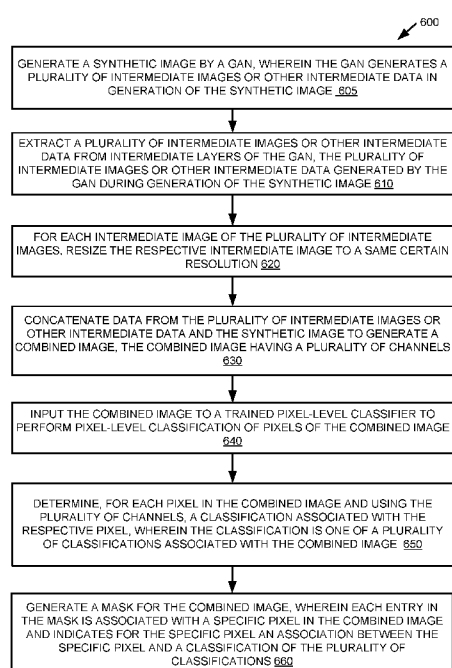


FIG. 6

(57) Abstract: Apparatuses, systems, and techniques to determine pixel-level labels of a synthetic image. In at least one embodiment, the synthetic image is generated by one or more generative networks and the pixel-level labels are generated using a combination of data output by a plurality of layers of the generative networks.

TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

- *as to 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))*
- *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

GENERATING LABELS FOR SYNTHETIC IMAGES USING ONE OR MORE NEURAL NETWORKS

TECHNICAL FIELD

[0001] At least one embodiment pertains to processing resources used to perform and facilitate artificial intelligence. For example, at least one embodiment pertains to processors or computing systems used to train and use neural networks according to various novel techniques described herein.

BACKGROUND

[0002] Semantic segmentation tasks in computer vision can be used in a wide range of applications including self-driving vehicles, robotics, and biomedical image diagnosis. These tasks target prediction of various labels within a given image. Traditionally, thousands of images are manually labeled to train a robust deep learning model in a full supervised approach, which is very expensive and time consuming. Additionally, even when a semi-supervised learning approach is used by traditional solutions, where both labeled and unlabeled images are used to train a deep learning model, hundreds of labeled images are still used to perform training. Thus cost and effort for training a machine learning model continues to be significant.

BRIEF DESCRIPTION OF DRAWINGS

[0003] FIG. 1A illustrates inference and/or training logic, according to at least one embodiment;

[0004] FIG. 1B illustrates inference and/or training logic, according to at least one embodiment;

[0005] FIG. 2 illustrates training and deployment of a neural network, according to at least one embodiment;

[0006] FIG. 3 is a flow diagram of a process to generate a training dataset using a GAN and a pixel-level classifier, in accordance with at least one embodiment;

[0007] FIG. 4 is an example flow diagram for a process to generate a combined image as an output from a GAN generator model, in accordance with at least one embodiment;

[0008] FIG. 5A is an example block diagram for a process 500 to generate a semantic mask containing pixel-level labels of an input image using a pixel-level classifier, in accordance with at least one embodiment;

- [0009] FIG. 5B is a flow diagram of a process of an example pixel-level labeling of parts of an automobile image, in accordance with at least one embodiment;
- [0010] FIG. 6 is a flow diagram illustrating a process of generating pixel-level labels of synthetic images, in accordance with at least one embodiment;
- [0011] FIG. 7 illustrates a flow diagram for a method of training a pixel-level classifier to determine parts of an object within images such as synthetic images generated by one or more GANs, in accordance with an embodiment;
- [0012] FIG. 8 illustrates a flow diagram for a method of training a neural network using a training dataset that includes synthetic images generated by a trained GAN and corresponding masks generated by a trained pixel-level classifier, in accordance with an embodiment;
- [0013] FIG. 9 illustrates an example data center system, according to at least one embodiment;
- [0014] FIG. 10A illustrates an example of an autonomous vehicle, according to at least one embodiment;
- [0015] FIG. 10B illustrates an example of camera locations and fields of view for the autonomous vehicle of FIG. 10A, according to at least one embodiment;
- [0016] FIG. 10C is a block diagram illustrating an example system architecture for the autonomous vehicle of FIG. 10A, according to at least one embodiment;
- [0017] FIG. 10D is a diagram illustrating a system for communication between cloud-based server(s) and the autonomous vehicle of FIG. 10A, according to at least one embodiment;
- [0018] FIG. 11 is a block diagram illustrating a computer system, according to at least one embodiment;
- [0019] FIG. 12 is a block diagram illustrating a computer system, according to at least one embodiment;
- [0020] FIG. 13 illustrates a computer system, according to at least one embodiment;
- [0021] FIG. 14 illustrates a computer system, according to at least one embodiment;
- [0022] FIG. 15A illustrates a computer system, according to at least one embodiment;
- [0023] FIG. 15B illustrates a computer system, according to at least one embodiment;
- [0024] FIG. 15C illustrates a computer system, according to at least one embodiment;
- [0025] FIG. 15D illustrates a computer system, according to at least one embodiment;
- [0026] FIG. 15E and 15F illustrate a shared programming model, according to at least one embodiment;
- [0027] FIG. 16 illustrates exemplary integrated circuits and associated graphics processors, according to at least one embodiment;

- [0028] FIGS. 17A-17B illustrate exemplary integrated circuits and associated graphics processors, according to at least one embodiment;
- [0029] FIGS. 18A-18B illustrate additional exemplary graphics processor logic according to at least one embodiment;
- [0030] FIG. 19 illustrates a computer system, according to at least one embodiment;
- [0031] FIG. 20A illustrates a parallel processor, according to at least one embodiment;
- [0032] FIG. 20B illustrates a partition unit, according to at least one embodiment;
- [0033] FIG. 20C illustrates a processing cluster, according to at least one embodiment;
- [0034] FIG. 20D illustrates a graphics multiprocessor, according to at least one embodiment;
- [0035] FIG. 21 illustrates a multi-graphics processing unit (GPU) system, according to at least one embodiment;
- [0036] FIG. 22 illustrates a graphics processor, according to at least one embodiment;
- [0037] FIG. 23 is a block diagram illustrating a processor micro-architecture for a processor, according to at least one embodiment;
- [0038] FIG. 24 illustrates a deep learning application processor, according to at least one embodiment;
- [0039] FIG. 25 is a block diagram illustrating an example neuromorphic processor, according to at least one embodiment;
- [0040] FIG. 26 illustrates at least portions of a graphics processor, according to one or more embodiments;
- [0041] FIG. 27 illustrates at least portions of a graphics processor, according to one or more embodiments;
- [0042] FIG. 28 illustrates at least portions of a graphics processor, according to one or more embodiments;
- [0043] FIG. 29 is a block diagram of a graphics processing engine of a graphics processor in accordance with at least one embodiment;
- [0044] FIG. 30 is a block diagram of at least portions of a graphics processor core, according to at least one embodiment;
- [0045] FIGS. 31A-31B illustrate thread execution logic including an array of processing elements of a graphics processor core according to at least one embodiment;
- [0046] FIG. 32 illustrates a parallel processing unit (“PPU”), according to at least one embodiment;

[0047] FIG. 33 illustrates a general processing cluster (“GPC”), according to at least one embodiment;

[0048] FIG. 34 illustrates a memory partition unit of a parallel processing unit (“PPU”), according to at least one embodiment;

[0049] FIG. 35 illustrates a streaming multi-processor, according to at least one embodiment.

[0050] FIG. 36 is an example data flow diagram for an advanced computing pipeline, in accordance with at least one embodiment;

[0051] FIG. 37 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;

[0052] FIG. 38 includes an example illustration of a deployment pipeline for processing imaging data, in accordance with at least one embodiment;

[0053] FIG. 39A includes an example data flow diagram of a virtual instrument supporting an ultrasound device, in accordance with at least one embodiment; and

[0054] FIG. 39B includes an example data flow diagram of a virtual instrument supporting a CT scanner, in accordance with at least one embodiment.

DETAILED DESCRIPTION

INFERENCE AND TRAINING LOGIC

[0055] FIG. 1A illustrates inference and/or training logic 115 used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided below in conjunction with FIGS. 1A and/or 1B.

[0056] In at least one embodiment, inference and/or training logic 115 may include, without limitation, code and/or data storage 101 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 115 may include, or be coupled to code and/or data storage 101 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) or simply circuits). 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 101 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 101 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0057] In at least one embodiment, any portion of code and/or data storage 101 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 101 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 data storage 101 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.

[0058] In at least one embodiment, inference and/or training logic 115 may include, without limitation, a code and/or data storage 105 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 embodiment, code and/or data storage 105 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 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 115 may include, or be coupled to code and/or data storage 105 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)).

[0059] 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 105 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 105 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 105 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 105 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.

[0060] In at least one embodiment, code and/or data storage 101 and code and/or data storage 105 may be separate storage structures. In at least one embodiment, code and/or data storage 101 and code and/or data storage 105 may be a combined storage structure. In at least one embodiment, code and/or data storage 101 and code and/or data storage 105 may be partially combined and partially separate. In at least one embodiment, any portion of code and/or data storage 101 and code and/or data storage 105 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0061] In at least one embodiment, inference and/or training logic 115 may include, without limitation, one or more arithmetic logic unit(s) ("ALU(s)") 110, including integer and/or floating point units, to perform logical and/or mathematical operations based, at least in part on, or indicated by, training and/or inference code (e.g., graph code), a result of which may produce activations (e.g., output values from layers or neurons within a neural network) stored in an activation storage 120 that are functions of input/output and/or weight parameter data stored in code and/or data storage 101 and/or code and/or data storage 105. In at least one embodiment, activations stored in activation storage 120 are generated according to linear algebraic and or matrix-based mathematics performed by ALU(s) 110 in response to performing instructions or other code, wherein weight values stored in code and/or data storage 105 and/or data storage 101 are used as operands along with other values, such as bias values, gradient information, momentum values, or other parameters or hyperparameters, any or all of which may be stored in code and/or data storage 105 or code and/or data storage 101 or another storage on or off-chip.

[0062] In at least one embodiment, ALU(s) 110 are included within one or more processors or other hardware logic devices or circuits, whereas in another embodiment, ALU(s) 110 may be external to a processor or other hardware logic device or circuit that uses them (e.g., a co-

processor). In at least one embodiment, ALUs 110 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 101, code and/or data storage 105, and activation storage 120 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 120 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, 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.

[0063] In at least one embodiment, activation storage 120 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, activation storage 120 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 120 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.

[0064] In at least one embodiment, inference and/or training logic 115 illustrated in FIG. 1A 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 115 illustrated in FIG. 1A 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").

[0065] FIG. 1B illustrates inference and/or training logic 115, according to at least one embodiment. In at least one embodiment, inference and/or training logic 115 may include, without limitation, hardware logic in which computational resources are dedicated or 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 115 illustrated in FIG. 1B 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 Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, inference and/or training logic 115 illustrated in FIG. 1B 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 115 includes, without limitation, code and/or data storage 101 and code and/or data storage 105, 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. 1B, each of code and/or data storage 101 and code and/or data storage 105 is associated with a dedicated computational resource, such as computational hardware 102 and computational hardware 106, respectively. In at least one embodiment, each of computational hardware 102 and computational hardware 106 comprises one or more ALUs that perform mathematical functions, such as linear algebraic functions, only on information stored in code and/or data storage 101 and code and/or data storage 105, respectively, result of which is stored in activation storage 120.

[0066] In at least one embodiment, each of code and/or data storage 101 and 105 and corresponding computational hardware 102 and 106, respectively, correspond to different layers of a neural network, such that resulting activation from one storage/computational pair 101/102 of code and/or data storage 101 and computational hardware 102 is provided as an input to a next storage/computational pair 105/106 of code and/or data storage 105 and computational hardware 106, in order to mirror a conceptual organization of a neural network. In at least one embodiment, each of storage/computational pairs 101/102 and 105/106 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 101/102 and 105/106 may be included in inference and/or training logic 115.

NEURAL NETWORK TRAINING AND DEPLOYMENT

[0067] FIG. 2 illustrates training and deployment of a deep neural network, according to at least one embodiment. In at least one embodiment, untrained neural network 206 is trained

using a training dataset 202. In at least one embodiment, the training dataset 202 is generated using the techniques set forth hereinbelow. In one embodiment, the training dataset 202 is generated using a generative adversarial network (GAN) that generates synthetic images and an associated trained neural network that generates labels for synthetic images generated by the GAN. In at least one embodiment, training framework 204 is a PyTorch framework, whereas in other embodiments, training framework 204 is a TensorFlow, Boost, Caffè, Microsoft Cognitive Toolkit/CNTK, MXNet, Chainer, Keras, Deeplearning4j, or other training framework. In at least one embodiment, training framework 204 trains an untrained neural network 206 and enables it to be trained using processing resources described herein to generate a trained neural network 208. 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.

[0068] In at least one embodiment, untrained neural network 206 is trained using supervised learning, wherein training dataset 202 includes an input paired with a desired output for an input, or where training dataset 202 includes input having a known output and an output of neural network 206 is manually graded. In at least one embodiment, untrained neural network 206 is trained in a supervised manner and processes inputs from training dataset 202 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 206. In at least one embodiment, training framework 204 adjusts weights that control untrained neural network 206. In at least one embodiment, training framework 204 includes tools to monitor how well untrained neural network 206 is converging towards a model, such as trained neural network 208, suitable to generating correct answers, such as in result 214, based on input data such as a new dataset 212. In at least one embodiment, training framework 204 trains untrained neural network 206 repeatedly while adjusting weights to refine an output of untrained neural network 206 using a loss function and adjustment algorithm, such as stochastic gradient descent. In at least one embodiment, training framework 204 trains untrained neural network 206 until untrained neural network 206 achieves a desired accuracy. In at least one embodiment, trained neural network 208 can then be deployed to implement any number of machine learning operations.

[0069] In at least one embodiment, untrained neural network 206 is trained using unsupervised learning, wherein untrained neural network 206 attempts to train itself using unlabeled data. In at least one embodiment, unsupervised learning training dataset 202 will include input data without any associated output data or “ground truth” data. In at least one

embodiment, untrained neural network 206 can learn groupings within training dataset 202 and can determine how individual inputs are related to untrained dataset 202. In at least one embodiment, unsupervised training can be used to generate a self-organizing map in trained neural network 208 capable of performing operations useful in reducing dimensionality of new dataset 212. In at least one embodiment, unsupervised training can also be used to perform anomaly detection, which allows identification of data points in new dataset 212 that deviate from normal patterns of new dataset 212.

[0070] In at least one embodiment, semi-supervised learning may be used, which is a technique in which in training dataset 202 includes a mix of labeled and unlabeled data. In at least one embodiment, training framework 204 may be used to perform incremental learning, such as through transferred learning techniques. In at least one embodiment, incremental learning enables trained neural network 208 to adapt to new dataset 212 without forgetting knowledge instilled within trained neural network 208 during initial training.

GENERATING TRAINING DATASETS AND GENERATING LABELS FOR SYNTHETIC IMAGES

[0071] Pixel-level segmentation tasks can be used in a wide range of applications including self-driving vehicles, robotics, and biomedical image diagnosis. These tasks target prediction of numerous labels within a given image. Traditionally, thousands of images are manually labeled to train a robust deep learning model in a full supervised approach, to perform pixel-level labeling or segmentation. Generating a large number of labeled images manually can be very expensive and time consuming. Additionally, even when a semi-supervised approach is used in traditional approaches, where both labeled and unlabeled images can be used to train a deep learning model, hundreds of labeled images are still used for training.

[0072] FIG. 3 is a flow diagram of a process 300 to generate a training dataset using a generative network or generative model and a pixel-level classifier or neural network trained to output a classification, prediction, regression target and/or other output based on processing of a synthetic image output by generative network or generative model, in accordance with at least one embodiment. In at least one embodiment, a training dataset may be generated for use in training a deep learning model.

[0073] At operation 310, processing logic generates a set of synthetic images using a trained generative network or generative model. In at least one embodiment, a generative network or generative model has been trained to generate a particular type of synthetic image. In at least one embodiment, a type of synthetic images that a generative network or

generative model is trained to generate are one of automobile images, medical images, facial images, images of animals, images of buildings, images of street scenes, images of street signage, or another type of image. In at least one embodiment, a type of medical images that a generative network or generative model is trained to generate includes one of x-ray images, cone beam computed tomography (CBCT) scan slices, panoramic x-ray images, ultrasound images, magnetic resonance imaging (MRI) images and so on of patient anatomy.

[0074] In at least one embodiment, a generative network is a generative adversarial network (GAN). In at least one embodiment, a generative network is a normalizing flow. In at least one embodiment, a generative model is a latent dirichlet allocation, a naïve Bayes network, a Gaussian mixture model, a restricted Boltzmann machine, normalizing flows, or a VAE (variational autoencoders). In at least one embodiment, a generative network is a Style Generative Adversarial Network (StyleGAN). StyleGAN is an extension to a GAN architecture to give control over disentangled style properties of generated images.

[0075] In at least one embodiment, a generative network is a generative adversarial network (GAN) that includes a generator model and a discriminator model, where a generator model includes use of a mapping network to map points in latent space to an intermediate latent space, includes use of intermediate latent space to control style at each point in a generator model, and uses introduction of noise as a source of variation at each point in a generator model. A resulting generator model is capable not only of generating impressively photorealistic high-quality synthetic images, but also offers control over a style of generated images at different levels of detail through varying style vectors and noise. In at least one embodiment, a generator starts from a learned constant input and adjusts a “style” of an image at each convolution layer based on a latent code, therefore directly controlling a strength of image features at different scales.

[0076] In at least one embodiment, a StyleGAN generator uses two sources of randomness used to generate a synthetic image: a standalone mapping network and noise layers, in addition to a starting point from latent space. An output from a mapping network is a vector that defines styles that is integrated at each point in a generator model via a layer called adaptive instance normalization. Use of this style vector gives control over style of a generated image. In at least one embodiment, stochastic variation is introduced through noise added at each point in a generator model. Noise is added to entire feature maps that allow a model to interpret a style in a fine-grained, per-pixel manner. This per-block incorporation of style vector and noise allows each block to localize both an interpretation of style and a stochastic variation to a given level of detail.

[0077] In at least one embodiment, a generator of a GAN, which may be an implementation of StyleGAN, includes multiple intermediate layers and a final layer, each of which generate a version of a synthetic image. In at least one embodiment, each layer generates a feature map for a version of synthetic image. In at least one embodiment, each subsequent version of an image generated by a later layer is a more detailed or refined image version. In at least one embodiment, data from a final version of a synthetic image generated by a final layer of a generator model is combined with data from intermediate versions of a synthetic image generated by intermediate layers of a generator network. In at least one embodiment, a feature map of a final version of a synthetic image is combined with feature maps of intermediate image versions. In at least one embodiment, data from different versions of an image is combined to generate a combined image, which may be or include a combined feature map. In at least one embodiment, feature maps of each intermediate version of a synthetic image include channel data for multiple channels, such as color channels, hue channels, lightness channels and so on. Feature maps of different versions of a synthetic image may include different numbers of channels. In at least one embodiment, a total number of channels in a combined image or combined feature map is based on a sum of channels of feature maps from versions of an image that have been combined.

[0078] In at least one embodiment, for each synthetic image in a training dataset a generative network or generative model also generates associated channel data. In at least one embodiment, for each synthetic image in a training dataset, a trained generative network or generative model generates a number of intermediate images or intermediate feature maps in a process of generating a synthetic image, and associated channel data is generated based on a combination of channel data associated with each intermediate image or intermediate feature map associated with that synthetic image.

[0079] In at least one embodiment, at operation 320, processing logic processes synthetic images using a trained neural network or other trained machine learning model that has been trained to generate one or more labels, one or more predictions, one or more regression targets and/or one or more other outputs based on processing of one or more synthetic images generated by a generative model or generative network. In at least one embodiment, a trained neural network is a trained pixel-level classifier that has been trained to assign pixel-level labels to objects in synthetic images of a type generated by a generative network or generative model. In at least one embodiment, a trained pixel-level classifier is a trained neural network. In at least one embodiment, a trained pixel-level classifier is a shallow neural network having 1-2 hidden layers. In at least one embodiment, for each synthetic image input

into a pixel level classifier, it outputs a probability map. A probability map comprises, for each pixel in a synthetic image, a separate probability of a pixel representing each class. In at least one embodiment, a probability map comprises a first probability that a pixel belongs to a first class and a second probability that a pixel belongs to a second class. In at least one embodiment, probability map comprises a first probability that a pixel belongs to a first class, a second probability that a pixel belongs to a second class, and a third probability that a pixel belongs to a third class. In at least one embodiment, a probability map comprises a first probability that a pixel belongs to a first class, a second probability that a pixel belongs to a second class, a third probability that a pixel belongs to a third class, and a fourth probability that a pixel belongs to a fourth class. In at least one embodiment, a probability map comprises a first probability that a pixel belongs to a first class, a second probability that a pixel belongs to a second class, a third probability that a pixel belongs to a third class, a fourth probability that a pixel belongs to a fourth class, and a fifth probability that a pixel belongs to a fifth class. In at least one embodiment, a pixel level classifier is trained to identify more than five classes. In at least one embodiment, a pixel-level classifier is trained to output a single class for each pixel in a processed synthetic image.

[0080] In at least one embodiment, a pixel level classifier generates a corresponding mask for each synthetic image. A mask output by pixel-level classifier can be a semantic mask with a number of entries, such that each entry in a mask is associated with a specific pixel in a corresponding synthetic image and indicates for a specific pixel a specific label from a set of labels. A label can indicate a part of an image to which a pixel belongs, as explained in more details herein below. In at least one embodiment, a mask is generated based on a probability map or pixel level classes assigned to each pixel. In at least one embodiment, a probability map may include probabilities of each pixel belonging to each possible class, and processing logic may apply thresholds to that probability map to determine labels based on determined probabilities. In at least one embodiment, for each pixel a label is selected based on one of a set of classes having a probability that meets or exceeds a probability threshold. In at least one embodiment, for each pixel a label is selected based on one class having a highest probability among a set of possible classes.

[0081] In at least one embodiment, a probability map or mask has a size that is equal to a pixel size of synthetic image. In at least one embodiment, each point or pixel in a probability map or mask may include a probability value that indicates a probability that a point represents one or more classes. In at least one embodiment, there may be three classes, including a first class representing a first type of object, a second class representing a second

type of object, and a third class representing a third type of object. Points that have a first class may have a value of (1,0,0) (100% probability of first class and 0% probability of second and dental classes), points that have a second class may have a value of (0,1,0), and points that have a third class may have a value of (0,0,1), for example. In at least one embodiment, a probability map or mask generated for each synthetic image is attached to or otherwise associated with a respective synthetic image.

[0082] In at least one embodiment, a trained machine learning model generates image-level classifications for processed synthetic images. In at least one embodiment, a trained machine learning model determines key points and generates key-point classifications for synthetic images. In at least one embodiment, key-point classifications label regions or groups of pixels as being particular classes of key points. In at least one embodiment, a trained machine learning model determines bounding boxes within synthetic images and labels such bounding boxes. In at least one embodiment, a trained machine learning model generates regression targets for synthetic images, regions of synthetic images and/or pixels of synthetic images. In at least one embodiment, a trained machine learning model outputs predictions for synthetic images and/or pixels or regions of synthetic images. In at least one embodiment, a trained machine learning model is trained to generate other types of outputs for synthetic images.

[0083] Pixel-level labeling is described for one or more embodiments described herein. It should be understood that other types of labeling or classifications can be performed, in addition to pixel-level labeling. In at least one embodiment, rather than performing pixel-level labeling, processing logic generates image-level classifications, key point classifications that can label regions of images rather than individual pixels, classifications based on bounding boxes within an image, and/or classifications based on characteristics and properties of an image or materials in an image. In at least one embodiment, it should further be understood that other types of outputs such as predictions and/or regression targets can also be generated by a trained machine learning model.

[0084] In at least one embodiment, a generative network or generative model is used to generate videos. In at least one embodiment, processing logic uses a trained machine learning model to generate classifications and/or labels of temporal data associated with video generated by a generative network. In at least one embodiment, processing logic uses a trained machine learning model to track objects between frames of a video.

[0085] In at least one embodiment, at operation 330, processing logic generates a training dataset that includes synthetic images generated by a generative network or generative model and corresponding masks. In at least one embodiment, a training dataset further includes real

images with associated labels. In at least one embodiment, a training dataset further includes one or more unlabeled real images.

[0086] In at least one embodiment, at operation 340, process 300 uses a training dataset to train one or more additional machine learning models to perform segmentation or pixel level labeling. In at least one embodiment, one or more additional machine learning models include an artificial neural network or a deep network. In at least one embodiment, a training dataset that is generated according to method 300 includes only labeled images, and supervised machine learning is performed to train one or more additional machine learning models. In at least one embodiment, a training dataset includes some unlabeled images, and semi-supervised machine learning is performed to train one or more additional machine learning models.

[0087] FIG. 4 is an example flow diagram for a process 400 to generate a combined image or combined feature map as an output from a GAN generator model 410, in accordance with at least one embodiment. In at least one embodiment, process 400 is performed for a synthetic image at operation 310 of process 300 after synthetic image has been generated by a GAN generator model 410. In at least one embodiment, a GAN generator model 410 is configured to output a combined image 440 rather than simply a final version 433 of a synthetic image. In at least one embodiment, process 400 may be performed by inference and/or training logic 115. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGs. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 1B 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.

[0088] Referring back to FIG. 4, a GAN generator model 410 generates multiple versions of a synthetic image 420. In at least one embodiment, GAN generator model 410 includes a generator model with multiple layers, including layer 1 415 through layer M 416 and layer M 416 through layer z 417 for generating synthetic images. In at least one embodiment, GAN generator model 410 includes 16 layers for generating synthetic images. In at least one embodiment, each layer generates data output including a feature map representing a corresponding version of synthetic image 420, also referred to herein as intermediate images or intermediate feature maps. As shown, layer 1 415 generates image 421, layer M 416 generates image 422 and layer z 417 generates final synthetic image 423. In at least one embodiment, version of the synthetic image generated by a particular layer such as layer 415

or layer 416 is input a subsequent layer of the GAN generator model 410. A subsequent layer uses data from received synthetic image version to generate a next synthetic image version. Ultimately, a final synthetic image 423 is generated by layer z 417. In at least one embodiment, rather than outputting only final synthetic image 423, GAN generator model 410 outputs each of intermediate versions of synthetic image 421, 422 as well as final synthetic image 423. Each version of image 420 may contain different data that can be useful in identifying labels in final synthetic image.

[0089] In at least one embodiment, each of versions of image 420 may be or include a feature map. In at least one embodiment, each feature map has a certain resolution of that image and a number of channels of image, each channel containing information about image. A channel of image can store information about image including color, hue, lightness, and the like. For example, an RGB image has three channels; red, green, and blue, an HSL image has three channels: hue, saturation, and lightness, and so on. In at least one embodiment, each layer 415-417 of GAN generator model 410 generates an image 421-423 that has an associated resolution and number of channels. In at least one embodiment, layer 415 of GAN generator model 410 can generate intermediate image or intermediate feature map 421 having resolution 4x4 and 512 channels. Layer 416 of GAN generator model 410 can generate intermediate image or intermediate feature map 422 having a higher resolution of 256x256 and 64 channels, and layer 417 can generate intermediate image or intermediate feature map 423 having yet a higher resolution than previous two layers. In at least one embodiment, a resolution of image 423 can be 512x512 and a number of channels of image 423 can be 32.

[0090] In at least one embodiment, after generation of intermediate images or intermediate feature maps 420, processing logic such as an image pre-processor can resize one or more of synthetic image versions 420, such that each synthetic image version has a same resolution. Resizing each image to same resolution can enable data from different versions of synthetic image 420 to be concatenated together to generate combined image or combined feature map 440. In at least one embodiment, resizing an image included adding additional pixels to image and determining channel information for each of added pixels based on channel information of surrounding pixels.

[0091] In at least one embodiment, at operation 412A, processing logic can resize image 421 from a 4x4 resolution to a 512x512 resolution to result in image 431. Similarly, at operation 412B, processing logic can resize image 422 from a 256x256 resolution to a 512x512 resolution to produce image 432. In at least one embodiment, final synthetic image

423 has a highest resolution among synthetic image versions 420, and final synthetic image 423 is not resized. In at least one embodiment, processing logic can keep number of channels unchanged when resizing synthetic image versions 420.

[0092] In at least one embodiment, each intermediate image or intermediate feature map includes certain data related to final synthetic image. Thus, concatenating intermediate images or intermediate feature maps (various synthetic image versions 420) can result in combining data from each image into combined image. In at least one embodiment, combined data can be represented by a number of channels included in combined image or combined feature map. In at least one embodiment, channel information from each of resized images 431, 432, 433 is added together. Combined image or combined feature map may include a sum of number of channels of each of resized images 430. For example, image 431 may include 512 channels, image 432 may include 64 channels and image 433 may include 32 channels. These channels may be added together with channels from other layers to form combined image 440. In at least one embodiment, as illustrated, combined image 440 includes 5056 channels, where each pixel has values for each of 5056 channels, and where each of channels is from one of resized images 430.

[0093] In at least one embodiment, at operation 414, processing logic combines resized images 430, such as image 431, image 432 and final synthetic image 423 into combined image 440. In at least one embodiment, combining images or feature maps into combined image or combined feature map 440 can result in combined image having same resolution (512x512) as that of each intermediate image or intermediate feature map whereas number of channels of combined image can be sum of number of channels of each resized image 431-432 and final synthetic image 423. In at least one embodiment, number of channels for combined image 440 can be 5056 channels, combining channels from each intermediate image or intermediate feature map generated by each layer of 16 layers of GAN 420. In at least one embodiment, combined image 440 can then be used as an input to one or more neural network that has been trained to perform pixel-level labeling for combined images or combined feature maps generated by GAN generator model 410, as described with reference to operation 320 of process 300.

[0094] FIG. 5A is an example block diagram for a process 500 to generate a semantic mask containing pixel-level labels of an input image using a pixel-level classifier, in accordance with at least one embodiment. In at least one embodiment, process 500 may be performed by inference and/or training logic 115. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGs. 1A and/or 1B. In at least one embodiment,

inference and/or training logic 115 may be used in system FIG. 1B 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.

[0095] Referring back to FIG. 5A, in at least one embodiment, at operation 510 a generator network of a GAN 505 outputs multiple versions of a synthetic image 515. multiple versions of synthetic image include intermediate versions as well as a final version of synthetic image. In at least one embodiment, rather than versions of a synthetic image, GAN generator model 505 outputs other intermediate data from different layers of GAN generator model 505. For example, GAN generator model 505 may include multiple layers each having one or more nodes that output data that is input to one or more nodes of a next layer, where such data is something other than an intermediate image or intermediate feature map or data from an intermediate image or intermediate feature map. In at least one embodiment, such intermediate data may be collected during generation of a synthetic image by GAN generator model 505.

[0096] In at least one embodiment, at operation 520 an image pre-processor 525 processes versions of synthetic image 515 to produce a combined image or combined feature map 530. In at least one embodiment, image pre-processor 525 performs operations for generating combined image or combined feature map 440 described with reference to FIG. 4. In at least one embodiment, image pre-processor 525 resizes one or more of versions of synthetic image 515 and combines channel information of each of resized versions of synthetic image to produce combined image 530.

[0097] In at least one embodiment, such as where GAN generator model 505 is not StyleGAN, at operation 520 image pre-processor 525 processes a single synthetic image and intermediate data generated by nodes of intermediate layers of GAN generator model that was collected at operation 510. In at least one embodiment, image pre-processor 525 adds additional extracted data from various layers to synthetic image as one or more additional channels of data.

[0098] At operation 560, combined image or combined feature map 540 is input to pixel-level classifier 555. In at least one embodiment, pixel-level classifier 555 was trained to generate pixel-level labels for combined images that are based on synthetic images generated by GAN generator model 505. In at least one embodiment, pixel-level classifier 555 can generate pixel-level labels for each pixel in combined image 540. Combined image or combined feature map 540 can be same as or similar to combined image 440 of FIG. 4. In at

least one embodiment, pixel-level classifier 555 can determine a classification for each pixel in combined image 540. In at least one embodiment, pixel-level classifier 555 can use information provided by channels of combined image 540 in determining classification of each pixel of combined image 540. Classification can be one of a set of predefined classifications corresponding to possible contents of combined image or combined feature map 540, as explained in more details herein below.

[0099] In at least one embodiment, pixel-level classifier 555 can generate, as an output, semantic mask 556 that has a set of entries corresponding to set of pixels in combined image or combined feature map 540. For example, a combined image 540 having a resolution of 512x512 has 262,144 pixels. In this case, semantic mask 556 can include 262,144 entries, each entry corresponding to a pixel of combined image 540 and indicating a classification for corresponding pixel. Accordingly, semantic mask 556 can specify for each pixel of combined image or combined feature map 540 specific label or class pixel belongs to. In at least one embodiment, processing logic uses trained neural network 208 of FIG. 2 as pixel-level classifier 555, to obtain result 214 that includes semantic mask 556.

[0100] In at least one embodiment, processing logic can use semantic masks generated by pixel-level classifier 555 and corresponding synthetic images generated by GAN 520 to train an additional machine learning model such as a deep neural network to perform pixel-level segmentation or classification of synthetic images.

[0101] FIG. 5B is a flow diagram of a process 560 of an example pixel-level labeling of parts of an automobile image, in accordance with at least one embodiment. Process 600, at operation 564, can use a generative adversarial networks (GAN) 562 to generate a synthetic image of an automobile 566. In at least one implementation, GAN 562 can be a StyleGAN network. A StyleGAN is a network that provides automatically learned, unsupervised separation of high-level attributes and stochastic variation in generated images, enabling scale-specific control of synthesis. In at least one embodiment, GAN 562 can generate automobile image 566 using a combination of data output by multiple layers of GAN 564. In at least one embodiment, GAN 562 also generates channel data 568 associated with automobile image 566. Channel data 568 can include information about automobile image 566 including color, hue, lightness, and the like as extracted from multiple intermediate versions of automobile image 566 generated by GAN 562 during generation of automobile image 566. In at least one embodiment, at operation 570, processing logic can provide automobile image 566 and channel data 568 as inputs to pixel-level classifier 572, in order to generate labels of different parts of an automobile in automobile image 566. In at least one

embodiment, pixel-level classifier 572 can determine, for each pixel in automobile image 566, using channel data 568, a classification of pixel corresponding to specific part of automobile to which pixel belongs.

[0102] In at least one embodiment, because channel data 568 includes channels from each of a set of intermediate images or intermediate feature maps that are generated by GAN 562 during generation of automobile image 566, channel data 568 can include information about each of intermediate images or intermediate feature maps. Each intermediate image or intermediate feature map can represent a distinct stage in process of creating automobile image 566, and thus can enable determination of various parts of automobile that can be developed with more clarity at each stage of generation of automobile image 566.

Accordingly, processing logic can use additional information about automobile image 566 that is stored in channel data 568 to predict classification of each pixel of automobile image 566. For example, classification of pixel can be an identifier indicating whether pixel is of a mirror, a bumper, a hood, a door, or a window of automobile within automobile image 566. In at least one embodiment, identifiers and corresponding classifications associated with an automobile image can be # 0 background, # 1 car body, # 2 bumper, # 3 tail bumper, # 4 head, lights, # 5 taillights, # 6 windshield, # 7 window, # 8 door, # 9 mirror, # 10 handles, # 11 license plate, # 12 grilles, # 13 wheel, # 14 hubcap, # 15 running boards, # 16 roof, # 17 trunk lids, # 18 hoods, and # 19 fender. Other parts of an automobile can also be identified as a classification for pixels of an automobile image.

[0103] In at least one embodiment, at operation 574, pixel-level classifier 572 generates a mask 580 for automobile image 566, such that each entry in 580 618 is associated with a specific pixel in automobile image 566 and indicates for specific pixel an association between specific pixel and a classification representing part for which specific pixel belongs. In this case, mask 580 has a number of entries that is equal to number of pixels in automobile image 566, as explained in more details herein. In at least one embodiment, processing logic can use GAN 562 and pixel-level 572 to generate a dataset containing multiple automobile images generated by GAN 562 and corresponding masks generated by pixel-level classifier 572 to train an additional machine learning model to perform pixel-level segmentation of automobile images.

[0104] FIG. 5B has been described with reference to a particular example of segmenting an automobile, in accordance with at least one embodiment. In at least one embodiment, GAN 562 may be trained to generate other types of synthetic images other than automobiles, and pixel-level classifier 572 may be trained to determine labels for such other types of images

generated by GAN 562. In at least one embodiment, GAN 562 and pixel-level classifier 572 are trained to generate and label, respectively, medical images of human anatomy, medical images of animal anatomy, other types of medical images, images of streets, images of buildings, images of manufactured products, images of nature scenes, images of human faces, and/or other types of images. In at least one embodiment, pixel-level classifier 572 is replaced by a trained machine learning model trained to generate another type of output based on synthetically generated medical images of human anatomy, medical images of animal anatomy, other types of medical images, images of streets, images of buildings, images of manufactured products, images of nature scenes, images of human faces, and/or other types of images generated by GAN 562 or another type of generative network or generative model. In at least one embodiment, a trained machine learning model is trained to automatically modify an input synthetic image, such as by applying one or more types of makeup to faces in synthetic images.

[0105] FIG. 6 is a flow diagram illustrating a process 600 of generating pixel-level labels of synthetic images, in accordance with at least one embodiment. In at least one embodiment, processing logic uses one or more neural network such as a GAN generator model to generate a synthetic image. In at least one embodiment, neural network(s) can include multiple layers for generating synthetic image, where each layer generates data output including an intermediate image and/or other intermediate data. Each intermediate image or intermediate feature map can have a certain resolution and a number of channels. At operation 610, processing logic can extract intermediate images and/or other intermediate data generated by each of layers of neural network. In at least one embodiment, processing logic obtains channel data associated with each intermediate image or intermediate feature map, which can be combined and used in prediction of pixel-level labels of synthetic image, as noted herein above. In at least one embodiment, different intermediate images or intermediate feature maps can have different resolutions and/or a different number of channels storing information about corresponding image.

[0106] In at least one embodiment, at operation 620, processing logic resizes each intermediate image or intermediate feature map, such that each intermediate image or intermediate feature map has same resolution. Resizing each image to same resolution can enable processing logic to concatenate intermediate images or intermediate feature maps together to generate a combined image or combined feature map that can be used as an input to a pixel-level classifier. In at least one embodiment, processing logic can determine an intermediate image or intermediate feature map having a maximum resolution and then can

resize each other intermediate image or intermediate feature map to have same resolution as maximum resolution.

[0107] In at least one embodiment, at operation 630, processing logic concatenates data from resized images and synthetic image to generate a combined image or combined feature map. In at least one embodiment, processing logic combines intermediate data extracted from intermediate layers of GAN generator network to generated synthetic image. In at least one embodiment, intermediate data is added as additional channels. In at least one embodiment, each intermediate image and/or other intermediate data can include certain data related to final synthetic image. Accordingly concatenating intermediate images and/or adding intermediate data to final synthetic image can provide additional relevant data usable to improve an accuracy of process of label prediction of each pixel in image. In at least one embodiment, channel data can be represented by sum of channels from each intermediate image or intermediate feature map.

[0108] In at least one embodiment, at operation 640, processing logic uses combined image or combined feature map as an input to a trained pixel-level labeling network to predict labels within combined image. In at least one embodiment, at operation 650, pixel-level classifier can generate a label (or a classification) for each pixel in combined image, based on provided channel data. In at least one embodiment, pixel-level classifier can use information provided by channels of combined image or combined feature map in determining classification of each pixel of combined image. Classification can be one of a set of predefined classifications.

[0109] In at least one embodiment, at operation 660, processing logic uses pixel-level classifier to generate a semantic mask as an output of pixel-level classifier. That mask has a set of entries corresponding to set of pixels in combined image or combined feature map, in order to provide a label for each pixel in combined image or combined feature map, indicating a classification for corresponding pixel.

[0110] FIG. 7 illustrates a flow diagram for a method 700 of training a pixel-level classifier to determine parts of an object within images such as synthetic images generated by one or more GANs, in accordance with an embodiment. At block 702 of method 700, an untrained pixel-level classifier is initialized. In at least one embodiment, pixel-level classifier that is initialized may be a deep learning model such as an artificial neural network. In at least one embodiment, pixel-level classifier is a shallow neural network. Initialization of artificial neural network may include selecting starting parameters for neural network. A solution to a non-convex optimization algorithm depends at least in part on initial parameters, and so initialization parameters should be chosen appropriately. In one embodiment, parameters are

initialized using Gaussian or uniform distributions with arbitrary set variances. In at least one embodiment, artificial neural network is initialized using a Xavier initialization.

[0111] In at least one embodiment, pixel-level classifier that is initialized and then trained is a neural network that is able to locate and detect one or more types of classes of objects in images. Output of neural network may be a set of shapes that closely match each of detected objects, as well as a class output for each detected object. In at least one embodiment, for an image of an automobile, detected objects can include mirror, window, door, wheel, and/or other car parts.

[0112] At block 705, untrained pixel-level classifier receives a set of combined images or combined feature maps from a training dataset. In at least one embodiment, a first combined image or combined feature map may be, for example, unlabeled image 740 along with a corresponding semantic mask or labelled image 750. In at least one embodiment, labeling of labelled image and/or generation of semantic mask may have been performed manually by a user. In at least one embodiment, training dataset includes fewer than 100, fewer than 50, fewer than 40, fewer than 30, or fewer than 20 labeled combined images. In at least one embodiment, training dataset includes 10-20 labeled combined images. In at least one embodiment, supervised learning is performed to train one or more machine learning models to function as a pixel-level classifier using labeled images. In at least one embodiment, training dataset includes a number of unlabeled images as well as labeled images, and semi-supervised learning is performed to train one or more machine learning models to function as a pixel-level classifier. Semantic mask includes entries corresponding to pixels of unlabeled image 740, such that each entry in semantic mask corresponds to a pixel of unlabeled image 740 and associates pixel with a specific label. For example, for a picture of an automobile, labels may include: car body, front bumper, mirror, head lights, grille, license plate, hood, fender, hubcap, wheel, running board, door, handle, tail bumper, roof, windshield, window, etc. of automobile.

[0113] In at least one embodiment, at block 710, processing logic divides combined image or combined feature map into multiple data points, where each data point is usable to train machine learning model such as neural network to perform pixel-level labeling. In at least one embodiment, each data point for a pixel includes channel information associated with that pixel from combined image or combined feature map. At block 715, processing logic selects a data point.

[0114] At block 720, processing logic optimizes parameters of neural network or other machine learning model that will function as a pixel-level classifier from selected data point.

In at least one embodiment, pixel-level classifier applies classifications or labels to pixels of image based on its current parameter values. An artificial neural network includes an input layer that consists of values in a data point, such as channels representing information about pixels in image 740, such as RGB values of pixels. A next layer is called a hidden layer, and nodes at hidden layer each receive one or more of input values. Each node contains parameters or weights to apply to input values. Each node therefore essentially inputs input values into a multivariate function such as a non-linear mathematical transformation to produce an output value. A next layer may be another hidden layer or an output layer. In either case, nodes at next layer receive output values from nodes at previous layer, and each node applies weights to those values and then generates its own output value. This may be performed at each layer. A final layer is output layer, where there is one node for each possible classification of pixels, such as one node for each part of object for which pixel can belong). In at least one embodiment, for artificial neural network being trained, a class is determined for each pixel in image. In at least one embodiment, for each pixel in image, final layer applies a probability that pixel of image belongs to one or more specific classes. For example, a particular pixel may be marked as a first class.

[0115] In at least one embodiment, processing logic compares classification or label (or multiple classifications or labels) for pixel of selected data point to provided classification(s) or label(s) for that pixel or point to determine one or more classification error. An error term or delta may be determined for each node in artificial neural network. Based on this error, artificial neural network adjusts one or more of its parameters for one or more of its nodes, which may include adjusting weights for one or more inputs of a node. Parameters may be updated in a back propagation manner, such that nodes at a highest layer are updated first, followed by nodes at a next layer, and so on. An artificial neural network contains multiple layers of “neurons”, where each layer receives as input values from neurons at a previous layer. Parameters for each neuron include weights associated with values that are received from each of neurons at a previous layer. Accordingly, adjusting parameters may include adjusting weights assigned to each of inputs for one or more neurons at one or more layers in artificial neural network.

[0116] In at least one embodiment, once model parameters have been optimized, model validation may be performed at block 725 to determine whether model has improved and to determine a current accuracy of pixel-level classifier. At block 730, processing logic determines whether a stopping criterion has been met. A stopping criterion may be a target level of accuracy, a target number of processed images from training dataset, a target amount

of change to parameters over one or more previous data points, a target amount of change of accuracy in a validation set, a combination thereof and/or other criteria. In one embodiment, stopping criteria is met when at least a minimum number of data points have been processed and at least a threshold accuracy is achieved. Threshold accuracy may be, for example, 70%, 80% or 90% accuracy.

[0117] In at least one embodiment, because pixel-level classifier is trained to generate a label for each pixel of input image 740, each pixel represents a training data point for pixel-level classifier, and thus only a small number of labeled images is required for training pixel-level classifier. In at least one embodiment, 10-20 labeled images are usable to train an accurate pixel-level classifier. If stopping criteria is not met, method may return to block 715 to further optimize pixel-level classifier based on another data point from training dataset. If stopping criteria has been met, method continues to block 735 and pixel-level classifier is trained.

[0118] In at least one embodiment, pixel-level classifier that is trained may output, for an input image, a mask that has a same resolution as input image, such as same number of horizontal and vertical pixels. Mask includes a value for each pixel indicating a label for that pixel or a set of label probabilities for that pixel. Accordingly, trained pixel-level classifier makes a pixel level decision for each pixel in an input image as to classification to assign to that pixel. In at least one embodiment, pixel-level classifier is trained to output multiple different masks, where each mask is associated with a different class or label. For example, pixel-level classifier may output a first binary mask having a first value for pixels belonging to a first class and a second value for pixels not belonging to first class, may output a second binary mask having a first value for pixels belonging to a second class and a second value for pixels not belonging to second class, and so on.

TRAINING ONE OR MORE NEURAL NETWORKS USING SYNTHETIC TRAINING DATASETS

[0119] FIG. 8 illustrates a flow diagram for a method 800 of training a neural network using a training dataset that includes synthetic images generated by a trained GAN and corresponding masks generated by a trained pixel-level classifier, in accordance with an embodiment. At block 802 of method 800, an untrained neural network is initialized. In at least one embodiment, neural network that is initialized may be a deep learning model such as a deep neural network. Initialization of artificial neural network may include selecting starting parameters for neural network. In at least one embodiment, parameters are initialized

using Gaussian or uniform distributions with arbitrary set variances. In at least one embodiment, artificial neural network is initialized using a Xavier initialization.

[0120] In at least one embodiment, neural network that is initialized and then trained is able to locate and detect one or more types of classes of objects in images. An output of neural network may be a set of shapes that closely match each of detected objects, as well as a class output for each detected object.

[0121] At block 805, untrained neural network receives a set of synthetic images and a set of corresponding masks from a training dataset. In at least one embodiment, some or all of images in training dataset were generated using techniques discussed above with reference to FIGs. 3-7. In at least one embodiment, a first synthetic image may be, for example, unlabeled image 840 along with a corresponding semantic mask 850. In at least one embodiment, synthetic image 840 may have been generated by a trained generative network or generative model, such as a trained GAN that is same or similar to GAN 562 of FIG. 5B. In at least one embodiment, labeling of labelled image and/or generation of semantic mask may have been generated by a pixel-level classifier that is same or similar to pixel-level classifier 555 of Fig. 5B. In at least one embodiment, training dataset includes any number of synthetic images and corresponding masks. In at least one embodiment, supervised learning is performed to train one or more machine learning models to function as neural network. Semantic mask 850 includes entries corresponding to pixels of unlabeled image 840, such that each entry in semantic mask 850 corresponds to a pixel of unlabeled image 840 and associates pixel with a specific label. For example, for a picture of an automobile, labels may include: car body, front bumper, mirror, head lights, grille, license plate, hood, fender, hubcap, wheel, running board, door, handle, tail bumper, roof, windshield, window, etc. of automobile.

[0122] In at least one embodiment, at block 810, processing logic determines data points for training neural network. In at least one embodiment, processing logic designates each pair of a synthetic image and corresponding mask as a data point, where each data point is usable to train neural network to perform pixel-level segmentation. At block 815, processing logic selects a data point.

[0123] At block 820, processing logic optimizes parameters of neural network from selected data point. In at least one embodiment, neural network applies classifications or labels to pixels of image based on its current parameter values. An artificial neural network includes an input layer that consists of values in a data point, such as channels representing information about pixels in image 840, such as RGB values of pixels. Nodes at a next hidden layer each receive one or more of input values. Each node contains parameters or weights to

apply to input values. Each node therefore essentially inputs input values into a multivariate function such as a non-linear mathematical transformation to produce an output value. A next layer may be another hidden layer or an output layer. In either case, nodes at next layer receive output values from nodes at previous layer, and each node applies weights to those values and then generates its own output value. This may be performed at each layer. A final layer is output layer, where there is one node for each possible classification of pixels, such as one node for each part of object for which pixel can belong). In at least one embodiment, for artificial neural network being trained, a class is determined for each pixel in image. In at least one embodiment, for each pixel in image, final layer applies a probability that pixel of image belongs to one or more specific classes. For example, a particular pixel may be marked as a first class.

[0124] In at least one embodiment, processing logic compares classification or label (or multiple classifications or labels) for pixel of selected data point to provided classification(s) or label(s) for that pixel or point to determine one or more classification error. An error term or delta may be determined for each node in artificial neural network. Based on this error, artificial neural network adjusts one or more of its parameters for one or more of its nodes, which may include adjusting weights for one or more inputs of a node. Parameters may be updated in a back propagation manner, such that nodes at a highest layer are updated first, followed by nodes at a next layer, and so on. An artificial neural network contains multiple layers of “neurons”, where each layer receives as input values from neurons at a previous layer. Parameters for each neuron include weights associated with values that are received from each of neurons at a previous layer. Accordingly, adjusting parameters may include adjusting weights assigned to each of inputs for one or more neurons at one or more layers in artificial neural network.

[0125] In at least one embodiment, once model parameters have been optimized, model validation may be performed at block 825 to determine whether model has improved and to determine a current accuracy of neural network. At block 830, processing logic determines whether a stopping criterion has been met. A stopping criterion may be a target level of accuracy, a target number of processed images from training dataset, a target amount of change to parameters over one or more previous data points, a target amount of change of accuracy in a validation set, a combination thereof and/or other criteria. In one embodiment, stopping criteria is met when at least a minimum number of data points have been processed and at least a threshold accuracy is achieved. Threshold accuracy may be, for example, 70%, 80% or 90% accuracy.

[0126] In at least one embodiment, if stopping criteria is not met, method may return to block 815 to further optimize neural network based on another data point from training dataset. If stopping criteria has been met, method continues to block 835 and neural network is trained.

DATA CENTER

[0127] FIG. 9 illustrates an example data center 900, in which at least one embodiment may be used. In at least one embodiment, data center 900 includes a data center infrastructure layer 910, a framework layer 920, a software layer 930 and an application layer 940.

[0128] In at least one embodiment, as shown in FIG. 9, data center infrastructure layer 910 may include a resource orchestrator 912, grouped computing resources 914, and node computing resources (“node C.R.s”) 916(1)-916(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 916(1)-916(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 918(1)-918(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 916(1)-916(N) may be a server having one or more of above-mentioned computing resources.

[0129] In at least one embodiment, grouped computing resources 914 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 914 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.

[0130] In at least one embodiment, resource orchestrator 912 may configure or otherwise control one or more node C.R.s 916(1)-916(N) and/or grouped computing resources 914. In at least one embodiment, resource orchestrator 912 may include a software design

infrastructure (“SDI”) management entity for data center 900. In at least one embodiment, resource orchestrator 112 may include hardware, software or some combination thereof.

[0131] In at least one embodiment, as shown in FIG. 9, framework layer 920 includes a job scheduler 922, a configuration manager 924, a resource manager 926 and a distributed file system 928. In at least one embodiment, framework layer 920 may include a framework to support software 932 of software layer 930 and/or one or more application(s) 942 of application layer 940. In at least one embodiment, software 932 or application(s) 942 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 embodiment, framework layer 920 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 928 for large-scale data processing (e.g., “big data”). In at least one embodiment, job scheduler 922 may include a Spark driver to facilitate scheduling of workloads supported by various layers of data center 900. In at least one embodiment, configuration manager 924 may be capable of configuring different layers such as software layer 930 and framework layer 920 including Spark and distributed file system 928 for supporting large-scale data processing. In at least one embodiment, resource manager 926 may be capable of managing clustered or grouped computing resources mapped to or allocated for support of distributed file system 928 and job scheduler 922. In at least one embodiment, clustered or grouped computing resources may include grouped computing resources 914 at data center infrastructure layer 910. In at least one embodiment, resource manager 926 may coordinate with resource orchestrator 912 to manage these mapped or allocated computing resources.

[0132] In at least one embodiment, software 932 included in software layer 930 may include software used by at least portions of node C.R.s 916(1)-916(N), grouped computing resources 914, and/or distributed file system 928 of framework layer 920. 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.

[0133] In at least one embodiment, application(s) 942 included in application layer 940 may include one or more types of applications used by at least portions of node C.R.s 916(1)-916(N), grouped computing resources 914, and/or distributed file system 928 of framework layer 920. 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, 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.

[0134] In at least one embodiment, any of configuration manager 924, resource manager 926, and resource orchestrator 912 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 900 from making possibly bad configuration decisions and possibly avoiding underutilized and/or poor performing portions of a data center.

[0135] In at least one embodiment, data center 900 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 900. 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 900 by using weight parameters calculated through one or more training techniques described herein.

[0136] 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.

[0137] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 9 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

[0138] FIG. 10A illustrates an example of an autonomous vehicle 1000, according to at least one embodiment. In at least one embodiment, autonomous vehicle 1000 (alternatively referred to herein as “vehicle 1000”) may be, without limitation, a passenger vehicle, such as a car, a truck, a bus, and/or another type of vehicle that accommodates one or more passengers. In at least one embodiment, vehicle 1000 may be a semi-tractor-trailer truck used for hauling cargo. In at least one embodiment, vehicle 1000 may be an airplane, robotic vehicle, or other kind of vehicle.

[0139] 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 1000 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 1000 may be capable of conditional automation (Level 3), high automation (Level 4), and/or full automation (Level 5), depending on embodiment.

[0140] In at least one embodiment, vehicle 1000 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 1000 may include, without limitation, a propulsion system 1050, 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 1050 may be connected to a drive train of vehicle 1000, which may include, without limitation, a transmission, to enable propulsion of vehicle 1000. In at least one embodiment, propulsion system 1050 may be controlled in response to receiving signals from a throttle/accelerator(s) 1052.

[0141] In at least one embodiment, a steering system 1054, which may include, without limitation, a steering wheel, is used to steer vehicle 1000 (e.g., along a desired path or route) when propulsion system 1050 is operating (e.g., when vehicle 1000 is in motion). In at least one embodiment, steering system 1054 may receive signals from steering actuator(s) 1056. 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 1046 may be used to operate

vehicle brakes in response to receiving signals from brake actuator(s) 1048 and/or brake sensors.

[0142] In at least one embodiment, controller(s) 1036, which may include, without limitation, one or more system on chips (“SoCs”) (not shown in FIG. 10A) 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 1000. For instance, in at least one embodiment, controller(s) 1036 may send signals to operate vehicle brakes via brake actuator(s) 1048, to operate steering system 1054 via steering actuator(s) 1056, to operate propulsion system 1050 via throttle/accelerator(s) 1052. In at least one embodiment, controller(s) 1036 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 1000. In at least one embodiment, controller(s) 1036 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.

[0143] In at least one embodiment, controller(s) 1036 provide signals for controlling one or more components and/or systems of vehicle 1000 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) 1058 (e.g., Global Positioning System sensor(s)), RADAR sensor(s) 1060, ultrasonic sensor(s) 1062, LIDAR sensor(s) 1064, inertial measurement unit (“IMU”) sensor(s) 1066 (e.g., accelerometer(s), gyroscope(s), a magnetic compass or magnetic compasses, magnetometer(s), etc.), microphone(s) 1096, stereo camera(s) 1068, wide-view camera(s) 1070 (e.g., fisheye cameras), infrared camera(s) 1072, surround camera(s) 1074 (e.g., 360 degree cameras), long-range cameras (not shown in FIG. 10A), mid-range camera(s) (not shown in FIG. 10A), speed sensor(s) 1044 (e.g., for measuring speed of vehicle 1000), vibration sensor(s) 1042, steering sensor(s) 1040, brake sensor(s) (e.g., as part of brake sensor system 1046), and/or other sensor types.

[0144] In at least one embodiment, one or more of controller(s) 1036 may receive inputs (e.g., represented by input data) from an instrument cluster 1032 of vehicle 1000 and provide outputs (e.g., represented by output data, display data, etc.) via a human-machine interface

(“HMI”) display 1034, an audible annunciator, a loudspeaker, and/or via other components of vehicle 1000. In at least one embodiment, outputs may include information such as vehicle velocity, speed, time, map data (e.g., a High Definition map (not shown in FIG. 10A), location data (e.g., vehicle’s 1000 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) 1036, etc. For example, in at least one embodiment, HMI display 1034 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.).

[0145] In at least one embodiment, vehicle 1000 further includes a network interface 1024 which may use wireless antenna(s) 1026 and/or modem(s) to communicate over one or more networks. For example, in at least one embodiment, network interface 1024 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) 1026 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.

[0146] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 10A 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.

[0147] FIG. 10B illustrates an example of camera locations and fields of view for autonomous vehicle 1000 of FIG. 10A, 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 alternative cameras may be included and/or cameras may be located at different locations on vehicle 1000.

[0148] 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 1000. 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.

[0149] 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.

[0150] 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 1000 (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.

[0151] In at least one embodiment, cameras with a field of view that include portions of an environment in front of vehicle 1000 (e.g., front-facing cameras) may be used for surround view, to help identify forward facing paths and obstacles, as well as aid in, with help of one or more of controller(s) 1036 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.

[0152] 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 1070 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 1070 is illustrated in FIG. 10B, in other embodiments, there may be any number (including zero) wide-view cameras on vehicle 1000. In at least one embodiment, any number of long-range camera(s) 1098 (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) 1098 may also be used for object detection and classification, as well as basic object tracking.

[0153] In at least one embodiment, any number of stereo camera(s) 1068 may also be included in a front-facing configuration. In at least one embodiment, one or more of stereo camera(s) 1068 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 1000, including a distance estimate for all points in an image. In at least one embodiment, one or more of stereo camera(s) 1068 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 1000 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) 1068 may be used in addition to, or alternatively from, those described herein.

[0154] In at least one embodiment, cameras with a field of view that include portions of environment to sides of vehicle 1000 (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) 1074 (e.g., four surround cameras as illustrated in FIG. 10B) could be positioned

on vehicle 1000. In at least one embodiment, surround camera(s) 1074 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 1000. In at least one embodiment, vehicle 1000 may use three surround camera(s) 1074 (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.

[0155] In at least one embodiment, cameras with a field of view that include portions of an environment behind vehicle 1000 (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 1098 and/or mid-range camera(s) 1076, stereo camera(s) 1068), infrared camera(s) 1072, etc.), as described herein.

[0156] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 10B 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.

[0157] FIG. 10C is a block diagram illustrating an example system architecture for autonomous vehicle 1000 of FIG. 10A, according to at least one embodiment. In at least one embodiment, each of components, features, and systems of vehicle 1000 in FIG. 10C is illustrated as being connected via a bus 1002. In at least one embodiment, bus 1002 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 1000 used to aid in control of various features and functionality of vehicle 1000, such as actuation of brakes, acceleration, braking, steering, windshield wipers, etc. In at least one embodiment, bus 1002 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 1002 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 1002 may be a CAN bus that is ASIL B compliant.

[0158] 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 1002, 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 1002 may communicate with any of components of vehicle 1000, and two or more busses of bus 1002 may communicate with corresponding components. In at least one embodiment, each of any number of system(s) on chip(s) (“SoC(s)”) 1004 (such as SoC 1004(A) and SoC 1004(B), each of controller(s) 1036, and/or each computer within vehicle may have access to same input data (e.g., inputs from sensors of vehicle 1000), and may be connected to a common bus, such as CAN bus.

[0159] In at least one embodiment, vehicle 1000 may include one or more controller(s) 1036, such as those described herein with respect to FIG. 10A. In at least one embodiment, controller(s) 1036 may be used for a variety of functions. In at least one embodiment, controller(s) 1036 may be coupled to any of various other components and systems of vehicle 1000, and may be used for control of vehicle 1000, artificial intelligence of vehicle 1000, infotainment for vehicle 1000, and/or other functions.

[0160] In at least one embodiment, vehicle 1000 may include any number of SoCs 1004. In at least one embodiment, each of SoCs 1004 may include, without limitation, central processing units (“CPU(s)”) 1006, graphics processing units (“GPU(s)”) 1008, processor(s) 1010, cache(s) 1012, accelerator(s) 1014, data store(s) 1016, and/or other components and features not illustrated. In at least one embodiment, SoC(s) 1004 may be used to control vehicle 1000 in a variety of platforms and systems. For example, in at least one embodiment, SoC(s) 1004 may be combined in a system (e.g., system of vehicle 1000) with a High Definition (“HD”) map 1022 which may obtain map refreshes and/or updates via network interface 1024 from one or more servers (not shown in FIG. 10C).

[0161] In at least one embodiment, CPU(s) 1006 may include a CPU cluster or CPU complex (alternatively referred to herein as a “CCPLEX”). In at least one embodiment, CPU(s) 1006 may include multiple cores and/or level two (“L2”) caches. For instance, in at least one embodiment, CPU(s) 1006 may include eight cores in a coherent multi-processor configuration. In at least one embodiment, CPU(s) 1006 may include four dual-core clusters

where each cluster has a dedicated L2 cache (e.g., a 2 megabyte (MB) L2 cache). In at least one embodiment, CPU(s) 1006 (e.g., CCPLEX) may be configured to support simultaneous cluster operations enabling any combination of clusters of CPU(s) 1006 to be active at any given time.

[0162] In at least one embodiment, one or more of CPU(s) 1006 may implement power 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 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) 1006 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 CCPLEX. In at least one embodiment, processing cores may support simplified power state entry sequences in software with work offloaded to microcode.

[0163] In at least one embodiment, GPU(s) 1008 may include an integrated GPU (alternatively referred to herein as an “iGPU”). In at least one embodiment, GPU(s) 1008 may be programmable and may be efficient for parallel workloads. In at least one embodiment, GPU(s) 1008 may use an enhanced tensor instruction set. In at least one embodiment, GPU(s) 1008 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 least 96 KB storage capacity), and two or more streaming microprocessors may share an L2 cache (e.g., an L2 cache with a 512 KB storage capacity). In at least one embodiment, GPU(s) 1008 may include at least eight streaming microprocessors. In at least one embodiment, GPU(s) 1008 may use compute application programming interface(s) (API(s)). In at least one embodiment, GPU(s) 1008 may use one or more parallel computing platforms and/or programming models (e.g., NVIDIA’s CUDA model).

[0164] In at least one embodiment, one or more of GPU(s) 1008 may be power-optimized for best performance in automotive and embedded use cases. For example, in at least one embodiment, GPU(s) 1008 could be fabricated on Fin field-effect transistor (“FinFET”) 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 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 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.

[0165] In at least one embodiment, one or more of GPU(s) 1008 may include a high bandwidth memory (“HBM”) and/or a 16 GB high-bandwidth memory second generation (“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”).

[0166] In at least one embodiment, GPU(s) 1008 may include unified memory technology. In at least one embodiment, address translation services (“ATS”) support may be used to allow GPU(s) 1008 to access CPU(s) 1006 page tables directly. In at least one embodiment, when a GPU of GPU(s) 1008 memory management unit (“MMU”) experiences a miss, an address translation request may be transmitted to CPU(s) 1006. In response, 2 CPU of CPU(s) 1006 may look in its page tables for a virtual-to-physical mapping for an address and transmit translation back to GPU(s) 1008, in at least one embodiment. In at least one embodiment, unified memory technology may allow a single unified virtual address space for memory of both CPU(s) 1006 and GPU(s) 1008, thereby simplifying GPU(s) 1008 programming and porting of applications to GPU(s) 1008.

[0167] In at least one embodiment, GPU(s) 1008 may include any number of access counters that may keep track of frequency of access of GPU(s) 1008 to memory of other 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.

[0168] In at least one embodiment, one or more of SoC(s) 1004 may include any number of cache(s) 1012, including those described herein. For example, in at least one embodiment, cache(s) 1012 could include a level three (“L3”) cache that is available to both CPU(s) 1006 and GPU(s) 1008 (e.g., that is connected to CPU(s) 1006 and GPU(s) 1008). In at least one embodiment, cache(s) 1012 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 embodiment, although smaller cache sizes may be used.

[0169] In at least one embodiment, one or more of SoC(s) 1004 may include one or more accelerator(s) 1014 (e.g., hardware accelerators, software accelerators, or a combination thereof). In at least one embodiment, SoC(s) 1004 may include a hardware acceleration cluster that may include optimized hardware accelerators and/or large on-chip memory. In at 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) 1008 and to off-load some of tasks of GPU(s) 1008 (e.g., to free up more cycles of GPU(s) 1008 for performing other tasks). In at least one embodiment, accelerator(s) 1014 could be used for 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 networks (“RCNNs”) and Fast RCNNs (e.g., as used for object detection) or other type of CNN.

[0170] In at least one embodiment, accelerator(s) 1014 (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.

[0171] In at least one embodiment, DLA(s) may perform any function of GPU(s) 1008, and by using an inference accelerator, for example, a designer may target either DLA(s) or GPU(s) 1008 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) 1008 and/or accelerator(s) 1014.

[0172] In at least one embodiment, accelerator(s) 1014 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”) 1038, autonomous driving, augmented reality (“AR”) applications, and/or virtual reality (“VR”) applications. In at least one embodiment, PVA may provide a balance between performance 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.

[0173] 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”). 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.

[0174] In at least one embodiment, DMA may enable components of PVA to access system memory independently of CPU(s) 1006. 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, vertical block stepping, and/or depth stepping.

[0175] 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 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, 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.

[0176] In at least one embodiment, each of vector processors may include an instruction 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.

[0177] In at least one embodiment, accelerator(s) 1014 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) 1014. 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).

[0178] 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.

[0179] In at least one embodiment, one or more of SoC(s) 1004 may include a real-time 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 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.

[0180] In at least one embodiment, accelerator(s) 1014 can have a wide array of uses for autonomous driving. In at least one embodiment, a PVA may be used for key processing 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 1000, PVAs might be designed to run classic computer vision algorithms, as they can be efficient at object detection and operating on integer math.

[0181] 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 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.

[0182] In at least one embodiment, a PVA may be used to perform dense optical flow. For 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.

[0183] In at least one embodiment, a DLA may be used to run any type of network to 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 relative “weight” of each detection compared to other detections. In at least one embodiment, 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 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 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) 1066 that correlates with vehicle 1000 orientation, distance, 3D location estimates of object obtained from neural network and/or other sensors (e.g., LIDAR sensor(s) 1064 or RADAR sensor(s) 1060), among others.

[0184] In at least one embodiment, one or more of SoC(s) 1004 may include data store(s) 1016 (e.g., memory). In at least one embodiment, data store(s) 1016 may be on-chip memory of SoC(s) 1004, which may store neural networks to be executed on GPU(s) 1008 and/or a DLA. In at least one embodiment, data store(s) 1016 may be large enough in capacity to store multiple instances of neural networks for redundancy and safety. In at least one embodiment, data store(s) 1016 may comprise L2 or L3 cache(s).

[0185] In at least one embodiment, one or more of SoC(s) 1004 may include any number of processor(s) 1010 (e.g., embedded processors). In at least one embodiment, processor(s) 1010

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 enforcement. In at least one embodiment, a boot and power management processor may be a part of a boot sequence of SoC(s) 1004 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) 1004 thermals and temperature sensors, and/or management of SoC(s) 1004 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) 1004 may use ring-oscillators to detect temperatures of CPU(s) 1006, GPU(s) 1008, and/or accelerator(s) 1014. In at least one embodiment, if temperatures are determined to exceed a threshold, then a boot and power management processor may enter a temperature fault routine and put SoC(s) 1004 into a lower power state and/or put vehicle 1000 into a chauffeur to safe stop mode (e.g., bring vehicle 1000 to a safe stop).

[0186] In at least one embodiment, processor(s) 1010 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.

[0187] In at least one embodiment, processor(s) 1010 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.

[0188] In at least one embodiment, processor(s) 1010 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) 1010 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) 1010 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.

[0189] In at least one embodiment, processor(s) 1010 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 compositor may perform lens distortion correction on wide-view camera(s) 1070, surround camera(s) 1074, 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 1004, 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.

[0190] 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.

[0191] 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) 1008 are not required to continuously render new surfaces. In at least one embodiment, when GPU(s) 1008 are powered on and active doing 3D rendering, a video image compositor may be used to offload GPU(s) 1008 to improve performance and responsiveness.

[0192] In at least one embodiment, one or more SoC of SoC(s) 1004 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) 1004 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.

[0193] In at least one embodiment, one or more of SoC(s) 1004 may further include a broad range of peripheral interfaces to enable communication with peripherals, audio encoders/decoders (“codecs”), power management, and/or other devices. In at least one embodiment, SoC(s) 1004 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) 1064, RADAR sensor(s) 1060, etc. that may be connected over Ethernet channels), data from bus 1002 (e.g., speed of vehicle 1000, steering wheel position, etc.), data from GNSS sensor(s) 1058 (e.g., connected over a Ethernet bus or a CAN bus), etc. In at least one embodiment, one or more SoC of SoC(s) 1004 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) 1006 from routine data management tasks.

[0194] In at least one embodiment, SoC(s) 1004 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) 1004 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) 1014, when combined with CPU(s) 1006, GPU(s) 1008, and data store(s) 1016, may provide for a fast, efficient platform for Level 3-5 autonomous vehicles.

[0195] 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.

[0196] 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) 1020) 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 further include a neural network that is able to identify, interpret, and provide semantic understanding of a sign, and to pass that semantic understanding to path planning modules running on a CPU Complex.

[0197] 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) 1008.

[0198] 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 1000. 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) 1004 provide for security against theft and/or carjacking.

[0199] In at least one embodiment, a CNN for emergency vehicle detection and identification may use data from microphones 1096 to detect and identify emergency vehicle sirens. In at least one embodiment, SoC(s) 1004 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) 1058. 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 is detected, a control program may be used to execute an emergency vehicle safety routine, 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) 1062, until emergency vehicles pass.

[0200] In at least one embodiment, vehicle 1000 may include CPU(s) 1018 (e.g., discrete CPU(s), or dCPU(s)), that may be coupled to SoC(s) 1004 via a high-speed interconnect (e.g., PCIe). In at least one embodiment, CPU(s) 1018 may include an X86 processor, for example. CPU(s) 1018 may be used to perform any of a variety of functions, including arbitrating potentially inconsistent results between ADAS sensors and SoC(s) 1004, and/or monitoring status and health of controller(s) 1036 and/or an infotainment system on a chip (“infotainment SoC”) 1030, for example.

[0201] In at least one embodiment, vehicle 1000 may include GPU(s) 1020 (e.g., discrete GPU(s), or dGPU(s)), that may be coupled to SoC(s) 1004 via a high-speed interconnect (e.g., NVIDIA’s NVLINK channel). In at least one embodiment, GPU(s) 1020 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 1000.

[0202] In at least one embodiment, vehicle 1000 may further include network interface 1024 which may include, without limitation, wireless antenna(s) 1026 (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 1024 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 1000 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 1000 information about vehicles in proximity to vehicle 1000 (e.g., vehicles in front of, on a side of, and/or behind vehicle 1000). In at least one embodiment, such aforementioned functionality may be part of a cooperative adaptive cruise control functionality of vehicle 1000.

[0203] In at least one embodiment, network interface 1024 may include an SoC that provides modulation and demodulation functionality and enables controller(s) 1036 to communicate over wireless networks. In at least one embodiment, network interface 1024 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.

[0204] In at least one embodiment, vehicle 1000 may further include data store(s) 1028 which may include, without limitation, off-chip (e.g., off SoC(s) 1004) storage. In at least one embodiment, data store(s) 1028 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.

[0205] In at least one embodiment, vehicle 1000 may further include GNSS sensor(s) 1058 (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) 1058 may be used, including, for example and without limitation, a GPS using a Universal Serial Bus (“USB”) connector with an Ethernet-to-Serial (e.g., RS-232) bridge.

[0206] In at least one embodiment, vehicle 1000 may further include RADAR sensor(s) 1060. In at least one embodiment, RADAR sensor(s) 1060 may be used by vehicle 1000 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) 1060 may use a CAN bus and/or bus 1002 (e.g., to transmit data generated by RADAR sensor(s) 1060) 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) 1060 may be suitable for front, rear, and side RADAR use. In at least one embodiment, one or more sensor of RADAR sensors(s) 1060 is a Pulse Doppler RADAR sensor.

[0207] In at least one embodiment, RADAR sensor(s) 1060 may include different 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 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) 1060 may help in distinguishing between static and moving objects, and may be used by ADAS system 1038 for emergency brake assist and forward collision warning. In at least one embodiment, sensors 1060(s) included in a long-range RADAR system may include, without 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 1000 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, making it possible to quickly detect vehicles entering or leaving a lane of vehicle 1000.

[0208] 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) 1060 designed to be installed at 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 1038 for blind spot detection and/or lane change assist.

[0209] In at least one embodiment, vehicle 1000 may further include ultrasonic sensor(s) 1062. In at least one embodiment, ultrasonic sensor(s) 1062, which may be positioned at a front, a back, and/or side location of vehicle 1000, 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) 1062 may be used, and different ultrasonic sensor(s) 1062 may be used for different ranges of detection (e.g., 2.5 m, 4 m). In at least one embodiment, ultrasonic sensor(s) 1062 may operate at functional safety levels of ASIL B.

[0210] In at least one embodiment, vehicle 1000 may include LIDAR sensor(s) 1064. In at least one embodiment, LIDAR sensor(s) 1064 may be used for object and pedestrian detection, emergency braking, collision avoidance, and/or other functions. In at least one

embodiment, LIDAR sensor(s) 1064 may operate at functional safety level ASIL B. In at least one embodiment, vehicle 1000 may include multiple LIDAR sensors 1064 (e.g., two, four, six, etc.) that may use an Ethernet channel (e.g., to provide data to a Gigabit Ethernet switch).

[0211] In at least one embodiment, LIDAR sensor(s) 1064 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) 1064 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) 1064 may include a small device that may be embedded into a front, a rear, a side, and/or a corner location of vehicle 1000. In at least one embodiment, LIDAR sensor(s) 1064, 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) 1064 may be configured for a horizontal field of view between 45 degrees and 135 degrees.

[0212] 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 1000 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 1000 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 1000. 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.

[0213] In at least one embodiment, vehicle 1000 may further include IMU sensor(s) 1066. In at least one embodiment, IMU sensor(s) 1066 may be located at a center of a rear axle of vehicle 1000. In at least one embodiment, IMU sensor(s) 1066 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) 1066 may include, without limitation, accelerometers and gyroscopes. In at least one embodiment, such as in nine-axis applications, IMU sensor(s) 1066 may include, without limitation, accelerometers, gyroscopes, and magnetometers.

[0214] In at least one embodiment, IMU sensor(s) 1066 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) 1066 may enable vehicle 1000 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) 1066. In at least one embodiment, IMU sensor(s) 1066 and GNSS sensor(s) 1058 may be combined in a single integrated unit.

[0215] In at least one embodiment, vehicle 1000 may include microphone(s) 1096 placed in and/or around vehicle 1000. In at least one embodiment, microphone(s) 1096 may be used for emergency vehicle detection and identification, among other things.

[0216] In at least one embodiment, vehicle 1000 may further include any number of camera types, including stereo camera(s) 1068, wide-view camera(s) 1070, infrared camera(s) 1072, surround camera(s) 1074, long-range camera(s) 1098, mid-range camera(s) 1076, and/or other camera types. In at least one embodiment, cameras may be used to capture image data around an entire periphery of vehicle 1000. In at least one embodiment, which types of cameras used depends on vehicle 1000. In at least one embodiment, any combination of camera types may be used to provide necessary coverage around vehicle 1000. In at least one embodiment, a number of cameras deployed may differ depending on embodiment. For example, in at least one embodiment, vehicle 1000 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 camera might be as described with more detail previously herein with respect to FIG. 10A and FIG. 10B.

[0217] In at least one embodiment, vehicle 1000 may further include vibration sensor(s) 1042. In at least one embodiment, vibration sensor(s) 1042 may measure vibrations of components of vehicle 1000, 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 1042 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).

[0218] In at least one embodiment, vehicle 1000 may include ADAS system 1038. In at least one embodiment, ADAS system 1038 may include, without limitation, an SoC, in some examples. In at least one embodiment, ADAS system 1038 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.

[0219] In at least one embodiment, ACC system may use RADAR sensor(s) 1060, LIDAR sensor(s) 1064, 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 1000 and automatically adjusts speed of vehicle 1000 to maintain a safe distance from vehicles ahead. In at least one embodiment, a lateral ACC system performs distance keeping, and advises vehicle 1000 to change lanes when necessary. In at least one embodiment, a lateral ACC is related to other ADAS applications, such as LC and CW.

[0220] In at least one embodiment, a CACC system uses information from other vehicles that may be received via network interface 1024 and/or wireless antenna(s) 1026 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 (“I2V”) communication link. In general, V2V communication provides information about immediately preceding vehicles (e.g., vehicles immediately ahead of and in same lane as vehicle 1000), 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 information sources. In at least one embodiment, given information of vehicles ahead of vehicle 1000, a CACC system may be more reliable and it has potential to improve traffic flow smoothness and reduce congestion on road.

[0221] 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 uses a front-facing camera and/or RADAR sensor(s) 1060, coupled to a dedicated processor, digital signal 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.

[0222] 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) 1060, coupled to a dedicated processor, DSP, FPGA, and/or ASIC. In at least one embodiment, 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.

[0223] 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 1000 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 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 provides steering input or braking to correct vehicle 1000 if vehicle 1000 starts to exit its lane.

[0224] 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) 1060, 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.

[0225] 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 1000 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) 1060, 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.

[0226] 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 1000 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 1036). For example, in at least one embodiment, ADAS system 1038 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 1038 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.

[0227] In at least one embodiment, a primary computer may be configured to provide a 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 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.

[0228] In at least one embodiment, a supervisory MCU may be configured to run a neural 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 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, 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) 1004.

[0229] In at least one embodiment, ADAS system 1038 may include a secondary computer 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 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 result is correct, and a bug in software or hardware on that primary computer is not causing a material error.

[0230] In at least one embodiment, an output of ADAS system 1038 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 1038 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.

[0231] In at least one embodiment, vehicle 1000 may further include infotainment SoC 1030 (e.g., an in-vehicle infotainment system (IVI)). Although illustrated and described as an SoC, infotainment system SoC 1030, 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 1030 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 1000. For example, infotainment SoC 1030 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 1034, 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 1030 may further be used to provide information (e.g., visual and/or audible) to user(s) of vehicle 1000, such as information from ADAS system 1038, 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.

[0232] In at least one embodiment, infotainment SoC 1030 may include any amount and type of GPU functionality. In at least one embodiment, infotainment SoC 1030 may communicate over bus 1002 with other devices, systems, and/or components of vehicle 1000. In at least one embodiment, infotainment SoC 1030 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) 1036 (e.g., primary and/or backup computers of vehicle 1000) fail. In at least one embodiment, infotainment SoC 1030 may put vehicle 1000 into a chauffeur to safe stop mode, as described herein.

[0233] In at least one embodiment, vehicle 1000 may further include instrument cluster 1032 (e.g., a digital dash, an electronic instrument cluster, a digital instrument panel, etc.). In at least one embodiment, instrument cluster 1032 may include, without limitation, a controller and/or supercomputer (e.g., a discrete controller or supercomputer). In at least one embodiment, instrument cluster 1032 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

(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 1030 and instrument cluster 1032. In at least one embodiment, instrument cluster 1032 may be included as part of infotainment SoC 1030, or vice versa.

[0234] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 10C 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.

[0235] FIG. 10D is a diagram of a system 1078 for communication between cloud-based server(s) and autonomous vehicle 1000 of FIG. 10A, according to at least one embodiment. In at least one embodiment, system 1078 may include, without limitation, server(s) 1078, network(s) 1090, and any number and type of vehicles, including vehicle 1000. In at least one embodiment, server(s) 1078 may include, without limitation, a plurality of GPUs 1084(A)-1084(H) (collectively referred to herein as GPUs 1084), PCIe switches 1082(A)-1082(D) (collectively referred to herein as PCIe switches 1082), and/or CPUs 1080(A)-1080(B) (collectively referred to herein as CPUs 1080). In at least one embodiment, GPUs 1084, CPUs 1080, and PCIe switches 1082 may be interconnected with high-speed interconnects such as, for example and without limitation, NVLink interfaces 1088 developed by NVIDIA and/or PCIe connections 1086. In at least one embodiment, GPUs 1084 are connected via an NVLink and/or NVSwitch SoC and GPUs 1084 and PCIe switches 1082 are connected via PCIe interconnects. Although eight GPUs 1084, two CPUs 1080, and four PCIe switches 1082 are illustrated, this is not intended to be limiting. In at least one embodiment, each of server(s) 1078 may include, without limitation, any number of GPUs 1084, CPUs 1080, and/or PCIe switches 1082, in any combination. For example, in at least one embodiment, server(s) 1078 could each include eight, sixteen, thirty-two, and/or more GPUs 1084.

[0236] In at least one embodiment, server(s) 1078 may receive, over network(s) 1090 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) 1078 may transmit, over network(s) 1090 and to vehicles, neural networks 1092, updated or otherwise, and/or map information 1094, including, without limitation, information regarding traffic and road conditions. In at least one embodiment, updates to map information 1094

may include, without limitation, updates for HD map 1022, such as information regarding construction sites, potholes, detours, flooding, and/or other obstructions. In at least one embodiment, neural networks 1092, and/or map information 1094 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) 1078 and/or other servers).

[0237] In at least one embodiment, server(s) 1078 may be used to train machine learning 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 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 network(s) 1090), and/or machine learning models may be used by server(s) 1078 to remotely monitor vehicles.

[0238] In at least one embodiment, server(s) 1078 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) 1078 may include deep-learning supercomputers and/or dedicated AI computers powered by GPU(s) 1084, such as a DGX and DGX Station machines developed by NVIDIA. However, in at least one embodiment, server(s) 1078 may include deep learning infrastructure that uses CPU-powered data centers.

[0239] In at least one embodiment, deep-learning infrastructure of server(s) 1078 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 1000. For example, in at least one embodiment, deep-learning infrastructure may receive periodic updates from vehicle 1000, such as a sequence of images and/or objects that vehicle 1000 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 1000 and, if results do not match and deep-learning infrastructure concludes that AI in vehicle 1000 is malfunctioning, then server(s) 1078 may transmit a signal to vehicle 1000

instructing a fail-safe computer of vehicle 1000 to assume control, notify passengers, and complete a safe parking maneuver.

[0240] In at least one embodiment, server(s) 1078 may include GPU(s) 1084 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) 115 are used to perform one or more embodiments. Details regarding hardware structure(x) 115 are provided herein in conjunction with FIGS. 1A and/or 1B.

COMPUTER SYSTEMS

[0241] FIG. 11 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 instruction, according to at least one embodiment. In at least one embodiment, a computer system 1100 may include, without limitation, a component, such as a processor 1102 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 1100 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 1100 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.

[0242] 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 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.

[0243] In at least one embodiment, computer system 1100 may include, without limitation, processor 1102 that may include, without limitation, one or more execution units 1108 to perform machine learning model training and/or inferencing according to techniques described herein. In at least one embodiment, computer system 1100 is a single processor desktop or server system, but in another embodiment, computer system 1100 may be a multiprocessor system. In at least one embodiment, processor 1102 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 1102 may be coupled to a processor bus 1110 that may transmit data signals between processor 1102 and other components in computer system 1100.

[0244] In at least one embodiment, processor 1102 may include, without limitation, a Level 1 (“L1”) internal cache memory (“cache”) 1104. In at least one embodiment, processor 1102 may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor 1102. 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 1106 may store different types of data in various registers including, without limitation, integer registers, floating point registers, status registers, and an instruction pointer register.

[0245] In at least one embodiment, execution unit 1108, including, without limitation, logic to perform integer and floating point operations, also resides in processor 1102. In at least one embodiment, processor 1102 may also include a microcode (“ucode”) read only memory (“ROM”) that stores microcode for certain macro instructions. In at least one embodiment, execution unit 1108 may include logic to handle a packed instruction set 1109. In at least one embodiment, by including packed instruction set 1109 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 1102. 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.

[0246] In at least one embodiment, execution unit 1108 may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic

circuits. In at least one embodiment, computer system 1100 may include, without limitation, a memory 1120. In at least one embodiment, memory 1120 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 1120 may store instruction(s) 1119 and/or data 1121 represented by data signals that may be executed by processor 1102.

[0247] In at least one embodiment, a system logic chip may be coupled to processor bus 1110 and memory 1120. In at least one embodiment, a system logic chip may include, without limitation, a memory controller hub (“MCH”) 1116, and processor 1102 may communicate with MCH 1116 via processor bus 1110. In at least one embodiment, MCH 1116 may provide a high bandwidth memory path 1118 to memory 1120 for instruction and data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH 1116 may direct data signals between processor 1102, memory 1120, and other components in computer system 1100 and to bridge data signals between processor bus 1110, memory 1120, and a system I/O interface 1122. 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 1116 may be coupled to memory 1120 through high bandwidth memory path 1118 and a graphics/video card 1112 may be coupled to MCH 1116 through an Accelerated Graphics Port (“AGP”) interconnect 1114.

[0248] In at least one embodiment, computer system 1100 may use system I/O interface 1122 as a proprietary hub interface bus to couple MCH 1116 to an I/O controller hub (“ICH”) 1130. In at least one embodiment, ICH 1130 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 1120, a chipset, and processor 1102. Examples may include, without limitation, an audio controller 1129, a firmware hub (“flash BIOS”) 1128, a wireless transceiver 1126, a data storage 1124, a legacy I/O controller 1123 containing user input and keyboard interfaces 1125, a serial expansion port 1127, such as a USB port, and a network controller 1134. In at least one embodiment, data storage 1124 may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

[0249] In at least one embodiment, FIG. 11 illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. 11 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 11 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 1100 are interconnected using compute express link (CXL) interconnects.

[0250] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 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.

[0251] FIG. 12 is a block diagram illustrating an electronic device 1200 for utilizing a processor 1210, according to at least one embodiment. In at least one embodiment, electronic device 1200 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.

[0252] In at least one embodiment, electronic device 1200 may include, without limitation, processor 1210 communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor 1210 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. 12 illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. 12 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 12 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. 12 are interconnected using compute express link (CXL) interconnects.

[0253] In at least one embodiment, FIG. 12 may include a display 1224, a touch screen 1225, a touch pad 1230, a Near Field Communications unit (“NFC”) 1245, a sensor hub 1240, a thermal sensor 1246, an Express Chipset (“EC”) 1235, a Trusted Platform Module (“TPM”) 1238, BIOS/firmware/flash memory (“BIOS, FW Flash”) 1222, a DSP 1260, a drive 1220 such as a Solid State Disk (“SSD”) or a Hard Disk Drive (“HDD”), a wireless local area network unit (“WLAN”) 1250, a Bluetooth unit 1252, a Wireless Wide Area Network unit (“WWAN”) 1256, a Global Positioning System (GPS) unit 1255, a camera

(“USB 3.0 camera”) 1254 such as a USB 3.0 camera, and/or a Low Power Double Data Rate (“LPDDR”) memory unit (“LPDDR3”) 1215 implemented in, for example, an LPDDR3 standard. These components may each be implemented in any suitable manner.

[0254] In at least one embodiment, other components may be communicatively coupled to processor 1210 through components described herein. In at least one embodiment, an accelerometer 1241, an ambient light sensor (“ALS”) 1242, a compass 1243, and a gyroscope 1244 may be communicatively coupled to sensor hub 1240. In at least one embodiment, a thermal sensor 1239, a fan 1237, a keyboard 1236, and touch pad 1230 may be communicatively coupled to EC 1235. In at least one embodiment, speakers 1263, headphones 1264, and a microphone (“mic”) 1265 may be communicatively coupled to an audio unit (“audio codec and class D amp”) 1262, which may in turn be communicatively coupled to DSP 1260. In at least one embodiment, audio unit 1262 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”) 1257 may be communicatively coupled to WWAN unit 1256. In at least one embodiment, components such as WLAN unit 1250 and Bluetooth unit 1252, as well as WWAN unit 1256 may be implemented in a Next Generation Form Factor (“NGFF”).

[0255] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 12 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.

[0256] FIG. 13 illustrates a computer system 1300, according to at least one embodiment. In at least one embodiment, computer system 1300 is configured to implement various processes and methods described throughout this disclosure.

[0257] In at least one embodiment, computer system 1300 comprises, without limitation, at least one central processing unit (“CPU”) 1302 that is connected to a communication bus 1310 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 1300 includes, without limitation, a main memory 1304 and control logic (e.g., implemented as hardware,

software, or a combination thereof) and data are stored in main memory 1304, which may take form of random access memory (“RAM”). In at least one embodiment, a network interface subsystem (“network interface”) 1322 provides an interface to other computing devices and networks for receiving data from and transmitting data to other systems with computer system 1300.

[0258] In at least one embodiment, computer system 1300, in at least one embodiment, includes, without limitation, input devices 1308, a parallel processing system 1312, and display devices 1306 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 1308 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.

[0259] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in system FIG. 13 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.

[0260] FIG. 14 illustrates a computer system 1400, according to at least one embodiment. In at least one embodiment, computer system 1400 includes, without limitation, a computer 1410 and a USB stick 1420. In at least one embodiment, computer 1410 may include, without limitation, any number and type of processor(s) (not shown) and a memory (not shown). In at least one embodiment, computer 1410 includes, without limitation, a server, a cloud instance, a laptop, and a desktop computer.

[0261] In at least one embodiment, USB stick 1420 includes, without limitation, a processing unit 1430, a USB interface 1440, and USB interface logic 1450. In at least one embodiment, processing unit 1430 may be any instruction execution system, apparatus, or device capable of executing instructions. In at least one embodiment, processing unit 1430 may include, without limitation, any number and type of processing cores (not shown). In at least one embodiment, processing unit 1430 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 1430 is a

tensor processing unit (“TPC”) that is optimized to perform machine learning inference operations. In at least one embodiment, processing unit 1430 is a vision processing unit (“VPU”) that is optimized to perform machine vision and machine learning inference operations.

[0262] In at least one embodiment, USB interface 1440 may be any type of USB connector or USB socket. For instance, in at least one embodiment, USB interface 1440 is a USB 3.0 Type-C socket for data and power. In at least one embodiment, USB interface 1440 is a USB 3.0 Type-A connector. In at least one embodiment, USB interface logic 1450 may include any amount and type of logic that enables processing unit 1430 to interface with devices (e.g., computer 1410) via USB interface 1440.

[0263] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 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.

[0264] FIG. 15A illustrates an exemplary architecture in which a plurality of GPUs 1510(1)-1510(N) is communicatively coupled to a plurality of multi-core processors 1505(1)-1505(M) over high-speed links 1540(1)-1540(N) (e.g., buses, point-to-point interconnects, etc.). In at least one embodiment, high-speed links 1540(1)-1540(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.

[0265] In addition, and in at least one embodiment, two or more of GPUs 1510 are interconnected over high-speed links 1529(1)-1529(2), which may be implemented using similar or different protocols/links than those used for high-speed links 1540(1)-1540(N). Similarly, two or more of multi-core processors 1505 may be connected over a high-speed link 1528 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. 15A may be accomplished using similar protocols/links (e.g., over a common interconnection fabric).

[0266] In at least one embodiment, each multi-core processor 1505 is communicatively coupled to a processor memory 1501(1)-1501(M), via memory interconnects 1526(1)-1526(M), respectively, and each GPU 1510(1)-1510(N) is communicatively coupled to GPU memory 1520(1)-1520(N) over GPU memory interconnects 1550(1)-1550(N), respectively. In at least one embodiment, memory interconnects 1526 and 1550 may utilize similar or different memory access technologies. By way of example, and not limitation, processor memories 1501(1)-1501(M) and GPU memories 1520 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 be non-volatile memories such as 3D XPoint or Nano-Ram. In at least one embodiment, some portion of processor memories 1501 may be volatile memory and another portion may be non-volatile memory (e.g., using a two-level memory (2LM) hierarchy).

[0267] As described herein, although various multi-core processors 1505 and GPUs 1510 may be physically coupled to a particular memory 1501, 1520, respectively, and/or a unified 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 1501(1)-1501(M) may each comprise 64 GB of system memory address space and GPU memories 1520(1)-1520(N) may each comprise 32 GB of system memory address space resulting in a total of 256 GB addressable memory when M=2 and N=4. Other values for N and M are possible.

[0268] FIG. 15B illustrates additional details for an interconnection between a multi-core processor 1507 and a graphics acceleration module 1546 in accordance with one exemplary embodiment. In at least one embodiment, graphics acceleration module 1546 may include one or more GPU chips integrated on a line card which is coupled to processor 1507 via high-speed link 1540 (e.g., a PCIe bus, NVLink, etc.). In at least one embodiment, graphics acceleration module 1546 may alternatively be integrated on a package or chip with processor 1507.

[0269] In at least one embodiment, processor 1507 includes a plurality of cores 1560A-1560D, each with a translation lookaside buffer (“TLB”) 1561A-1561D and one or more caches 1562A-1562D. In at least one embodiment, cores 1560A-1560D may include various other components for executing instructions and processing data that are not illustrated. In at least one embodiment, caches 1562A-1562D may comprise Level 1 (L1) and Level 2 (L2) caches. In addition, one or more shared caches 1556 may be included in caches 1562A-1562D and shared by sets of cores 1560A-1560D. For example, one embodiment of

processor 1507 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 1507 and graphics acceleration module 1546 connect with system memory 1514, which may include processor memories 1501(1)-1501(M) of FIG. 15A.

[0270] In at least one embodiment, coherency is maintained for data and instructions stored in various caches 1562A-1562D, 1556 and system memory 1514 via inter-core communication over a coherence bus 1564. In at least one embodiment, for example, each cache may have cache coherency logic/circuitry associated therewith to communicate to over coherence bus 1564 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 1564 to snoop cache accesses.

[0271] In at least one embodiment, a proxy circuit 1525 communicatively couples graphics acceleration module 1546 to coherence bus 1564, allowing graphics acceleration module 1546 to participate in a cache coherence protocol as a peer of cores 1560A-1560D. In particular, in at least one embodiment, an interface 1535 provides connectivity to proxy circuit 1525 over high-speed link 1540 and an interface 1537 connects graphics acceleration module 1546 to high-speed link 1540.

[0272] In at least one embodiment, an accelerator integration circuit 1536 provides cache management, memory access, context management, and interrupt management services on behalf of a plurality of graphics processing engines 1531(1)-1531(N) of graphics acceleration module 1546. In at least one embodiment, graphics processing engines 1531(1)-1531(N) may each comprise a separate GPU. In at least one embodiment, graphics processing engines 1531(1)-1531(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 1546 may be a GPU with a plurality of graphics processing engines 1531(1)-1531(N) or graphics processing engines 1531(1)-1531(N) may be individual GPUs integrated on a common package, line card, or chip.

[0273] In at least one embodiment, accelerator integration circuit 1536 includes a memory management unit (MMU) 1539 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 1514. In at least one embodiment, MMU 1539 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 1538 can store commands and data for efficient access by graphics processing engines 1531(1)-1531(N). In at least one embodiment, data stored in cache 1538 and graphics memories 1533(1)-1533(M) is kept coherent with core caches 1562A-1562D, 1556 and system memory 1514, possibly using a fetch unit 1544. As mentioned, this may be accomplished via proxy circuit 1525 on behalf of cache 1538 and memories 1533(1)-1533(M) (e.g., sending updates to cache 1538 related to modifications/accesses of cache lines on processor caches 1562A-1562D, 1556 and receiving updates from cache 1538).

[0274] In at least one embodiment, a set of registers 1545 store context data for threads executed by graphics processing engines 1531(1)-1531(N) and a context management circuit 1548 manages thread contexts. For example, context management circuit 1548 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 execute by a graphics processing engine). For example, on a context switch, context management circuit 1548 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 1547 receives and processes interrupts received from system devices.

[0275] In at least one embodiment, virtual/effective addresses from a graphics processing engine 1531 are translated to real/physical addresses in system memory 1514 by MMU 1539. In at least one embodiment, accelerator integration circuit 1536 supports multiple (e.g., 4, 8, 16) graphics accelerator modules 1546 and/or other accelerator devices. In at least one embodiment, graphics accelerator module 1546 may be dedicated to a single application executed on processor 1507 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 1531(1)-1531(N) are shared with multiple applications or virtual machines (VMs). In at least one embodiment, resources may be subdivided into “slices” which are allocated to different VMs and/or applications based on processing requirements and priorities associated with VMs and/or applications.

[0276] In at least one embodiment, accelerator integration circuit 1536 performs as a bridge to a system for graphics acceleration module 1546 and provides address translation and system memory cache services. In addition, in at least one embodiment, accelerator integration circuit 1536 may provide virtualization facilities for a host processor to manage

virtualization of graphics processing engines 1531(1)-1531(N), interrupts, and memory management.

[0277] In at least one embodiment, because hardware resources of graphics processing engines 1531(1)-1531(N) are mapped explicitly to a real address space seen by host processor 1507, any host processor can address these resources directly using an effective address value. In at least one embodiment, one function of accelerator integration circuit 1536 is physical separation of graphics processing engines 1531(1)-1531(N) so that they appear to a system as independent units.

[0278] In at least one embodiment, one or more graphics memories 1533(1)-1533(M) are coupled to each of graphics processing engines 1531(1)-1531(N), respectively and $N=M$. In at least one embodiment, graphics memories 1533(1)-1533(M) store instructions and data being processed by each of graphics processing engines 1531(1)-1531(N). In at least one embodiment, graphics memories 1533(1)-1533(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.

[0279] In at least one embodiment, to reduce data traffic over high-speed link 1540, biasing techniques can be used to ensure that data stored in graphics memories 1533(1)-1533(M) is data that will be used most frequently by graphics processing engines 1531(1)-1531(N) and preferably not used by cores 1560A-1560D (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 1531(1)-1531(N)) within caches 1562A-1562D, 1556 and system memory 1514.

[0280] FIG. 15C illustrates another exemplary embodiment in which accelerator integration circuit 1536 is integrated within processor 1507. In this embodiment, graphics processing engines 1531(1)-1531(N) communicate directly over high-speed link 1540 to accelerator integration circuit 1536 via interface 1537 and interface 1535 (which, again, may be any form of bus or interface protocol). In at least one embodiment, accelerator integration circuit 1536 may perform similar operations as those described with respect to FIG. 15B, but potentially at a higher throughput given its close proximity to coherence bus 1564 and caches 1562A-1562D, 1556. 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 1536 and programming models which are controlled by graphics acceleration module 1546.

[0281] In at least one embodiment, graphics processing engines 1531(1)-1531(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 1531(1)-1531(N), providing virtualization within a VM/partition.

[0282] In at least one embodiment, graphics processing engines 1531(1)-1531(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 1531(1)-1531(N) to allow access by each operating system. In at least one embodiment, for single-partition systems without a hypervisor, graphics processing engines 1531(1)-1531(N) are owned by an operating system. In at least one embodiment, an operating system can virtualize graphics processing engines 1531(1)-1531(N) to provide access to each process or application.

[0283] In at least one embodiment, graphics acceleration module 1546 or an individual graphics processing engine 1531(1)-1531(N) selects a process element using a process handle. In at least one embodiment, process elements are stored in system memory 1514 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 1531(1)-1531(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.

[0284] FIG. 15D illustrates an exemplary accelerator integration slice 1590. In at least one embodiment, a “slice” comprises a specified portion of processing resources of accelerator integration circuit 1536. In at least one embodiment, an application is effective address space 1582 within system memory 1514 stores process elements 1583. In at least one embodiment, process elements 1583 are stored in response to GPU invocations 1581 from applications 1580 executed on processor 1507. In at least one embodiment, a process element 1583 contains process state for corresponding application 1580. In at least one embodiment, a work descriptor (WD) 1584 contained in process element 1583 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 1584 is a pointer to a job request queue in an application’s effective address space 1582.

[0285] In at least one embodiment, graphics acceleration module 1546 and/or individual graphics processing engines 1531(1)-1531(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 1584 to a graphics acceleration module 1546 to start a job in a virtualized environment may be included.

[0286] 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 1546 or an individual graphics processing engine 1531. In at least one embodiment, when graphics acceleration module 1546 is owned by a single process, a hypervisor initializes accelerator integration circuit 1536 for an owning partition and an operating system initializes accelerator integration circuit 1536 for an owning process when graphics acceleration module 1546 is assigned.

[0287] In at least one embodiment, in operation, a WD fetch unit 1591 in accelerator integration slice 1590 fetches next WD 1584, which includes an indication of work to be done by one or more graphics processing engines of graphics acceleration module 1546. In at least one embodiment, data from WD 1584 may be stored in registers 1545 and used by MMU 1539, interrupt management circuit 1547 and/or context management circuit 1548 as illustrated. For example, one embodiment of MMU 1539 includes segment/page walk circuitry for accessing segment/page tables 1586 within an OS virtual address space 1585. In at least one embodiment, interrupt management circuit 1547 may process interrupt events 1592 received from graphics acceleration module 1546. In at least one embodiment, when performing graphics operations, an effective address 1593 generated by a graphics processing engine 1531(1)-1531(N) is translated to a real address by MMU 1539.

[0288] In at least one embodiment, registers 1545 are duplicated for each graphics processing engine 1531(1)-1531(N) and/or graphics acceleration module 1546 and may be initialized by a hypervisor or an operating system. In at least one embodiment, each of these duplicated registers may be included in an accelerator integration slice 1590. 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

[0289] Exemplary registers that may be initialized by an operating system are shown in 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

[0290] In at least one embodiment, each WD 1584 is specific to a particular graphics acceleration module 1546 and/or graphics processing engines 1531(1)-1531(N). In at least one embodiment, it contains all information required by a graphics processing engine 1531(1)-1531(N) to do work, or it can be a pointer to a memory location where an application has set up a command queue of work to be completed.

[0291] FIG. 15E illustrates additional details for one exemplary embodiment of a shared model. This embodiment includes a hypervisor real address space 1598 in which a process element list 1599 is stored. In at least one embodiment, hypervisor real address space 1598 is accessible via a hypervisor 1596 which virtualizes graphics acceleration module engines for operating system 1595.

[0292] 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 1546. In at least one embodiment, there are two programming models where graphics acceleration module 1546 is shared by multiple processes and partitions, namely time-sliced shared and graphics directed shared.

[0293] In at least one embodiment, in this model, system hypervisor 1596 owns graphics acceleration module 1546 and makes its function available to all operating systems 1595. In

at least one embodiment, for a graphics acceleration module 1546 to support virtualization by system hypervisor 1596, graphics acceleration module 1546 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 1546 must provide a context save and restore mechanism, (2) an application's job request is guaranteed by graphics acceleration module 1546 to complete in a specified amount of time, including any translation faults, or graphics acceleration module 1546 provides an ability to preempt processing of a job, and (3) graphics acceleration module 1546 must be guaranteed fairness between processes when operating in a directed shared programming model.

[0294] In at least one embodiment, application 1580 is required to make an operating system 1595 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 (CSRP). 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 1546 and can be in a form of a graphics acceleration module 1546 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 1546.

[0295] In at least one embodiment, an AMR value is an AMR state to use for a current 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 1536 (not shown) and graphics acceleration module 1546 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 1596 may optionally apply a current Authority Mask Override Register (AMOR) value before placing an AMR into process element 1583. In at least one embodiment, CSRP is one of registers 1545 containing an effective address of an area in an application's effective address space 1582 for graphics acceleration module 1546 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.

[0296] Upon receiving a system call, operating system 1595 may verify that application 1580 has registered and been given authority to use graphics acceleration module 1546. In at

least one embodiment, operating system 1595 then calls hypervisor 1596 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)

[0297] In at least one embodiment, upon receiving a hypervisor call, hypervisor 1596 verifies that operating system 1595 has registered and been given authority to use graphics acceleration module 1546. In at least one embodiment, hypervisor 1596 then puts process element 1583 into a process element linked list for a corresponding graphics acceleration module 1546 type. In at least one embodiment, a process element may include information shown in Table 4.

Table 4 –Process Element Information

<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)

[0298] In at least one embodiment, hypervisor initializes a plurality of accelerator integration slice 1590 registers 1545.

[0299] As illustrated in FIG. 15F, in at least one embodiment, a unified memory is used, addressable via a common virtual memory address space used to access physical processor memories 1501(1)-1501(N) and GPU memories 1520(1)-1520(N). In this implementation, operations executed on GPUs 1510(1)-1510(N) utilize a same virtual/effective memory address space to access processor memories 1501(1)-1501(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 1501(1), a second portion to second processor memory 1501(N), a third portion to GPU memory 1520(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 1501 and GPU memories 1520, allowing any processor or GPU to access any physical memory with a virtual address mapped to that memory.

[0300] In at least one embodiment, bias/coherence management circuitry 1594A-1594E within one or more of MMUs 1539A-1539E ensures cache coherence between caches of one or more host processors (e.g., 1505) and GPUs 1510 and implements biasing techniques indicating physical memories in which certain types of data should be stored. In at least one embodiment, while multiple instances of bias/coherence management circuitry 1594A-1594E are illustrated in FIG. 15F, bias/coherence circuitry may be implemented within an MMU of one or more host processors 1505 and/or within accelerator integration circuit 1536.

[0301] One embodiment allows GPU memories 1520 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 1520 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 1505 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 1520 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 1510. 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.

[0302] 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 1520, with or without a bias cache in a GPU 1510 (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.

[0303] In at least one embodiment, a bias table entry associated with each access to a GPU attached memory 1520 is accessed prior to actual access to a GPU memory, causing following operations. In at least one embodiment, local requests from a GPU 1510 that find their page in GPU bias are forwarded directly to a corresponding GPU memory 1520. In at least one embodiment, local requests from a GPU that find their page in host bias are forwarded to processor 1505 (e.g., over a high-speed link as described herein). In at least one embodiment, requests from processor 1505 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 1510. In at least one embodiment, a GPU may then 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.

[0304] In at least one embodiment, one mechanism for changing bias state employs an API 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 1505 bias to GPU bias, but is not for an opposite transition.

[0305] In at least one embodiment, cache coherency is maintained by temporarily rendering GPU-biased pages uncacheable by host processor 1505. In at least one embodiment, to access these pages, processor 1505 may request access from GPU 1510, which may or may not grant access right away. In at least one embodiment, thus, to reduce communication between

processor 1505 and GPU 1510 it is beneficial to ensure that GPU-biased pages are those which are required by a GPU but not host processor 1505 and vice versa.

[0306] Hardware structure(s) 115 are used to perform one or more embodiments. Details regarding a hardware structure(s) 115 may be provided herein in conjunction with FIGS. 1A and/or 1B.

[0307] FIG. 16 illustrates 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.

[0308] FIG. 16 is a block diagram illustrating an exemplary system on a chip integrated circuit 1600 that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, integrated circuit 1600 includes one or more application processor(s) 1605 (e.g., CPUs), at least one graphics processor 1610, and may additionally include an image processor 1615 and/or a video processor 1620, any of which may be a modular IP core. In at least one embodiment, integrated circuit 1600 includes peripheral or bus logic including a USB controller 1625, a UART controller 1630, an SPI/SDIO controller 1635, and an I²S/I²C controller 1640. In at least one embodiment, integrated circuit 1600 can include a display device 1645 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1650 and a mobile industry processor interface (MIPI) display interface 1655. In at least one embodiment, storage may be provided by a flash memory subsystem 1660 including flash memory and a flash memory controller. In at least one embodiment, a memory interface may be provided via a memory controller 1665 for access to SDRAM or SRAM memory devices. In at least one embodiment, some integrated circuits additionally include an embedded security engine 1670.

[0309] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in integrated circuit 1600 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.

[0310] FIGS. 17A-17B 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.

[0311] FIGS. 17A-17B are block diagrams illustrating exemplary graphics processors for use within an SoC, according to embodiments described herein. FIG. 17A illustrates an exemplary graphics processor 1710 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. 17B illustrates an additional exemplary graphics processor 1740 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 1710 of FIG. 17A is a low power graphics processor core. In at least one embodiment, graphics processor 1740 of FIG. 17B is a higher performance graphics processor core. In at least one embodiment, each of graphics processors 1710, 1740 can be variants of graphics processor 1610 of FIG. 16.

[0312] In at least one embodiment, graphics processor 1710 includes a vertex processor 1705 and one or more fragment processor(s) 1715A-1715N (e.g., 1715A, 1715B, 1715C, 1715D, through 1715N-1, and 1715N). In at least one embodiment, graphics processor 1710 can execute different shader programs via separate logic, such that vertex processor 1705 is optimized to execute operations for vertex shader programs, while one or more fragment processor(s) 1715A-1715N execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. In at least one embodiment, vertex processor 1705 performs a vertex processing stage of a 3D graphics pipeline and generates primitives and vertex data. In at least one embodiment, fragment processor(s) 1715A-1715N use primitive and vertex data generated by vertex processor 1705 to produce a framebuffer that is displayed on a display device. In at least one embodiment, fragment processor(s) 1715A-1715N 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.

[0313] In at least one embodiment, graphics processor 1710 additionally includes one or more memory management units (MMUs) 1720A-1720B, cache(s) 1725A-1725B, and circuit interconnect(s) 1730A-1730B. In at least one embodiment, one or more MMU(s) 1720A-1720B provide for virtual to physical address mapping for graphics processor 1710, including for vertex processor 1705 and/or fragment processor(s) 1715A-1715N, 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) 1725A-1725B. In at least one embodiment, one or more MMU(s) 1720A-1720B may be synchronized with other MMUs within a system, including

one or more MMUs associated with one or more application processor(s) 1605, image processors 1015, and/or video processors 1620 of FIG. 16, such that each processor 1605-1620 can participate in a shared or unified virtual memory system. In at least one embodiment, one or more circuit interconnect(s) 1730A-1730B enable graphics processor 1710 to interface with other IP cores within SoC, either via an internal bus of SoC or via a direct connection.

[0314] In at least one embodiment, graphics processor 1740 includes one or more shader core(s) 1755A-1755N (e.g., 1755A, 1755B, 1755C, 1755D, 1755E, 1755F, through 1755N-1, and 1755N) as shown in FIG. 17B, which provides for a unified shader core architecture in which a single core or type or core can execute all types of programmable shader code, 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 1740 includes an inter-core task manager 1745, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores 1755A-1755N and a tiling unit 1758 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.

[0315] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in integrated circuit 11A and/or 11B 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.

[0316] FIGS. 18A-18B illustrate additional exemplary graphics processor logic according to embodiments described herein. FIG. 18A illustrates a graphics core 1800 that may be included within graphics processor 1610 of FIG. 16, in at least one embodiment, and may be a unified shader core 1755A-1755N as in FIG. 17B in at least one embodiment. FIG. 18B illustrates a highly-parallel general-purpose graphics processing unit (“GPGPU”) 1830 suitable for deployment on a multi-chip module in at least one embodiment.

[0317] In at least one embodiment, graphics core 1800 includes a shared instruction cache 1802, a texture unit 1818, and a cache/shared memory 1820 that are common to execution resources within graphics core 1800. In at least one embodiment, graphics core 1800 can include multiple slices 1801A-1801N or a partition for each core, and a graphics processor

can include multiple instances of graphics core 1800. In at least one embodiment, slices 1801A-1801N can include support logic including a local instruction cache 1804A-1804N, a thread scheduler 1806A-1806N, a thread dispatcher 1808A-1808N, and a set of registers 1810A-1810N. In at least one embodiment, slices 1801A-1801N can include a set of additional function units (AFUs 1812A-1812N), floating-point units (FPUs 1814A-1814N), integer arithmetic logic units (ALUs 1816A-1816N), address computational units (ACUs 1813A-1813N), double-precision floating-point units (DPFPUs 1815A-1815N), and matrix processing units (MPUs 1817A-1817N).

[0318] In at least one embodiment, FPUs 1814A-1814N can perform single-precision (32-bit) and half-precision (16-bit) floating point operations, while DPFPUs 1815A-1815N perform double precision (64-bit) floating point operations. In at least one embodiment, ALUs 1816A-1816N 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 1817A-1817N can also be configured for mixed precision matrix operations, including half-precision floating point and 8-bit integer operations. In at least one embodiment, MPUs 1817-1817N 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 1812A-1812N can perform additional logic operations not supported by floating-point or integer units, including trigonometric operations (e.g., sine, cosine, etc.).

[0319] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in graphics core 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.

[0320] FIG. 18B illustrates a general-purpose processing unit (GPGPU) 1830 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 1830 can be linked directly to other instances of GPGPU 1830 to create a multi-GPU cluster to improve training speed for deep neural networks. In at least one embodiment, GPGPU 1830 includes a host interface 1832 to enable a connection with a host processor. In at least one embodiment, host interface 1832 is a PCI Express interface. In at least one embodiment,

host interface 1832 can be a vendor-specific communications interface or communications fabric. In at least one embodiment, GPGPU 1830 receives commands from a host processor and uses a global scheduler 1834 to distribute execution threads associated with those commands to a set of compute clusters 1836A-1836H. In at least one embodiment, compute clusters 1836A-1836H share a cache memory 1838. In at least one embodiment, cache memory 1838 can serve as a higher-level cache for cache memories within compute clusters 1836A-1836H.

[0321] In at least one embodiment, GPGPU 1830 includes memory 1844A-1844B coupled with compute clusters 1836A-1836H via a set of memory controllers 1842A-1842B. In at least one embodiment, memory 1844A-1844B 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.

[0322] In at least one embodiment, compute clusters 1836A-1836H each include a set of graphics cores, such as graphics core 1800 of FIG. 18A, 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 1836A-1836H 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.

[0323] In at least one embodiment, multiple instances of GPGPU 1830 can be configured to operate as a compute cluster. In at least one embodiment, communication used by compute clusters 1836A-1836H for synchronization and data exchange varies across embodiments. In at least one embodiment, multiple instances of GPGPU 1830 communicate over host interface 1832. In at least one embodiment, GPGPU 1830 includes an I/O hub 1839 that couples GPGPU 1830 with a GPU link 1840 that enables a direct connection to other instances of GPGPU 1830. In at least one embodiment, GPU link 1840 is coupled to a dedicated GPU-to-GPU bridge that enables communication and synchronization between multiple instances of GPGPU 1830. In at least one embodiment, GPU link 1840 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 1830 are located in separate data processing systems and communicate via a network device that is accessible via

host interface 1832. In at least one embodiment GPU link 1840 can be configured to enable a connection to a host processor in addition to or as an alternative to host interface 1832.

[0324] In at least one embodiment, GPGPU 1830 can be configured to train neural networks. In at least one embodiment, GPGPU 1830 can be used within an inferencing platform. In at least one embodiment, in which GPGPU 1830 is used for inferencing, GPGPU 1830 may include fewer compute clusters 1836A-1836H relative to when GPGPU 1830 is used for training a neural network. In at least one embodiment, memory technology associated with memory 1844A-1844B 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 1830 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.

[0325] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in GPGPU 1830 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.

[0326] FIG. 19 is a block diagram illustrating a computing system 1900 according to at least one embodiment. In at least one embodiment, computing system 1900 includes a processing subsystem 1901 having one or more processor(s) 1902 and a system memory 1904 communicating via an interconnection path that may include a memory hub 1905. In at least one embodiment, memory hub 1905 may be a separate component within a chipset component or may be integrated within one or more processor(s) 1902. In at least one embodiment, memory hub 1905 couples with an I/O subsystem 1911 via a communication link 1906. In at least one embodiment, I/O subsystem 1911 includes an I/O hub 1907 that can enable computing system 1900 to receive input from one or more input device(s) 1908. In at least one embodiment, I/O hub 1907 can enable a display controller, which may be included in one or more processor(s) 1902, to provide outputs to one or more display device(s) 1910A. In at least one embodiment, one or more display device(s) 1910A coupled with I/O hub 1907 can include a local, internal, or embedded display device.

[0327] In at least one embodiment, processing subsystem 1901 includes one or more parallel processor(s) 1912 coupled to memory hub 1905 via a bus or other communication link 1913. In at least one embodiment, communication link 1913 may use one of any number of standards based communication link technologies or protocols, such as, but not limited to PCI Express, or may be a vendor-specific communications interface or communications fabric. In at least one embodiment, one or more parallel processor(s) 1912 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) 1912 form a graphics processing subsystem that can output pixels to one of one or more display device(s) 1910A coupled via I/O Hub 1907. In at least one embodiment, parallel processor(s) 1912 can also include a display controller and display interface (not shown) to enable a direct connection to one or more display device(s) 1910B.

[0328] In at least one embodiment, a system storage unit 1914 can connect to I/O hub 1907 to provide a storage mechanism for computing system 1900. In at least one embodiment, an I/O switch 1916 can be used to provide an interface mechanism to enable connections between I/O hub 1907 and other components, such as a network adapter 1918 and/or a wireless network adapter 1919 that may be integrated into platform, and various other devices that can be added via one or more add-in device(s) 1920. In at least one embodiment, network adapter 1918 can be an Ethernet adapter or another wired network adapter. In at least one embodiment, wireless network adapter 1919 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.

[0329] In at least one embodiment, computing system 1900 can include other components 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 1907. In at least one embodiment, communication paths interconnecting various components in FIG. 19 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 and/or protocol(s), such as NV-Link high-speed interconnect, or interconnect protocols.

[0330] In at least one embodiment, parallel processor(s) 1912 incorporate circuitry optimized for graphics and video processing, including, for example, video output circuitry, and constitutes a graphics processing unit (GPU). In at least one embodiment, parallel processor(s) 1912 incorporate circuitry optimized for general purpose processing. In at least

embodiment, components of computing system 1900 may be integrated with one or more other system elements on a single integrated circuit. For example, in at least one embodiment, parallel processor(s) 1912, memory hub 1905, processor(s) 1902, and I/O hub 1907 can be integrated into a system on chip (SoC) integrated circuit. In at least one embodiment, components of computing system 1900 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 1900 can be integrated into a multi-chip module (MCM), which can be interconnected with other multi-chip modules into a modular computing system.

[0331] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used computing system 1900 of FIG. 19 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

[0332] FIG. 20A illustrates a parallel processor 2000 according to at least one embodiment. In at least one embodiment, various components of parallel processor 2000 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 2000 is a variant of one or more parallel processor(s) 1912 shown in FIG. 19 according to an exemplary embodiment.

[0333] In at least one embodiment, parallel processor 2000 includes a parallel processing unit 2002. In at least one embodiment, parallel processing unit 2002 includes an I/O unit 2004 that enables communication with other devices, including other instances of parallel processing unit 2002. In at least one embodiment, I/O unit 2004 may be directly connected to other devices. In at least one embodiment, I/O unit 2004 connects with other devices via use of a hub or switch interface, such as a memory hub 2005. In at least one embodiment, connections between memory hub 2005 and I/O unit 2004 form a communication link 2013. In at least one embodiment, I/O unit 2004 connects with a host interface 2006 and a memory crossbar 2016, where host interface 2006 receives commands directed to performing

processing operations and memory crossbar 2016 receives commands directed to performing memory operations.

[0334] In at least one embodiment, when host interface 2006 receives a command buffer via I/O unit 2004, host interface 2006 can direct work operations to perform those commands to a front end 2008. In at least one embodiment, front end 2008 couples with a scheduler 2010, which is configured to distribute commands or other work items to a processing cluster array 2012. In at least one embodiment, scheduler 2010 ensures that processing cluster array 2012 is properly configured and in a valid state before tasks are distributed to a cluster of processing cluster array 2012. In at least one embodiment, scheduler 2010 is implemented via firmware logic executing on a microcontroller. In at least one embodiment, microcontroller implemented scheduler 2010 is configurable to perform complex scheduling and work distribution operations at coarse and fine granularity, enabling rapid preemption and context switching of threads executing on processing array 2012. In at least one embodiment, host software can provide workloads for scheduling on processing cluster array 2012 via one of multiple graphics processing paths. In at least one embodiment, workloads can then be automatically distributed across processing array cluster 2012 by scheduler 2010 logic within a microcontroller including scheduler 2010.

[0335] In at least one embodiment, processing cluster array 2012 can include up to “N” processing clusters (e.g., cluster 2014A, cluster 2014B, through cluster 2014N), 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 2014A-2014N of processing cluster array 2012 can execute a large number of concurrent threads. In at least one embodiment, scheduler 2010 can allocate work to clusters 2014A-2014N of processing cluster array 2012 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 2010, or can be assisted in part by compiler logic during compilation of program logic configured for execution by processing cluster array 2012. In at least one embodiment, different clusters 2014A-2014N of processing cluster array 2012 can be allocated for processing different types of programs or for performing different types of computations.

[0336] In at least one embodiment, processing cluster array 2012 can be configured to perform various types of parallel processing operations. In at least one embodiment, processing cluster array 2012 is configured to perform general-purpose parallel compute operations. For example, in at least one embodiment, processing cluster array 2012 can

include logic to execute processing tasks including filtering of video and/or audio data, performing modeling operations, including physics operations, and performing data transformations.

[0337] In at least one embodiment, processing cluster array 2012 is configured to perform parallel graphics processing operations. In at least one embodiment, processing cluster array 2012 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 2012 can be configured to execute graphics processing related shader programs such as, but not limited to, vertex shaders, tessellation shaders, geometry shaders, and pixel shaders. In at least one embodiment, parallel processing unit 2002 can transfer data from system memory via I/O unit 2004 for processing. In at least one embodiment, during processing, transferred data can be stored to on-chip memory (e.g., parallel processor memory 2022) during processing, then written back to system memory.

[0338] In at least one embodiment, when parallel processing unit 2002 is used to perform graphics processing, scheduler 2010 can be configured to divide a processing workload into approximately equal sized tasks, to better enable distribution of graphics processing operations to multiple clusters 2014A-2014N of processing cluster array 2012. In at least one embodiment, portions of processing cluster array 2012 can be configured to perform different types of processing. For example, in at least one embodiment, a first portion may be 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 2014A-2014N may be stored in buffers to allow intermediate data to be transmitted between clusters 2014A-2014N for further processing.

[0339] In at least one embodiment, processing cluster array 2012 can receive processing tasks to be executed via scheduler 2010, which receives commands defining processing tasks from front end 2008. In at least one embodiment, processing tasks can include indices of data 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 2010 may be configured to fetch indices corresponding to tasks or may receive indices from front end 2008. In at least one embodiment, front end 2008 can be configured to ensure processing cluster array 2012 is

configured to a valid state before a workload specified by incoming command buffers (e.g., batch-buffers, push buffers, etc.) is initiated.

[0340] In at least one embodiment, each of one or more instances of parallel processing unit 2002 can couple with a parallel processor memory 2022. In at least one embodiment, parallel processor memory 2022 can be accessed via memory crossbar 2016, which can receive memory requests from processing cluster array 2012 as well as I/O unit 2004. In at least one embodiment, memory crossbar 2016 can access parallel processor memory 2022 via a memory interface 2018. In at least one embodiment, memory interface 2018 can include multiple partition units (e.g., partition unit 2020A, partition unit 2020B, through partition unit 2020N) that can each couple to a portion (e.g., memory unit) of parallel processor memory 2022. In at least one embodiment, a number of partition units 2020A-2020N is configured to be equal to a number of memory units, such that a first partition unit 2020A has a corresponding first memory unit 2024A, a second partition unit 2020B has a corresponding memory unit 2024B, and an N-th partition unit 2020N has a corresponding N-th memory unit 2024N. In at least one embodiment, a number of partition units 2020A-2020N may not be equal to a number of memory units.

[0341] In at least one embodiment, memory units 2024A-2024N 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. In at least one embodiment, memory units 2024A-2024N 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 2024A-2024N, allowing partition units 2020A-2020N to write portions of each render target in parallel to efficiently use available bandwidth of parallel processor memory 2022. In at least one embodiment, a local instance of parallel processor memory 2022 may be excluded in favor of a unified memory design that utilizes system memory in conjunction with local cache memory.

[0342] In at least one embodiment, any one of clusters 2014A-2014N of processing cluster array 2012 can process data that will be written to any of memory units 2024A-2024N within parallel processor memory 2022. In at least one embodiment, memory crossbar 2016 can be configured to transfer an output of each cluster 2014A-2014N to any partition unit 2020A-2020N or to another cluster 2014A-2014N, which can perform additional processing operations on an output. In at least one embodiment, each cluster 2014A-2014N can communicate with memory interface 2018 through memory crossbar 2016 to read from or

write to various external memory devices. In at least one embodiment, memory crossbar 2016 has a connection to memory interface 2018 to communicate with I/O unit 2004, as well as a connection to a local instance of parallel processor memory 2022, enabling processing units within different processing clusters 2014A-2014N to communicate with system memory or other memory that is not local to parallel processing unit 2002. In at least one embodiment, memory crossbar 2016 can use virtual channels to separate traffic streams between clusters 2014A-2014N and partition units 2020A-2020N.

[0343] In at least one embodiment, multiple instances of parallel processing unit 2002 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 2002 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 2002 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 2002 or parallel processor 2000 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.

[0344] FIG. 20B is a block diagram of a partition unit 2020 according to at least one embodiment. In at least one embodiment, partition unit 2020 is an instance of one of partition units 2020A-2020N of FIG. 20A. In at least one embodiment, partition unit 2020 includes an L2 cache 2021, a frame buffer interface 2025, and a ROP 2026 (raster operations unit). In at least one embodiment, L2 cache 2021 is a read/write cache that is configured to perform load and store operations received from memory crossbar 2016 and ROP 2026. In at least one embodiment, read misses and urgent write-back requests are output by L2 cache 2021 to frame buffer interface 2025 for processing. In at least one embodiment, updates can also be sent to a frame buffer via frame buffer interface 2025 for processing. In at least one embodiment, frame buffer interface 2025 interfaces with one of memory units in parallel processor memory, such as memory units 2024A-2024N of FIG. 20 (e.g., within parallel processor memory 2022).

[0345] In at least one embodiment, ROP 2026 is a processing unit that performs raster operations such as stencil, z test, blending, etc. In at least one embodiment, ROP 2026 then outputs processed graphics data that is stored in graphics memory. In at least one embodiment, ROP 2026 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 2026 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.

[0346] In at least one embodiment, ROP 2026 is included within each processing cluster (e.g., cluster 2014A-2014N of FIG. 20A) instead of within partition unit 2020. In at least one embodiment, read and write requests for pixel data are transmitted over memory crossbar 2016 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) 1910 of FIG. 19, routed for further processing by processor(s) 1302, or routed for further processing by one of processing entities within parallel processor 2000 of FIG. 20A.

[0347] FIG. 20C is a block diagram of a processing cluster 2014 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 2014A-2014N of FIG. 20A. In at least one embodiment, processing cluster 2014 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 instruction unit configured to issue instructions to a set of processing engines within each one of processing clusters.

[0348] In at least one embodiment, operation of processing cluster 2014 can be controlled via a pipeline manager 2032 that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager 2032 receives instructions from scheduler 2010 of FIG. 20A and manages execution of those instructions via a graphics multiprocessor 2034 and/or a texture unit 2036. In at least one embodiment, graphics multiprocessor 2034 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 2014. In at least one embodiment, one or more instances of graphics multiprocessor 2034 can be included within a processing cluster 2014. In at least one

embodiment, graphics multiprocessor 2034 can process data and a data crossbar 2040 can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager 2032 can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data crossbar 2040.

[0349] In at least one embodiment, each graphics multiprocessor 2034 within processing cluster 2014 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.

[0350] In at least one embodiment, instructions transmitted to processing cluster 2014 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 2034. In at least one embodiment, a thread group may include fewer threads than a number of processing engines within graphics multiprocessor 2034. In at least one embodiment, when a thread group includes fewer threads than a number of processing 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 2034. In at least one embodiment, when a thread group includes more threads than number of processing engines within graphics multiprocessor 2034, processing can be performed over consecutive clock cycles. In at least one embodiment, multiple thread groups can be executed concurrently on a graphics multiprocessor 2034.

[0351] In at least one embodiment, graphics multiprocessor 2034 includes an internal cache memory to perform load and store operations. In at least one embodiment, graphics multiprocessor 2034 can forego an internal cache and use a cache memory (e.g., L1 cache 2048) within processing cluster 2014. In at least one embodiment, each graphics multiprocessor 2034 also has access to L2 caches within partition units (e.g., partition units

2020A-2020N of FIG. 20A) that are shared among all processing clusters 2014 and may be used to transfer data between threads. In at least one embodiment, graphics multiprocessor 2034 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 2002 may be used as global memory. In at least one embodiment, processing cluster 2014 includes multiple instances of graphics multiprocessor 2034 and can share common instructions and data, which may be stored in L1 cache 2048.

[0352] In at least one embodiment, each processing cluster 2014 may include an MMU 2045 (memory management unit) that is configured to map virtual addresses into physical addresses. In at least one embodiment, one or more instances of MMU 2045 may reside within memory interface 2018 of FIG. 20A. In at least one embodiment, MMU 2045 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 2045 may include address translation lookaside buffers (TLB) or caches that may reside within graphics multiprocessor 2034 or L1 2048 cache or processing cluster 2014. 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.

[0353] In at least one embodiment, a processing cluster 2014 may be configured such that each graphics multiprocessor 2034 is coupled to a texture unit 2036 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 L1 cache (not shown) or from an L1 cache within graphics multiprocessor 2034 and is fetched from an L2 cache, local parallel processor memory, or system memory, as needed. In at least one embodiment, each graphics multiprocessor 2034 outputs processed tasks to data crossbar 2040 to provide processed task to another processing cluster 2014 for further processing or to store processed task in an L2 cache, local parallel processor memory, or system memory via memory crossbar 2016. In at least one embodiment, a preROP 2042 (pre-raster operations unit) is configured to receive data from graphics multiprocessor 2034, and direct data to ROP units, which may be located with partition units as described herein (e.g., partition units 2020A-2020N of FIG. 20A). In at least one embodiment, preROP 2042 unit can perform optimizations for color blending, organizing pixel color data, and performing address translations.

[0354] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in graphics processing cluster 2014 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.

[0355] FIG. 20D shows a graphics multiprocessor 2034 according to at least one embodiment. In at least one embodiment, graphics multiprocessor 2034 couples with pipeline manager 2032 of processing cluster 2014. In at least one embodiment, graphics multiprocessor 2034 has an execution pipeline including but not limited to an instruction cache 2052, an instruction unit 2054, an address mapping unit 2056, a register file 2058, one or more general purpose graphics processing unit (GPGPU) cores 2062, and one or more load/store units 2066. In at least one embodiment, GPGPU cores 2062 and load/store units 2066 are coupled with cache memory 2072 and shared memory 2070 via a memory and cache interconnect 2068.

[0356] In at least one embodiment, instruction cache 2052 receives a stream of instructions to execute from pipeline manager 2032. In at least one embodiment, instructions are cached in instruction cache 2052 and dispatched for execution by an instruction unit 2054. In at least one embodiment, instruction unit 2054 can dispatch instructions as thread groups (e.g., warps), with each thread of thread group assigned to a different execution unit within GPGPU cores 2062. 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 2056 can be used to translate addresses in a unified address space into a distinct memory address that can be accessed by load/store units 2066.

[0357] In at least one embodiment, register file 2058 provides a set of registers for functional units of graphics multiprocessor 2034. In at least one embodiment, register file 2058 provides temporary storage for operands connected to data paths of functional units (e.g., GPGPU cores 2062, load/store units 2066) of graphics multiprocessor 2034. In at least one embodiment, register file 2058 is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file 2058. In at least one embodiment, register file 2058 is divided between different warps being executed by graphics multiprocessor 2034.

[0358] In at least one embodiment, GPGPU cores 2062 can each include floating point units (FPUs) and/or integer arithmetic logic units (ALUs) that are used to execute instructions of graphics multiprocessor 2034. In at least one embodiment, GPGPU cores 2062 can be similar in architecture or can differ in architecture. In at least one embodiment, a first portion of GPGPU cores 2062 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 2034 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 2062 can also include fixed or special function logic.

[0359] In at least one embodiment, GPGPU cores 2062 include SIMD logic capable of performing a single instruction on multiple sets of data. In at least one embodiment, GPGPU cores 2062 can physically execute SIMD4, SIMD8, and SIMD16 instructions and logically execute SIMD1, SIMD2, and SIMD32 instructions. In at least one embodiment, SIMD 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 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.

[0360] In at least one embodiment, memory and cache interconnect 2068 is an interconnect network that connects each functional unit of graphics multiprocessor 2034 to register file 2058 and to shared memory 2070. In at least one embodiment, memory and cache interconnect 2068 is a crossbar interconnect that allows load/store unit 2066 to implement load and store operations between shared memory 2070 and register file 2058. In at least one embodiment, register file 2058 can operate at a same frequency as GPGPU cores 2062, thus data transfer between GPGPU cores 2062 and register file 2058 can have very low latency. In at least one embodiment, shared memory 2070 can be used to enable communication between threads that execute on functional units within graphics multiprocessor 2034. In at least one embodiment, cache memory 2072 can be used as a data cache for example, to cache texture data communicated between functional units and texture unit 2036. In at least one embodiment, shared memory 2070 can also be used as a program managed cache. In at least one embodiment, threads executing on GPGPU cores 2062 can programmatically store data

within shared memory in addition to automatically cached data that is stored within cache memory 2072.

[0361] 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-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 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.

[0362] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in graphics multiprocessor 2034 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.

[0363] FIG. 21 illustrates a multi-GPU computing system 2100, according to at least one embodiment. In at least one embodiment, multi-GPU computing system 2100 can include a processor 2102 coupled to multiple general purpose graphics processing units (GPGPUs) 2106A-D via a host interface switch 2104. In at least one embodiment, host interface switch 2104 is a PCI express switch device that couples processor 2102 to a PCI express bus over which processor 2102 can communicate with GPGPUs 2106A-D. In at least one embodiment, GPGPUs 2106A-D can interconnect via a set of high-speed point-to-point GPU-to-GPU links 2116. In at least one embodiment, GPU-to-GPU links 2116 connect to each of GPGPUs 2106A-D via a dedicated GPU link. In at least one embodiment, P2P GPU links 2116 enable direct communication between each of GPGPUs 2106A-D without requiring communication over host interface bus 2104 to which processor 2102 is connected. In at least one embodiment, with GPU-to-GPU traffic directed to P2P GPU links 2116, host interface bus 2104 remains available for system memory access or to communicate with other instances of multi-GPU computing system 2100, for example, via one or more network devices. While in

at least one embodiment GPGPUs 2106A-D connect to processor 2102 via host interface switch 2104, in at least one embodiment processor 2102 includes direct support for P2P GPU links 2116 and can connect directly to GPGPUs 2106A-D.

[0364] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in multi-GPU computing system 1500 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.

[0365] FIG. 22 is a block diagram of a graphics processor 2200, according to at least one embodiment. In at least one embodiment, graphics processor 2200 includes a ring interconnect 2202, a pipeline front-end 2204, a media engine 2237, and graphics cores 2280A-2280N. In at least one embodiment, ring interconnect 2202 couples graphics processor 2200 to other processing units, including other graphics processors or one or more general-purpose processor cores. In at least one embodiment, graphics processor 2200 is one of many processors integrated within a multi-core processing system.

[0366] In at least one embodiment, graphics processor 2200 receives batches of commands via ring interconnect 2202. In at least one embodiment, incoming commands are interpreted by a command streamer 2203 in pipeline front-end 2204. In at least one embodiment, graphics processor 2200 includes scalable execution logic to perform 3D geometry processing and media processing via graphics core(s) 2280A-2280N. In at least one embodiment, for 3D geometry processing commands, command streamer 2203 supplies commands to geometry pipeline 2236. In at least one embodiment, for at least some media processing commands, command streamer 2203 supplies commands to a video front end 2234, which couples with media engine 2237. In at least one embodiment, media engine 2237 includes a Video Quality Engine (VQE) 2230 for video and image post-processing and a multi-format encode/decode (MFX) 2233 engine to provide hardware-accelerated media data encoding and decoding. In at least one embodiment, geometry pipeline 2236 and media engine 2237 each generate execution threads for thread execution resources provided by at least one graphics core 2280.

[0367] In at least one embodiment, graphics processor 2200 includes scalable thread execution resources featuring graphics cores 2280A-2280N (which can be modular and are sometimes referred to as core slices), each having multiple sub-cores 2250A-50N, 2260A-

2260N (sometimes referred to as core sub-slices). In at least one embodiment, graphics processor 2200 can have any number of graphics cores 2280A. In at least one embodiment, graphics processor 2200 includes a graphics core 2280A having at least a first sub-core 2250A and a second sub-core 2260A. In at least one embodiment, graphics processor 2200 is a low power processor with a single sub-core (e.g., 2250A). In at least one embodiment, graphics processor 2200 includes multiple graphics cores 2280A-2280N, each including a set of first sub-cores 2250A-2250N and a set of second sub-cores 2260A-2260N. In at least one embodiment, each sub-core in first sub-cores 2250A-2250N includes at least a first set of execution units 2252A-2252N and media/texture samplers 2254A-2254N. In at least one embodiment, each sub-core in second sub-cores 2260A-2260N includes at least a second set of execution units 2262A-2262N and samplers 2264A-2264N. In at least one embodiment, each sub-core 2250A-2250N, 2260A-2260N shares a set of shared resources 2270A-2270N. In at least one embodiment, shared resources include shared cache memory and pixel operation logic.

[0368] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, inference and/or training logic 115 may be used in graphics processor 2200 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.

[0369] FIG. 23 is a block diagram illustrating micro-architecture for a processor 2300 that may include logic circuits to perform instructions, according to at least one embodiment. In at least one embodiment, processor 2300 may perform instructions, including x86 instructions, ARM instructions, specialized instructions for application-specific integrated circuits (ASICs), etc. In at least one embodiment, processor 2300 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, 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 2300

may perform instructions to accelerate machine learning or deep learning algorithms, training, or inferencing.

[0370] In at least one embodiment, processor 2300 includes an in-order front end (“front end”) 2301 to fetch instructions to be executed and prepare instructions to be used later in a processor pipeline. In at least one embodiment, front end 2301 may include several units. In at least one embodiment, an instruction prefetcher 2326 fetches instructions from memory and feeds instructions to an instruction decoder 2328 which in turn decodes or interprets instructions. For example, in at least one embodiment, instruction decoder 2328 decodes a received instruction into one or more operations called “micro-instructions” or “micro-operations” (also called “micro ops” or “uops”) that a machine may execute. In at least one embodiment, instruction decoder 2328 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 2330 may assemble decoded uops into program ordered sequences or traces in a uop queue 2334 for execution. In at least one embodiment, when trace cache 2330 encounters a complex instruction, a microcode ROM 2332 provides uops needed to complete an operation.

[0371] 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 2328 may access microcode ROM 2332 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 2328. In at least one embodiment, an instruction may be stored within microcode ROM 2332 should a number of micro-ops be needed to accomplish such operation. In at least one embodiment, trace cache 2330 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 2332 in accordance with at least one embodiment. In at least one embodiment, after microcode ROM 2332 finishes sequencing micro-ops for an instruction, front end 2301 of a machine may resume fetching micro-ops from trace cache 2330.

[0372] In at least one embodiment, out-of-order execution engine (“out of order engine”) 2303 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 2303 includes, without limitation, an

allocator/register renamer 2340, a memory uop queue 2342, an integer/floating point uop queue 2344, a memory scheduler 2346, a fast scheduler 2302, a slow/general floating point scheduler (“slow/general FP scheduler”) 2304, and a simple floating point scheduler (“simple FP scheduler”) 2306. In at least one embodiment, fast scheduler 2302, slow/general floating point scheduler 2304, and simple floating point scheduler 2306 are also collectively referred to herein as “uop schedulers 2302, 2304, 2306.” In at least one embodiment, allocator/register renamer 2340 allocates machine buffers and resources that each uop needs in order to execute. In at least one embodiment, allocator/register renamer 2340 renames logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer 2340 also allocates an entry for each uop in one of two uop queues, memory uop queue 2342 for memory operations and integer/floating point uop queue 2344 for non-memory operations, in front of memory scheduler 2346 and uop schedulers 2302, 2304, 2306. In at least one embodiment, uop schedulers 2302, 2304, 2306, 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 2302 may schedule on each half of a main clock cycle while slow/general floating point scheduler 2304 and simple floating point scheduler 2306 may schedule once per main processor clock cycle. In at least one embodiment, uop schedulers 2302, 2304, 2306 arbitrate for dispatch ports to schedule uops for execution.

[0373] In at least one embodiment, execution block 2311 includes, without limitation, an integer register file/bypass network 2308, a floating point register file/bypass network (“FP register file/bypass network”) 2310, address generation units (“AGUs”) 2312 and 2314, fast Arithmetic Logic Units (ALUs) (“fast ALUs”) 2316 and 2318, a slow Arithmetic Logic Unit (“slow ALU”) 2320, a floating point ALU (“FP”) 2322, and a floating point move unit (“FP move”) 2324. In at least one embodiment, integer register file/bypass network 2308 and floating point register file/bypass network 2310 are also referred to herein as “register files 2308, 2310.” In at least one embodiment, AGUs 2312 and 2314, fast ALUs 2316 and 2318, slow ALU 2320, floating point ALU 2322, and floating point move unit 2324 are also referred to herein as “execution units 2312, 2314, 2316, 2318, 2320, 2322, and 2324.” In at least one embodiment, execution block 2311 may include, without limitation, any number (including zero) and type of register files, bypass networks, address generation units, and execution units, in any combination.

[0374] In at least one embodiment, register networks 2308, 2310 may be arranged between uop schedulers 2302, 2304, 2306, and execution units 2312, 2314, 2316, 2318, 2320, 2322,

and 2324. In at least one embodiment, integer register file/bypass network 2308 performs integer operations. In at least one embodiment, floating point register file/bypass network 2310 performs floating point operations. In at least one embodiment, each of register networks 2308, 2310 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 2308, 2310 may communicate data with each other. In at least one embodiment, integer register file/bypass network 2308 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 2310 may include, without limitation, 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

[0375] In at least one embodiment, execution units 2312, 2314, 2316, 2318, 2320, 2322, 2324 may execute instructions. In at least one embodiment, register networks 2308, 2310 store integer and floating point data operand values that micro-instructions need to execute. In at least one embodiment, processor 2300 may include, without limitation, any number and combination of execution units 2312, 2314, 2316, 2318, 2320, 2322, 2324. In at least one embodiment, floating point ALU 2322 and floating point move unit 2324, 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 2322 may 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 2316, 2318. In at least one embodiment, fast ALUS 2316, 2318 may execute fast operations with an effective latency of half a clock cycle. In at least one embodiment, most complex integer operations go to slow ALU 2320 as slow ALU 2320 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 2312, 2314. In at least one embodiment, fast ALU 2316, fast ALU 2318, and slow ALU 2320 may perform integer operations on 64-bit data operands. In at least one embodiment, fast ALU 2316, fast ALU 2318, and slow ALU 2320 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 2322 and floating point move unit 2324 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 SIMD and multimedia instructions.

[0376] In at least one embodiment, uop schedulers 2302, 2304, 2306 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 2300, processor 2300 may also include logic to handle memory misses. In at least one embodiment, if a data load misses in a 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 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.

[0377] In at least one embodiment, “registers” may refer to on-board processor storage 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 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 packed data.

[0378] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment portions or all of inference and/or training logic 115 may be incorporated into execution block 2311 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 2311. 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 2311 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0379] FIG. 24 illustrates a deep learning application processor 2400, according to at least one embodiment. In at least one embodiment, deep learning application processor 2400 uses instructions that, if executed by deep learning application processor 2400, cause deep learning application processor 2400 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, deep learning application processor 2400 is an application-specific integrated circuit (ASIC). In at least one embodiment, application processor 2400 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 2400 includes, without limitation, processing clusters 2410(1)-2410(12), Inter-Chip Links (“ICLs”) 2420(1)-2420(12), Inter-Chip Controllers (“ICCs”) 2430(1)-2430(2), high-bandwidth memory second generation (“HBM2”) 2440(1)-2440(4), memory controllers (“Mem Ctrlrs”) 2442(1)-2442(4), high bandwidth memory physical layer (“HBM PHY”) 2444(1)-2444(4), a management-controller central processing unit (“management-controller CPU”) 2450, a Serial Peripheral Interface, Inter-Integrated Circuit, and General Purpose Input/Output block (“SPI, I²C, GPIO”) 2460, a peripheral component interconnect express controller and direct memory access block (“PCIe Controller and DMA”) 2470, and a sixteen-lane peripheral component interconnect express port (“PCI Express x 16”) 2480.

[0380] In at least one embodiment, processing clusters 2410 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 embodiment, each processing cluster 2410 may include, without limitation, any number and type of processors. In at least one embodiment, deep learning application processor 2400 may include any number and type of processing clusters. In at least one embodiment, Inter-Chip Links 2420 are bi-directional. In at least one embodiment, Inter-Chip Links 2420 and Inter-Chip Controllers 2430 enable multiple deep learning application processors 2400 to 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 2400 may include any number (including zero) and type of ICLs 2420 and ICCs 2430.

[0381] In at least one embodiment, HBM2s 2440 provide a total of 32 Gigabytes (GB) of memory. In at least one embodiment, HBM2 2440(i) is associated with both memory

controller 2442(i) and HBM PHY 2444(i) where “i” is an arbitrary integer. In at least one embodiment, any number of HBM2s 2440 may provide any type and total amount of high bandwidth memory and may be associated with any number (including zero) and type of memory controllers 2442 and HBM PHYs 2444. In at least one embodiment, SPI, I²C, GPIO 2460, PCIe Controller and DMA 2470, and/or PCIe 2480 may be replaced with any number and type of blocks that enable any number and type of communication standards in any technically feasible fashion.

[0382] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. 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 application processor 2400. In at least one embodiment, deep learning application processor 2400 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 2400. In at least one embodiment, processor 2400 may be used to perform one or more neural network use cases described herein.

[0383] FIG. 25 is a block diagram of a neuromorphic processor 2500, according to at least one embodiment. In at least one embodiment, neuromorphic processor 2500 may receive one or more inputs from sources external to neuromorphic processor 2500. In at least one embodiment, these inputs may be transmitted to one or more neurons 2502 within neuromorphic processor 2500. In at least one embodiment, neurons 2502 and components thereof may be implemented using circuitry or logic, including one or more arithmetic logic units (ALUs). In at least one embodiment, neuromorphic processor 2500 may include, without limitation, thousands or millions of instances of neurons 2502, but any suitable number of neurons 2502 may be used. In at least one embodiment, each instance of neuron 2502 may include a neuron input 2504 and a neuron output 2506. In at least one embodiment, neurons 2502 may generate outputs that may be transmitted to inputs of other instances of neurons 2502. For example, in at least one embodiment, neuron inputs 2504 and neuron outputs 2506 may be interconnected via synapses 2508.

[0384] In at least one embodiment, neurons 2502 and synapses 2508 may be interconnected such that neuromorphic processor 2500 operates to process or analyze information received by neuromorphic processor 2500. In at least one embodiment, neurons 2502 may transmit an output pulse (or “fire” or “spike”) when inputs received through neuron input 2504 exceed a

threshold. In at least one embodiment, neurons 2502 may sum or integrate signals received at neuron inputs 2504. For example, in at least one embodiment, neurons 2502 may be implemented as leaky integrate-and-fire neurons, wherein if a sum (referred to as a “membrane potential”) exceeds a threshold value, neuron 2502 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 2504 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 2504 rapidly enough to exceed a threshold value (i.e., before a membrane potential decays too low to fire). In at least one embodiment, neurons 2502 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 2502 may include, without limitation, comparator circuits or logic that generate an output spike at neuron output 2506 when result of applying a transfer function to neuron input 2504 exceeds a threshold. In at least one embodiment, once neuron 2502 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 2502 may resume normal operation after a suitable period of time (or refractory period).

[0385] In at least one embodiment, neurons 2502 may be interconnected through synapses 2508. In at least one embodiment, synapses 2508 may operate to transmit signals from an output of a first neuron 2502 to an input of a second neuron 2502. In at least one embodiment, neurons 2502 may transmit information over more than one instance of synapse 2508. In at least one embodiment, one or more instances of neuron output 2506 may be connected, via an instance of synapse 2508, to an instance of neuron input 2504 in same neuron 2502. In at least one embodiment, an instance of neuron 2502 generating an output to be transmitted over an instance of synapse 2508 may be referred to as a “pre-synaptic neuron” with respect to that instance of synapse 2508. In at least one embodiment, an instance of neuron 2502 receiving an input transmitted over an instance of synapse 2508 may be referred to as a “post-synaptic neuron” with respect to that instance of synapse 2508. Because an instance of neuron 2502 may receive inputs from one or more instances of synapse 2508, and may also transmit outputs over one or more instances of synapse 2508, a single instance of neuron 2502 may therefore be both a “pre-synaptic neuron” and “post-

synaptic neuron,” with respect to various instances of synapses 2508, in at least one embodiment.

[0386] In at least one embodiment, neurons 2502 may be organized into one or more layers. In at least one embodiment, each instance of neuron 2502 may have one neuron output 2506 that may fan out through one or more synapses 2508 to one or more neuron inputs 2504. In at least one embodiment, neuron outputs 2506 of neurons 2502 in a first layer 2510 may be connected to neuron inputs 2504 of neurons 2502 in a second layer 2512. In at least one embodiment, layer 2510 may be referred to as a “feed-forward layer.” In at least one embodiment, each instance of neuron 2502 in an instance of first layer 2510 may fan out to each instance of neuron 2502 in second layer 2512. In at least one embodiment, first layer 2510 may be referred to as a “fully connected feed-forward layer.” In at least one embodiment, each instance of neuron 2502 in an instance of second layer 2512 may fan out to fewer than all instances of neuron 2502 in a third layer 2514. In at least one embodiment, second layer 2512 may be referred to as a “sparsely connected feed-forward layer.” In at least one embodiment, neurons 2502 in second layer 2512 may fan out to neurons 2502 in multiple other layers, including to neurons 2502 also in second layer 2512. In at least one embodiment, second layer 2512 may be referred to as a “recurrent layer.” In at least one embodiment, neuromorphic processor 2500 may include, without limitation, any suitable combination of recurrent layers and feed-forward layers, including, without limitation, both sparsely connected feed-forward layers and fully connected feed-forward layers.

[0387] In at least one embodiment, neuromorphic processor 2500 may include, without limitation, a reconfigurable interconnect architecture or dedicated hard-wired interconnects to connect synapse 2508 to neurons 2502. In at least one embodiment, neuromorphic processor 2500 may include, without limitation, circuitry or logic that allows synapses to be allocated to different neurons 2502 as needed based on neural network topology and neuron fan-in/out. For example, in at least one embodiment, synapses 2508 may be connected to neurons 2502 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.

[0388] FIG. 26 is a block diagram of a processing system, according to at least one embodiment. In at least one embodiment, system 2600 includes one or more processors 2602 and one or more graphics processors 2608, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 2602 or processor cores 2607. In at least one embodiment, system 2600 is a processing

platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

[0389] In at least one embodiment, system 2600 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 2600 is a mobile phone, a smart phone, a tablet computing device or a mobile Internet device. In at least one embodiment, processing system 2600 can also include, 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 2600 is a television or set top box device having one or more processors 2602 and a graphical interface generated by one or more graphics processors 2608.

[0390] In at least one embodiment, one or more processors 2602 each include one or more processor cores 2607 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 2607 is configured to process a specific instruction sequence 2609. In at least one embodiment, instruction sequence 2609 may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). In at least one embodiment, processor cores 2607 may each process a different instruction sequence 2609, which may include instructions to facilitate emulation of other instruction sequences. In at least one embodiment, processor core 2607 may also include other processing devices, such a Digital Signal Processor (DSP).

[0391] In at least one embodiment, processor 2602 includes a cache memory 2604. In at least one embodiment, processor 2602 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 2602. In at least one embodiment, processor 2602 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among processor cores 2607 using known cache coherency techniques. In at least one embodiment, a register file 2606 is additionally included in processor 2602, which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). In at least one embodiment, register file 2606 may include general-purpose registers or other registers.

[0392] In at least one embodiment, one or more processor(s) 2602 are coupled with one or more interface bus(es) 2610 to transmit communication signals such as address, data, or control signals between processor 2602 and other components in system 2600. In at least one embodiment, interface bus 2610 can be a processor bus, such as a version of a Direct Media Interface (DMI) bus. In at least one embodiment, interface bus 2610 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) 2602 include an integrated memory controller 2616 and a platform controller hub 2630. In at least one embodiment, memory controller 2616 facilitates communication between a memory device and other components of system 2600, while platform controller hub (PCH) 2630 provides connections to I/O devices via a local I/O bus.

[0393] In at least one embodiment, a memory device 2620 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 2620 can operate as system memory for system 2600, to store data 2622 and instructions 2621 for use when one or more processors 2602 executes an application or process. In at least one embodiment, memory controller 2616 also couples with an optional external graphics processor 2612, which may communicate with one or more graphics processors 2608 in processors 2602 to perform graphics and media operations. In at least one embodiment, a display device 2611 can connect to processor(s) 2602. In at least one embodiment, display device 2611 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 2611 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.

[0394] In at least one embodiment, platform controller hub 2630 enables peripherals to connect to memory device 2620 and processor 2602 via a high-speed I/O bus. In at least one embodiment, I/O peripherals include, but are not limited to, an audio controller 2646, a network controller 2634, a firmware interface 2628, a wireless transceiver 2626, touch sensors 2625, a data storage device 2624 (e.g., hard disk drive, flash memory, etc.). In at least one embodiment, data storage device 2624 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 2625 can include touch screen sensors,

pressure sensors, or fingerprint sensors. In at least one embodiment, wireless transceiver 2626 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 2628 enables communication with system firmware, and can be, for example, a unified extensible firmware interface (UEFI). In at least one embodiment, network controller 2634 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 2610. In at least one embodiment, audio controller 2646 is a multi-channel high definition audio controller. In at least one embodiment, system 2600 includes an optional legacy I/O controller 2640 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to system 2600. In at least one embodiment, platform controller hub 2630 can also connect to one or more Universal Serial Bus (USB) controllers 2642 connect input devices, such as keyboard and mouse 2643 combinations, a camera 2644, or other USB input devices.

[0395] In at least one embodiment, an instance of memory controller 2616 and platform controller hub 2630 may be integrated into a discreet external graphics processor, such as external graphics processor 2612. In at least one embodiment, platform controller hub 2630 and/or memory controller 2616 may be external to one or more processor(s) 2602. For example, in at least one embodiment, system 2600 can include an external memory controller 2616 and platform controller hub 2630, which may be configured as a memory controller hub and peripheral controller hub within a system chipset that is in communication with processor(s) 2602.

[0396] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment portions or all of inference and/or training logic 115 may be incorporated into system 2600. 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 described herein may be done using logic other than logic illustrated in FIGS. 1A or 1B. 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 2600 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0397] FIG. 27 is a block diagram of a processor 2700 having one or more processor cores 2702A-2702N, an integrated memory controller 2714, and an integrated graphics processor 2708, according to at least one embodiment. In at least one embodiment, processor 2700 can include additional cores up to and including additional core 2702N represented by dashed lined boxes. In at least one embodiment, each of processor cores 2702A-2702N includes one or more internal cache units 2704A-2704N. In at least one embodiment, each processor core also has access to one or more shared cache units 2706.

[0398] In at least one embodiment, internal cache units 2704A-2704N and shared cache units 2706 represent a cache memory hierarchy within processor 2700. In at least one embodiment, cache memory units 2704A-2704N 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 2706 and 2704A-2704N.

[0399] In at least one embodiment, processor 2700 may also include a set of one or more bus controller units 2716 and a system agent core 2710. In at least one embodiment, bus controller units 2716 manage a set of peripheral buses, such as one or more PCI or PCI express busses. In at least one embodiment, system agent core 2710 provides management functionality for various processor components. In at least one embodiment, system agent core 2710 includes one or more integrated memory controllers 2714 to manage access to various external memory devices (not shown).

[0400] In at least one embodiment, one or more of processor cores 2702A-2702N include support for simultaneous multi-threading. In at least one embodiment, system agent core 2710 includes components for coordinating and operating cores 2702A-2702N during multi-threaded processing. In at least one embodiment, system agent core 2710 may additionally include a power control unit (PCU), which includes logic and components to regulate one or more power states of processor cores 2702A-2702N and graphics processor 2708.

[0401] In at least one embodiment, processor 2700 additionally includes graphics processor 2708 to execute graphics processing operations. In at least one embodiment, graphics processor 2708 couples with shared cache units 2706, and system agent core 2710, including one or more integrated memory controllers 2714. In at least one embodiment, system agent core 2710 also includes a display controller 2711 to drive graphics processor output to one or more coupled displays. In at least one embodiment, display controller 2711 may also be a

separate module coupled with graphics processor 2708 via at least one interconnect, or may be integrated within graphics processor 2708.

[0402] In at least one embodiment, a ring-based interconnect unit 2712 is used to couple internal components of processor 2700. 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 2708 couples with ring interconnect 2712 via an I/O link 2713.

[0403] In at least one embodiment, I/O link 2713 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 2718, such as an eDRAM module. In at least one embodiment, each of processor cores 2702A-2702N and graphics processor 2708 use embedded memory module 2718 as a shared Last Level Cache.

[0404] In at least one embodiment, processor cores 2702A-2702N are homogeneous cores executing a common instruction set architecture. In at least one embodiment, processor cores 2702A-2702N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 2702A-2702N execute a common instruction set, while one or more other cores of processor cores 2702A-2702N executes a subset of a common instruction set or a different instruction set. In at least one embodiment, processor cores 2702A-2702N 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 2700 can be implemented on one or more chips or as an SoC integrated circuit.

[0405] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment portions or all of inference and/or training logic 115 may be incorporated into graphics processor 2708. 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) 2702, shared function logic, or other logic in FIG. 27. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 1A or 1B. 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 2700 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0406] FIG. 28 is a block diagram of a graphics processor 2800, 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 2800 communicates via a memory mapped I/O interface to registers on graphics processor 2800 and with commands placed into memory. In at least one embodiment, graphics processor 2800 includes a memory interface 2814 to access memory. In at least one embodiment, memory interface 2814 is an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0407] In at least one embodiment, graphics processor 2800 also includes a display controller 2802 to drive display output data to a display device 2820. In at least one embodiment, display controller 2802 includes hardware for one or more overlay planes for display device 2820 and composition of multiple layers of video or user interface elements. In at least one embodiment, display device 2820 can be an internal or external display device. In at least one embodiment, display device 2820 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 2800 includes a video codec engine 2806 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 Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

[0408] In at least one embodiment, graphics processor 2800 includes a block image transfer (BLIT) engine 2804 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) 2810. In at least one embodiment, GPE 2810 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0409] In at least one embodiment, GPE 2810 includes a 3D pipeline 2812 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 2812 includes programmable and fixed function elements that perform various tasks and/or spawn execution threads to a 3D/Media sub-system 2815. While 3D pipeline 2812 can be used to perform media operations, in at least one embodiment, GPE 2810 also includes a media pipeline 2816 that is used to perform media operations, such as video post-processing and image enhancement.

[0410] In at least one embodiment, media pipeline 2816 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 2806. In at least one embodiment, media pipeline 2816 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 2815. 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 2815.

[0411] In at least one embodiment, 3D/Media subsystem 2815 includes logic for executing threads spawned by 3D pipeline 2812 and media pipeline 2816. In at least one embodiment, 3D pipeline 2812 and media pipeline 2816 send thread execution requests to 3D/Media subsystem 2815, 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 2815 includes one or more internal caches for thread instructions and data. In at least one embodiment, subsystem 2815 also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

[0412] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment portions or all of inference and/or training logic 115 may be incorporated into graphics processor 2800. 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 2812. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 1A or 1B. 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 2800 to

perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0413] FIG. 29 is a block diagram of a graphics processing engine 2910 of a graphics processor in accordance with at least one embodiment. In at least one embodiment, graphics processing engine (GPE) 2910 is a version of GPE 2810 shown in FIG. 28. In at least one embodiment, a media pipeline 2916 is optional and may not be explicitly included within GPE 2910. In at least one embodiment, a separate media and/or image processor is coupled to GPE 2910.

[0414] In at least one embodiment, GPE 2910 is coupled to or includes a command streamer 2903, which provides a command stream to a 3D pipeline 2912 and/or media pipeline 2916. In at least one embodiment, command streamer 2903 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 2903 receives commands from memory and sends commands to 3D pipeline 2912 and/or media pipeline 2916. In at least one embodiment, commands are instructions, primitives, or micro-operations fetched from a ring buffer, which stores commands for 3D pipeline 2912 and media pipeline 2916. 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 2912 can also include references to data stored in memory, such as, but not limited to, vertex and geometry data for 3D pipeline 2912 and/or image data and memory objects for media pipeline 2916. In at least one embodiment, 3D pipeline 2912 and media pipeline 2916 process commands and data by performing operations or by dispatching one or more execution threads to a graphics core array 2914. In at least one embodiment, graphics core array 2914 includes one or more blocks of graphics cores (e.g., graphics core(s) 2915A, graphics core(s) 2915B), 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 115 in FIG. 1A and FIG. 1B.

[0415] In at least one embodiment, 3D pipeline 2912 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 2914. In at least one embodiment, graphics core array 2914 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) 2915A-2915B of graphic core array 2914 includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

[0416] In at least one embodiment, graphics core array 2914 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.

[0417] In at least one embodiment, output data generated by threads executing on graphics core array 2914 can output data to memory in a unified return buffer (URB) 2918. In at least one embodiment, URB 2918 can store data for multiple threads. In at least one embodiment, URB 2918 may be used to send data between different threads executing on graphics core array 2914. In at least one embodiment, URB 2918 may additionally be used for synchronization between threads on graphics core array 2914 and fixed function logic within shared function logic 2920.

[0418] In at least one embodiment, graphics core array 2914 is scalable, such that graphics core array 2914 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 2910. In at least one embodiment, execution resources are dynamically scalable, such that execution resources may be enabled or disabled as needed.

[0419] In at least one embodiment, graphics core array 2914 is coupled to shared function logic 2920 that includes multiple resources that are shared between graphics cores in graphics core array 2914. In at least one embodiment, shared functions performed by shared function logic 2920 are embodied in hardware logic units that provide specialized supplemental functionality to graphics core array 2914. In at least one embodiment, shared function logic 2920 includes but is not limited to a sampler unit 2921, a math unit 2922, and inter-thread communication (ITC) logic 2929. In at least one embodiment, one or more cache(s) 2925 are included in, or coupled to, shared function logic 2920.

[0420] In at least one embodiment, a shared function is used if demand for a specialized function is insufficient for inclusion within graphics core array 2914. In at least one embodiment, a single instantiation of a specialized function is used in shared function logic 2920 and shared among other execution resources within graphics core array 2914. In at least one embodiment, specific shared functions within shared function logic 2920 that are used

extensively by graphics core array 2914 may be included within shared function logic 2920 within graphics core array 2914. In at least one embodiment, shared function logic 2920 within graphics core array 2914 can include some or all logic within shared function logic 2920. In at least one embodiment, all logic elements within shared function logic 2920 may be duplicated within shared function logic 2926 of graphics core array 2914. In at least one embodiment, shared function logic 2920 is excluded in favor of shared function logic 2926 within graphics core array 2914.

[0421] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment portions or all of inference and/or training logic 115 may be incorporated into graphics processor 2910. 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 2912, graphics core(s) 2915, shared function logic 2926, shared function logic 2920, 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. 1A or 1B. 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 2910 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0422] FIG. 30 is a block diagram of hardware logic of a graphics processor core 3000, according to at least one embodiment described herein. In at least one embodiment, graphics processor core 3000 is included within a graphics core array. In at least one embodiment, graphics processor core 3000, 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 3000 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 processor core 3000 can include a fixed function block 3030 coupled with multiple sub-cores 3001A-3001F, also referred to as sub-slices, that include modular blocks of general-purpose and fixed function logic.

[0423] In at least one embodiment, fixed function block 3030 includes a geometry and fixed function pipeline 3036 that can be shared by all sub-cores in graphics processor 3000, for example, in lower performance and/or lower power graphics processor implementations.

In at least one embodiment, geometry and fixed function pipeline 3036 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.

[0424] In at least one embodiment, fixed function block 3030 also includes a graphics SoC interface 3037, a graphics microcontroller 3038, and a media pipeline 3039. In at least one embodiment, graphics SoC interface 3037 provides an interface between graphics processor core 3000 and other processor cores within a system on a chip integrated circuit. In at least one embodiment, graphics microcontroller 3038 is a programmable sub-processor that is configurable to manage various functions of graphics processor 3000, including thread dispatch, scheduling, and pre-emption. In at least one embodiment, media pipeline 3039 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 3039 implements media operations via requests to compute or sampling logic within sub-cores 3001A-3001F.

[0425] In at least one embodiment, SoC interface 3037 enables graphics processor core 3000 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 3037 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 processor core 3000 and CPUs within an SoC. In at least one embodiment, graphics SoC interface 3037 can also implement power management controls for graphics processor core 3000 and enable an interface between a clock domain of graphics processor core 3000 and other clock domains within an SoC. In at least one embodiment, SoC interface 3037 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 3039, when media operations are to be performed, or a geometry and fixed function pipeline (e.g., geometry and fixed function pipeline 3036, and/or a geometry and fixed function pipeline 3014) when graphics processing operations are to be performed.

[0426] In at least one embodiment, graphics microcontroller 3038 can be configured to perform various scheduling and management tasks for graphics processor core 3000. In at least one embodiment, graphics microcontroller 3038 can perform graphics and/or compute

workload scheduling on various graphics parallel engines within execution unit (EU) arrays 3002A-3002F, 3004A-3004F within sub-cores 3001A-3001F. In at least one embodiment, host software executing on a CPU core of an SoC including graphics processor core 3000 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 3038 can also facilitate low-power or idle states for graphics processor core 3000, providing graphics processor core 3000 with an ability to save and restore registers within graphics processor core 3000 across low-power state transitions independently from an operating system and/or graphics driver software on a system.

[0427] In at least one embodiment, graphics processor core 3000 may have greater than or fewer than illustrated sub-cores 3001A-3001F, up to N modular sub-cores. For each set of N sub-cores, in at least one embodiment, graphics processor core 3000 can also include shared function logic 3010, shared and/or cache memory 3012, geometry/fixed function pipeline 3014, as well as additional fixed function logic 3016 to accelerate various graphics and compute processing operations. In at least one embodiment, shared function logic 3010 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 processor core 3000. In at least one embodiment, shared and/or cache memory 3012 can be a last-level cache for N sub-cores 3001A-3001F within graphics processor core 3000 and can also serve as shared memory that is accessible by multiple sub-cores. In at least one embodiment, geometry/fixed function pipeline 3014 can be included instead of geometry/fixed function pipeline 3036 within fixed function block 3030 and can include similar logic units.

[0428] In at least one embodiment, graphics processor core 3000 includes additional fixed function logic 3016 that can include various fixed function acceleration logic for use by graphics processor core 3000. In at least one embodiment, additional fixed function logic 3016 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 3014, 3036, and a cull pipeline, which is an additional geometry pipeline that may be included within additional fixed function logic 3016. 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 3016 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 instance may be referred to as a replay pipeline) can consume visibility information to skip culled triangles to shade only visible triangles that are finally passed to a rasterization phase.

[0429] In at least one embodiment, additional fixed function logic 3016 can also include machine-learning acceleration logic, such as fixed function matrix multiplication logic, for implementations including optimizations for machine learning training or inferencing.

[0430] In at least one embodiment, within each graphics sub-core 3001A-3001F 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 3001A-3001F include multiple EU arrays 3002A-3002F, 3004A-3004F, thread dispatch and inter-thread communication (TD/IC) logic 3003A-3003F, a 3D (e.g., texture) sampler 3005A-3005F, a media sampler 3006A-3006F, a shader processor 3007A-3007F, and shared local memory (SLM) 3008A-3008F. In at least one embodiment, EU arrays 3002A-3002F, 3004A-3004F 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 3003A-3003F 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 3005A-3005F 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 3006A-3006F can perform similar read operations based on a type and format

associated with media data. In at least one embodiment, each graphics sub-core 3001A-3001F can alternately include a unified 3D and media sampler. In at least one embodiment, threads executing on execution units within each of sub-cores 3001A-3001F can make use of shared local memory 3008A-3008F within each sub-core, to enable threads executing within a thread group to execute using a common pool of on-chip memory.

[0431] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, portions or all of inference and/or training logic 115 may be incorporated into graphics processor core 3000. 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 microcontroller 3038, geometry and fixed function pipeline 3014 and 3036, or other logic in FIG. 30. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 1A or 1B. 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 core 3000 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0432] FIGS. 31A-31B illustrate thread execution logic 3100 including an array of processing elements of a graphics processor core according to at least one embodiment. FIG. 31A illustrates at least one embodiment, in which thread execution logic 3100 is used. FIG. 31B illustrates exemplary internal details of a graphics execution unit 3108, according to at least one embodiment.

[0433] As illustrated in FIG. 31A, in at least one embodiment, thread execution logic 3100 includes a shader processor 3102, a thread dispatcher 3104, an instruction cache 3106, a scalable execution unit array including a plurality of execution units 3107A-3107N and 3108A-3108N, a sampler 3110, a data cache 3112, and a data port 3114. 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 3108A-N or 3107A-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 3100 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 3106, data port 3114, sampler 3110, and execution units 3107 or 3108. In at least one

embodiment, each execution unit (e.g., 3107A) 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 3107 and/or 3108 is scalable to include any number individual execution units.

[0434] In at least one embodiment, execution units 3107 and/or 3108 are primarily used to execute shader programs. In at least one embodiment, shader processor 3102 can process various shader programs and dispatch execution threads associated with shader programs via a thread dispatcher 3104. In at least one embodiment, thread dispatcher 3104 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 3107 and/or 3108. 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 3104 can also process runtime thread spawning requests from executing shader programs.

[0435] In at least one embodiment, execution units 3107 and/or 3108 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 3107 and/or 3108, 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 3107 and/or 3108 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.

[0436] In at least one embodiment, each execution unit in execution units 3107 and/or 3108 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, an execution channel is a logical unit of execution for data element access, masking, and flow 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 3107 and/or 3108 support integer and floating-point data types.

[0437] 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, 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 vector widths and register sizes are possible.

[0438] In at least one embodiment, one or more execution units can be combined into a fused execution unit 3109A-3109N having thread control logic (3111A-3111N) that is common to fused EUs such as execution unit 3107A fused with execution unit 3108A into fused execution unit 3109A. In at least one embodiment, multiple EUs can be fused into an 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 3109A-3109N includes at least two execution units. For example, in at least one embodiment, fused execution unit 3109A includes a first EU 3107A, second EU 3108A, and thread control logic 3111A that is common to first EU 3107A and second EU 3108A. In at least one embodiment, thread control logic 3111A controls threads executed on fused graphics execution unit

3109A, allowing each EU within fused execution units 3109A-3109N to execute using a common instruction pointer register.

[0439] In at least one embodiment, one or more internal instruction caches (e.g., 3106) are included in thread execution logic 3100 to cache thread instructions for execution units. In at least one embodiment, one or more data caches (e.g., 3112) are included to cache thread data during thread execution. In at least one embodiment, sampler 3110 is included to provide texture sampling for 3D operations and media sampling for media operations. In at least one embodiment, sampler 3110 includes specialized texture or media sampling functionality to process texture or media data during sampling process before providing sampled data to an execution unit.

[0440] During execution, in at least one embodiment, graphics and media pipelines send thread initiation requests to thread execution logic 3100 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 3102 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 3102 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 3102 dispatches threads to an execution unit (e.g., 3108A) via thread dispatcher 3104. In at least one embodiment, shader processor 3102 uses texture sampling logic in sampler 3110 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.

[0441] In at least one embodiment, data port 3114 provides a memory access mechanism for thread execution logic 3100 to output processed data to memory for further processing on a graphics processor output pipeline. In at least one embodiment, data port 3114 includes or couples to one or more cache memories (e.g., data cache 3112) to cache data for memory access via a data port.

[0442] As illustrated in FIG. 31B, in at least one embodiment, a graphics execution unit 3108 can include an instruction fetch unit 3137, a general register file array (GRF) 3124, an architectural register file array (ARF) 3126, a thread arbiter 3122, a send unit 3130, a branch

unit 3132, a set of SIMD floating point units (FPUs) 3134, and a set of dedicated integer SIMD ALUs 3135. In at least one embodiment, GRF 3124 and ARF 3126 includes a set of general register files and architecture register files associated with each simultaneous hardware thread that may be active in graphics execution unit 3108. In at least one embodiment, per thread architectural state is maintained in ARF 3126, while data used during thread execution is stored in GRF 3124. 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 3126.

[0443] In at least one embodiment, graphics execution unit 3108 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.

[0444] In at least one embodiment, graphics execution unit 3108 can co-issue multiple instructions, which may each be different instructions. In at least one embodiment, thread arbiter 3122 of graphics execution unit thread 3108 can dispatch instructions to one of send unit 3130, branch unit 3132, or SIMD FPU(s) 3134 for execution. In at least one embodiment, each execution thread can access 128 general-purpose registers within GRF 3124, 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 3124, 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 3124 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.

[0445] 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 3130. In at least one embodiment, branch instructions are dispatched to branch unit 3132 to facilitate SIMD divergence and eventual convergence.

[0446] In at least one embodiment, graphics execution unit 3108 includes one or more SIMD floating point units (FPU(s)) 3134 to perform floating-point operations. In at least one embodiment, FPU(s) 3134 also support integer computation. In at least one embodiment, FPU(s) 3134 can SIMD execute up to M number of 32-bit floating-point (or integer) 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 3135 are also present, and may be specifically optimized to perform operations associated with machine learning computations.

[0447] In at least one embodiment, arrays of multiple instances of graphics execution unit 3108 can be instantiated in a graphics sub-core grouping (e.g., a sub-slice). In at least one embodiment, execution unit 3108 can execute instructions across a plurality of execution channels. In at least one embodiment, each thread executed on graphics execution unit 3108 is executed on a different channel.

[0448] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. In at least one embodiment, portions or all of inference and/or training logic 115 may be incorporated into thread execution logic 3100. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 1A or 1B. 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 3100 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0449] FIG. 32 illustrates a parallel processing unit (“PPU”) 3200, according to at least one embodiment. In at least one embodiment, PPU 3200 is configured with machine-readable code that, if executed by PPU 3200, causes PPU 3200 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, PPU 3200 is a multi-threaded processor that is implemented on one or more integrated circuit devices 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 execution and is an instantiation of a set of instructions configured to be executed by PPU 3200. In at least one embodiment, PPU 3200 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 3200 is utilized to perform computations such as linear algebra operations and machine-learning operations. FIG. 32 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.

[0450] In at least one embodiment, one or more PPUs 3200 are configured to accelerate High Performance Computing (“HPC”), data center, and machine learning applications. In at least one embodiment, PPU 3200 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.

[0451] In at least one embodiment, PPU 3200 includes, without limitation, an Input/Output (“I/O”) unit 3206, a front-end unit 3210, a scheduler unit 3212, a work distribution unit 3214, a hub 3216, a crossbar (“XBar”) 3220, one or more general processing clusters (“GPCs”) 3218, and one or more partition units (“memory partition units”) 3222. In at least one embodiment, PPU 3200 is connected to a host processor or other PPUs 3200 via one or more high-speed GPU interconnects (“GPU interconnects”) 3208. In at least one embodiment, PPU 3200 is connected to a host processor or other peripheral devices via a system bus 3202. In at least one embodiment, PPU 3200 is connected to a local memory comprising one or more memory devices (“memory”) 3204. In at least one embodiment, memory devices 3204 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.

[0452] In at least one embodiment, high-speed GPU interconnect 3208 may refer to a wire-based multi-lane communications link that is used by systems to scale and include one or more PPUs 3200 combined with one or more central processing units (“CPUs”), supports cache coherence between PPUs 3200 and CPUs, and CPU mastering. In at least one

embodiment, data and/or commands are transmitted by high-speed GPU interconnect 3208 through hub 3216 to/from other units of PPU 3200 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. 32.

[0453] In at least one embodiment, I/O unit 3206 is configured to transmit and receive communications (e.g., commands, data) from a host processor (not illustrated in FIG. 32) over system bus 3202. In at least one embodiment, I/O unit 3206 communicates with host processor directly via system bus 3202 or through one or more intermediate devices such as a memory bridge. In at least one embodiment, I/O unit 3206 may communicate with one or more other processors, such as one or more of PPUs 3200 via system bus 3202. In at least one embodiment, I/O unit 3206 implements a Peripheral Component Interconnect Express (“PCIe”) interface for communications over a PCIe bus. In at least one embodiment, I/O unit 3206 implements interfaces for communicating with external devices.

[0454] In at least one embodiment, I/O unit 3206 decodes packets received via system bus 3202. In at least one embodiment, at least some packets represent commands configured to cause PPU 3200 to perform various operations. In at least one embodiment, I/O unit 3206 transmits decoded commands to various other units of PPU 3200 as specified by commands. In at least one embodiment, commands are transmitted to front-end unit 3210 and/or transmitted to hub 3216 or other units of PPU 3200 such as one or more copy engines, a video encoder, a video decoder, a power management unit, etc. (not explicitly illustrated in FIG. 32). In at least one embodiment, I/O unit 3206 is configured to route communications between and among various logical units of PPU 3200.

[0455] In at least one embodiment, a program executed by host processor encodes a command stream in a buffer that provides workloads to PPU 3200 for processing. In at least one embodiment, a workload comprises instructions and data to be processed by those 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 3200 — a host interface unit may be configured to access that buffer in a system memory connected to system bus 3202 via memory requests transmitted over system bus 3202 by I/O unit 3206. In at least one 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 3200 such that front-end unit 3210 receives pointers to one or more command streams and manages one or more command streams, reading commands from command streams and forwarding commands to various units of PPU 3200.

[0456] In at least one embodiment, front-end unit 3210 is coupled to scheduler unit 3212 that configures various GPCs 3218 to process tasks defined by one or more command streams. In at least one embodiment, scheduler unit 3212 is configured to track state information related to various tasks managed by scheduler unit 3212 where state information may indicate which of GPCs 3218 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 3212 manages execution of a plurality of tasks on one or more of GPCs 3218.

[0457] In at least one embodiment, scheduler unit 3212 is coupled to work distribution unit 3214 that is configured to dispatch tasks for execution on GPCs 3218. In at least one embodiment, work distribution unit 3214 tracks a number of scheduled tasks received from scheduler unit 3212 and work distribution unit 3214 manages a pending task pool and an active task pool for each of GPCs 3218. 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 particular GPC 3218; an active task pool may comprise a number of slots (e.g., 4 slots) for tasks that are actively being processed by GPCs 3218 such that as one of GPCs 3218 completes execution of a task, that task is evicted from that active task pool for GPC 3218 and another task from a pending task pool is selected and scheduled for execution on GPC 3218. In at least one embodiment, if an active task is idle on GPC 3218, such as while waiting for a data dependency to be resolved, then that active task is evicted from GPC 3218 and returned to that pending task pool while another task in that pending task pool is selected and scheduled for execution on GPC 3218.

[0458] In at least one embodiment, work distribution unit 3214 communicates with one or more GPCs 3218 via XBar 3220. In at least one embodiment, XBar 3220 is an interconnect network that couples many of units of PPU 3200 to other units of PPU 3200 and can be configured to couple work distribution unit 3214 to a particular GPC 3218. In at least one embodiment, one or more other units of PPU 3200 may also be connected to XBar 3220 via hub 3216.

[0459] In at least one embodiment, tasks are managed by scheduler unit 3212 and dispatched to one of GPCs 3218 by work distribution unit 3214. In at least one embodiment, GPC 3218 is configured to process task and generate results. In at least one embodiment, results may be consumed by other tasks within GPC 3218, routed to a different GPC 3218 via XBar 3220, or stored in memory 3204. In at least one embodiment, results can be written to memory 3204 via partition units 3222, which implement a memory interface for reading and writing data to/from memory 3204. In at least one embodiment, results can be transmitted to

another PPU 3204 or CPU via high-speed GPU interconnect 3208. In at least one embodiment, PPU 3200 includes, without limitation, a number U of partition units 3222 that is equal to a number of separate and distinct memory devices 3204 coupled to PPU 3200, as described in more detail herein in conjunction with FIG. 34.

[0460] 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 3200. In at least one embodiment, multiple compute applications are simultaneously executed by PPU 3200 and PPU 3200 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 3200 and that driver kernel outputs tasks to one or more streams being processed by PPU 3200. 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. 34.

[0461] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. 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 PPU 3200. In at least one embodiment, PPU 3200 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 3200. In at least one embodiment, PPU 3200 may be used to perform one or more neural network use cases described herein.

[0462] FIG. 33 illustrates a general processing cluster (“GPC”) 3300, according to at least one embodiment. In at least one embodiment, GPC 3300 is GPC 3218 of FIG. 32. In at least one embodiment, each GPC 3300 includes, without limitation, a number of hardware units for processing tasks and each GPC 3300 includes, without limitation, a pipeline manager 3302, a pre-raster operations unit (“preROP”) 3304, a raster engine 3308, a work distribution

crossbar (“WDX”) 3316, a memory management unit (“MMU”) 3318, one or more Data Processing Clusters (“DPCs”) 3306, and any suitable combination of parts.

[0463] In at least one embodiment, operation of GPC 3300 is controlled by pipeline manager 3302. In at least one embodiment, pipeline manager 3302 manages configuration of one or more DPCs 3306 for processing tasks allocated to GPC 3300. In at least one embodiment, pipeline manager 3302 configures at least one of one or more DPCs 3306 to implement at least a portion of a graphics rendering pipeline. In at least one embodiment, DPC 3306 is configured to execute a vertex shader program on a programmable streaming multi-processor (“SM”) 3314. In at least one embodiment, pipeline manager 3302 is configured to route packets received from a work distribution unit to appropriate logical units within GPC 3300, in at least one embodiment, and some packets may be routed to fixed function hardware units in preROP 3304 and/or raster engine 3308 while other packets may be routed to DPCs 3306 for processing by a primitive engine 3312 or SM 3314. In at least one embodiment, pipeline manager 3302 configures at least one of DPCs 3306 to implement a neural network model and/or a computing pipeline.

[0464] In at least one embodiment, preROP unit 3304 is configured, in at least one embodiment, to route data generated by raster engine 3308 and DPCs 3306 to a Raster Operations (“ROP”) unit in partition unit 3222, described in more detail above in conjunction with FIG. 32. In at least one embodiment, preROP unit 3304 is configured to perform optimizations for color blending, organize pixel data, perform address translations, and more. In at least one embodiment, raster engine 3308 includes, without limitation, a number of fixed function hardware units configured to perform various raster operations, in at least one embodiment, and raster engine 3308 includes, without limitation, a setup engine, a coarse 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 3308 comprises fragments to be processed by any suitable entity, such as by a fragment shader implemented within DPC 3306.

[0465] In at least one embodiment, each DPC 3306 included in GPC 3300 comprises, without limitation, an M-Pipe Controller (“MPC”) 3310; primitive engine 3312; one or more SMs 3314; and any suitable combination thereof. In at least one embodiment, MPC 3310 controls operation of DPC 3306, routing packets received from pipeline manager 3302 to appropriate units in DPC 3306. In at least one embodiment, packets associated with a vertex are routed to primitive engine 3312, 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 3314.

[0466] In at least one embodiment, SM 3314 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 3314 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 3314 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 state is maintained for each warp, enabling concurrency between warps and serial 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 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 3314 is described in more detail herein.

[0467] In at least one embodiment, MMU 3318 provides an interface between GPC 3300 and a memory partition unit (e.g., partition unit 3222 of FIG. 32) and MMU 3318 provides translation of virtual addresses into physical addresses, memory protection, and arbitration of memory requests. In at least one embodiment, MMU 3318 provides one or more translation

lookaside buffers (“TLBs”) for performing translation of virtual addresses into physical addresses in memory.

[0468] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. 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 3300. In at least one embodiment, GPC 3300 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 GPC 3300. In at least one embodiment, GPC 3300 may be used to perform one or more neural network use cases described herein.

[0469] FIG. 34 illustrates a memory partition unit 3400 of a parallel processing unit (“PPU”), in accordance with at least one embodiment. In at least one embodiment, memory partition unit 3400 includes, without limitation, a Raster Operations (“ROP”) unit 3402, a level two (“L2”) cache 3404, a memory interface 3406, and any suitable combination thereof. In at least one embodiment, memory interface 3406 is coupled to memory. In at least one embodiment, memory interface 3406 may implement 32, 64, 134, 1024-bit data buses, or like, for high-speed data transfer. In at least one embodiment, PPU incorporates U memory interfaces 3406 where U is a positive integer, with one memory interface 3406 per pair of partition units 3400, where each pair of partition units 3400 is connected to a corresponding memory device. For example, in at least one embodiment, PPU may be connected to up to Y memory devices, such as high bandwidth memory stacks or graphics double-data-rate, version 5, synchronous dynamic random access memory (“GDDR5 SDRAM”).

[0470] In at least one embodiment, memory interface 3406 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.

[0471] In at least one embodiment, PPU implements a multi-level memory hierarchy. In at least one embodiment, memory partition unit 3400 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 3208 supports address translation services allowing PPU to directly access a CPU’s page tables and providing full access to CPU memory by a PPU.

[0472] 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 3400 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.

[0473] Data from memory 3204 of FIG. 32 or other system memory is fetched by memory partition unit 3400 and stored in L2 cache 3404, which is located on-chip and is shared between various GPCs, in accordance with at least one embodiment. Each memory partition unit 3400, 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 2714 in FIG. 33 may implement a Level 1 (“L1”) cache wherein that L1 cache is private memory that is dedicated to a particular SM 2714 and data from L2 cache 3404 is fetched and stored in each L1 cache for processing in functional units of SMs 2714. In at least one embodiment, L2 cache 3404 is coupled to memory interface 3406 and XBar 3220 shown in FIG. 32.

[0474] ROP unit 3402 performs graphics raster operations related to pixel color, such as color compression, pixel blending, and more, in at least one embodiment. ROP unit 3402, in at least one embodiment, implements depth testing in conjunction with raster engine 3308, receiving a depth for a sample location associated with a pixel fragment from a culling engine of raster engine 3308. 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 3402 updates depth buffer and transmits a result of that depth test to raster engine 3308. It will be appreciated that a number of partition units 3400 may be different than a number of GPCs and, therefore, each ROP unit 3402 can, in at least one embodiment, be coupled to each GPC. In at least one embodiment, ROP unit 3402 tracks packets received from different GPCs and determines whether a result generated by ROP unit 3402 is to be routed to through XBar 3220.

[0475] FIG. 35 illustrates a streaming multi-processor (“SM”) 3500, according to at least one embodiment. In at least one embodiment, SM 3500 is SM of FIG. 33. In at least one embodiment, SM 3500 includes, without limitation, an instruction cache 3502, one or more scheduler units 3504, a register file 3508, one or more processing cores (“cores”) 3510, one or more special function units (“SFUs”) 3512, one or more load/store units (“LSUs”) 3514, an interconnect network 3516, a shared memory/level one (“L1”) cache 3518, and/or any suitable combination thereof.

[0476] 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 associated with a shader program, that task is allocated to one of SMs 3500. In at least one embodiment, scheduler unit 3504 receives tasks from a work distribution unit and manages instruction scheduling for one or more thread blocks assigned to SM 3500. In at least one embodiment, scheduler unit 3504 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 3504 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 3510, SFUs 3512, and LSUs 3514) during each clock cycle.

[0477] 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., `synctreads()` 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.

[0478] In at least one embodiment, a dispatch unit 3506 is configured to transmit instructions to one or more functional units and scheduler unit 3504 and includes, without limitation, two dispatch units 3506 that enable two different instructions from a common warp to be dispatched during each clock cycle. In at least one embodiment, each scheduler unit 3504 includes a single dispatch unit 3506 or additional dispatch units 3506.

[0479] In at least one embodiment, each SM 3500, in at least one embodiment, includes, without limitation, register file 3508 that provides a set of registers for functional units of SM 3500. In at least one embodiment, register file 3508 is divided between each functional unit such that each functional unit is allocated a dedicated portion of register file 3508. In at least one embodiment, register file 3508 is divided between different warps being executed by SM 3500 and register file 3508 provides temporary storage for operands connected to data paths of functional units. In at least one embodiment, each SM 3500 comprises, without limitation, a plurality of L processing cores 3510, where L is a positive integer. In at least one embodiment, SM 3500 includes, without limitation, a large number (e.g., 128 or more) of distinct processing cores 3510. In at least one embodiment, each processing core 3510 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 3510 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.

[0480] 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 3510. 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.

[0481] 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 multiply. Tensor cores are used to perform much larger two-dimensional or higher 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 CUDA level, a warp-level interface assumes 16x16 size matrices spanning all 32 threads of warp.

[0482] In at least one embodiment, each SM 3500 comprises, without limitation, M SFUs 3512 that perform special functions (e.g., attribute evaluation, reciprocal square root, and like). In at least one embodiment, SFUs 3512 include, without limitation, a tree traversal unit configured to traverse a hierarchical tree data structure. In at least one embodiment, SFUs 3512 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 values for use in shader programs executed by SM 3500. In at least one embodiment, texture maps are stored in shared memory/L1 cache 3518. 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 3500 includes, without limitation, two texture units.

[0483] Each SM 3500 comprises, without limitation, N LSUs 3514 that implement load and store operations between shared memory/L1 cache 3518 and register file 3508, in at least one embodiment. Interconnect network 3516 connects each functional unit to register file 3508 and LSU 3514 to register file 3508 and shared memory/L1 cache 3518 in at least one embodiment. In at least one embodiment, interconnect network 3516 is a crossbar that can be configured to connect any functional units to any registers in register file 3508 and connect LSUs 3514 to register file 3508 and memory locations in shared memory/L1 cache 3518.

[0484] In at least one embodiment, shared memory/L1 cache 3518 is an array of on-chip memory that allows for data storage and communication between SM 3500 and primitive engine and between threads in SM 3500, in at least one embodiment. In at least one embodiment, shared memory/L1 cache 3518 comprises, without limitation, 128 KB of storage capacity and is in a path from SM 3500 to a partition unit. In at least one embodiment, shared memory/L1 cache 3518, in at least one embodiment, is used to cache reads and writes. In at least one embodiment, one or more of shared memory/L1 cache 3518, L2 cache, and memory are backing stores.

[0485] 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 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 3518 enables shared memory/L1 cache 3518 to function as a high-throughput conduit for streaming data while simultaneously providing high-bandwidth 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 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 3500 to execute program and perform calculations, shared memory/L1 cache 3518 to communicate between threads, and LSU 3514 to read and write global memory through shared memory/L1 cache 3518 and memory partition unit. In at least one embodiment, when

configured for general purpose parallel computation, SM 3500 writes commands that scheduler unit 3504 can use to launch new work on DPCs.

[0486] 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 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 management unit (“MMU”), a digital-to-analog converter (“DAC”), and like.

[0487] 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 configured to interface with a PCIe slot on a motherboard of a desktop computer. In at least one embodiment, that PPU may be an integrated graphics processing unit (“iGPU”) included in chipset of a motherboard.

[0488] Inference and/or training logic 115 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 115 are provided herein in conjunction with FIGS. 1A and/or 1B. 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 3500. In at least one embodiment, SM 3500 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 SM 3500. In at least one embodiment, SM 3500 may be used to perform one or more neural network use cases described herein.

[0489] Embodiments are disclosed related a virtualized computing platform for advanced computing.

[0490] With reference to FIG. 36, FIG. 36 of generating and deploying an image processing and inferencing pipeline, in accordance with at least one embodiment. In at least one embodiment, process 3600 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 3602, such as medical facilities, hospitals, healthcare institutes, clinics, research or diagnostic labs, etc. In at least one embodiment, process 3600 may be deployed to perform genomics analysis and inferencing on sequencing data. Examples of genomic analyses that may be performed using systems and processes described herein

include, without limitation, variant calling, mutation detection, and gene expression quantification.

[0491] In at least one embodiment, process 3600 may be executed within a training system 3604 and/or a deployment system 3606. In at least one embodiment, training system 3604 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 3606. In at least one embodiment, deployment system 3606 may be configured to offload processing and compute resources among a distributed computing environment to reduce infrastructure requirements at facility 3602. In at least one embodiment, deployment system 3606 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 3602. 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 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 3606 during execution of applications.

[0492] In at least one embodiment, some of applications used in advanced processing and 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 3602 using data 3608 (such as imaging data) generated at facility 3602 (and stored on one or more picture archiving and communication system (PACS) servers at facility 3602), may be trained using imaging or sequencing data 3608 from another facility or facilities (e.g., a different hospital, lab, clinic, etc.), or a combination thereof. In at least one embodiment, training system 3604 may be used to provide applications, services, and/or other resources for generating working, deployable machine learning models for deployment system 3606.

[0493] In at least one embodiment, a model registry 3624 may be backed by object storage that may support versioning and object metadata. In at least one embodiment, object storage may be accessible through, for example, a cloud storage (e.g., a cloud 3126 of FIG. 31) compatible application programming interface (API) from within a cloud platform. In at least one embodiment, machine learning models within model registry 3624 may be uploaded, listed, modified, or deleted by developers or partners of a system interacting with an API. In at least 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.

[0494] In at least one embodiment, a training pipeline 3704 (FIG. 37) may include a scenario where facility 3602 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 3608 generated by imaging device(s), sequencing devices, and/or other device types may be received. In at least one embodiment, once imaging data 3608 is received, AI-assisted annotation 3610 may be used to aid in generating annotations corresponding to imaging data 3608 to be used as ground truth data for a machine learning model. In at least one embodiment, AI-assisted annotation 3610 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 3608 (e.g., from certain devices) and/or certain types of anomalies in imaging data 3608. In at least one embodiment, AI-assisted annotations 3610 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 3612 (e.g., annotations provided by a clinician, doctor, scientist, technician, etc.) may be used as ground truth data for training a machine learning model. In at least one embodiment, AI-assisted annotations 3610, labeled clinic data 3612, 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 3616, and may be used by deployment system 3606, as described herein.

[0495] In at least one embodiment, training pipeline 3704 (FIG. 37) may include a scenario where facility 3602 needs a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3606, but facility 3602 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 machine learning model may be selected from model registry 3624. In at least one embodiment, model registry 3624 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 3624 may have been trained on imaging data from different facilities than facility 3602 (e.g., facilities remotely located). In at least one embodiment, 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 3624. 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 3624. In at least one embodiment, a machine learning model may then be selected from model registry 3624 – and referred to as output model 3616 – and may be used in deployment system 3606 to perform one or more processing tasks for one or more applications of a deployment system.

[0496] In at least one embodiment, training pipeline 3704 (FIG. 37) may be used in a scenario that includes facility 3602 requiring a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3606, but facility 3602 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 3624 might not be fine-tuned or optimized for imaging data 3608 generated at facility 3602 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 3610 may be used to aid in generating annotations corresponding to imaging data 3608 to be used as ground truth data for retraining or updating a machine learning model. In at least one embodiment, labeled data 3612 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 3614. In at least one embodiment, model training 3614 – e.g., AI-assisted annotations 3610, labeled data 3612, or a combination thereof – may be used as ground truth data for retraining or updating a machine learning model.

[0497] In at least one embodiment, deployment system 3606 may include software 3618, services 3620, hardware 3622, and/or other components, features, and functionality. In at least one embodiment, deployment system 3606 may include a software “stack,” such that software 3618 may be built on top of services 3620 and may use services 3620 to perform some or all of processing tasks, and services 3620 and software 3618 may be built on top of hardware 3622 and use hardware 3622 to execute processing, storage, and/or other compute tasks of deployment system 3606.

[0498] In at least one embodiment, software 3618 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 computing device there may be any number of containers that may perform a data processing task with respect to imaging data 3608 (or other data types, such as those described herein). 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 3608, in addition to containers that receive and configure imaging data for use by each container and/or for use by facility 3602 after processing through a pipeline (e.g., to convert outputs back to a usable data type for storage and display at facility 3602). In at least one embodiment, a combination of containers within software 3618 (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 3620 and hardware 3622 to execute some or all processing tasks of applications instantiated in containers.

[0499] In at least one embodiment, data may undergo pre-processing as 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 3616 of training system 3604.

[0500] 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 3624 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.

[0501] In at least one embodiment, developers may develop, publish, and store applications (e.g., as containers) for performing 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 3620 as a system (e.g., system 3700 of FIG. 37). In at least one embodiment, once validated by system 3700 (e.g., for accuracy, 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 one or more processing tasks with respect to data at a facility (e.g., a second facility) of a user.

[0502] 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 3700 of FIG. 37). 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 3624. In at least one embodiment, a requesting entity— who provides an inference or image processing request — may browse a container registry and/or model registry 3624 for an application, container, dataset, machine learning model, etc., select a desired combination of elements for inclusion in data processing pipeline, and submit an processing request. In at least one embodiment, a request may include input data 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 3606 (e.g., a cloud) to perform processing of data processing pipeline. In at least one embodiment, processing by deployment system 3606 may include referencing selected elements (e.g., applications, containers, models, etc.) from a container registry and/or model registry 3624. 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).

[0503] In at least one embodiment, to aid in processing or execution of applications or containers in pipelines, services 3620 may be leveraged. In at least one embodiment, services 3620 may include compute services, artificial intelligence (AI) services, visualization

services, and/or other service types. In at least one embodiment, services 3620 may provide functionality that is common to one or more applications in software 3618, 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 3620 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 3730 (FIG. 37)). In at least one embodiment, rather than each application that shares a same functionality offered by a service 3620 being required to have a respective instance of service 3620, service 3620 may be shared between 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.

[0504] In at least one embodiment, where a service 3620 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 3618 implementing advanced processing and inferencing pipeline may be streamlined because each application may call upon a same inference service to perform one or more inferencing tasks.

[0505] In at least one embodiment, hardware 3622 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 3622 may be used to provide efficient, purpose-built support for software 3618 and services 3620 in deployment system 3606. In at least one embodiment, use of GPU processing may be implemented for processing locally (e.g., at facility 3602), within an AI/deep learning system, in a cloud system, and/or in other processing components of deployment system 3606 to improve efficiency, accuracy, and efficacy of game name recognition.

[0506] In at least one embodiment, software 3618 and/or services 3620 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 3606 and/or training system 3604 may be 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, hardware 3622 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 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.

[0507] FIG. 37 is a system diagram for an example system 3700 for generating and deploying a deployment pipeline, in accordance with at least one embodiment. In at least one embodiment, system 3700 may be used to implement process 3600 of FIG. 36 and/or other processes including advanced processing and inferencing pipelines. In at least one embodiment, system 3700 may include training system 3604 and deployment system 3606. In at least one embodiment, training system 3604 and deployment system 3606 may be implemented using software 3618, services 3620, and/or hardware 3622, as described herein.

[0508] In at least one embodiment, system 3700 (e.g., training system 3604 and/or deployment system 3006) may implemented in a cloud computing environment (e.g., using cloud 3726). In at least one embodiment, system 3700 may be implemented locally with respect to a facility, or as a combination of both cloud and local computing resources. In at least one embodiment, access to APIs in cloud 3726 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 3700, may be restricted to a set of public IPs that have been vetted or authorized for interaction.

[0509] In at least one embodiment, various components of system 3700 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 3700 (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.

[0510] In at least one embodiment, training system 3604 may execute training pipelines 3704, similar to those described herein with respect to FIG. 36. In at least one embodiment, where one or more machine learning models are to be used in deployment pipelines 3710 by deployment system 3606, training pipelines 3704 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 3706 (e.g., without a need for retraining or updating). In at least one embodiment, as a result of training pipelines 3704, output model(s) 3616 may be generated. In at least one embodiment, training pipelines 3704 may include any number of processing steps³⁷, AI-assisted annotation 3610, labeling or annotating of imaging data 3608 to generate labeled data 3612, model selection from a model registry, model training 3614, training, retraining, or updating models, and/or other processing steps. In at least one embodiment, for different machine learning models used by deployment system 3606, different training pipelines 3704 may be used. In at least one embodiment, training pipeline 3704 similar to a first example described with respect to FIG. 36 may be used for a first machine learning model, training pipeline 3704 similar to a second example described with respect to FIG. 36 may be used for a second machine learning model, and training pipeline 3704 similar to a third example described with respect to FIG. 36 may be used for a third machine learning model. In at least one embodiment, any combination of tasks within training system 3604 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 3604, and may be implemented by deployment system 3606.

[0511] In at least one embodiment, output model(s) 3616 and/or pre-trained model(s) 3706 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 3700 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), Bi-LSTM, Hopfield, Boltzmann, deep belief, deconvolutional, generative adversarial, liquid state machine, etc.), and/or other types of machine learning models.

[0512] In at least one embodiment, training pipelines 3704 may include AI-assisted annotation. In at least one embodiment, labeled data 3612 (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 instance of imaging data 3608 (or other data type used by machine learning models), there may be corresponding ground truth data generated by training system 3604. In at least one embodiment, AI-assisted annotation may be performed as part of deployment pipelines 3710; either in addition to, or in lieu of AI-assisted annotation included in training pipelines 3704. In at least one embodiment, system 3700 may include a multi-layer platform that may include a software layer (e.g., software 3618) of diagnostic applications (or other application types) that may perform one or more medical imaging and diagnostic functions. .

[0513] 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 3602). In at least one embodiment, applications may then call or execute one or more services 3620 for performing compute, AI, or visualization tasks associated with respective applications, and software 3618 and/or services 3620 may leverage hardware 3622 to perform processing tasks in an effective and efficient manner.

[0514] In at least one embodiment, deployment system 3606 may execute deployment pipelines 3710. In at least one embodiment, deployment pipelines 3710 may include any number of applications that may be sequentially, non-sequentially, or otherwise applied to feedback data (and/or other data types) – including AI-assisted annotation, as described above. In at least one embodiment, as described herein, a deployment pipeline 3710 for an individual device may be referred to as a virtual instrument for a device. In at least one

embodiment, for a single device, there may be more than one deployment pipeline 3710 depending on information desired from data generated by a device.

[0515] In at least one embodiment, applications available for deployment pipelines 3710 may include any application that may be used for performing processing tasks on feedback data or other data from devices. . 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 3620) 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 3730 may be used for GPU acceleration of these processing tasks.

[0516] In at least one embodiment, deployment system 3606 may include a user interface 3714 (e.g., a graphical user interface, a web interface, etc.) that may be used to select applications for inclusion in deployment pipeline(s) 3710, arrange applications, modify or change applications or parameters or constructs thereof, use and interact with deployment pipeline(s) 3710 during set-up and/or deployment, and/or to otherwise interact with deployment system 3606. In at least one embodiment, although not illustrated with respect to training system 3604, user interface 3714 (or a different user interface) may be used for selecting models for use in deployment system 3606, for selecting models for training, or retraining, in training system 3604, and/or for otherwise interacting with training system 3604.

[0517] In at least one embodiment, pipeline manager 3712 may be used, in addition to an application orchestration system 3728, to manage interaction between applications or containers of deployment pipeline(s) 3710 and services 3620 and/or hardware 3622. In at least one embodiment, pipeline manager 3712 may be configured to facilitate interactions from application to application, from application to service 3620, and/or from application or service to hardware 3622. In at least one embodiment, although illustrated as included in software 3618, this is not intended to be limiting, and in some examples pipeline manager 3712 may be included in services 3620. In at least one embodiment, application orchestration system 3728 (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) 3710 (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.

[0518] In at least one embodiment, each application and/or container (or image thereof) 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 3712 and application orchestration system 3728. 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 3728 and/or pipeline manager 3712 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) 3710 may share same services and resources, application orchestration system 3728 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 3728) 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.

[0519] In at least one embodiment, services 3620 leveraged by and shared by applications or containers in deployment system 3606 may include compute services 3716, AI services 3718, visualization services 3720, and/or other service types. In at least one embodiment, applications may call (e.g., execute) one or more of services 3620 to perform processing operations for an application. In at least one embodiment, compute services 3716 may be leveraged by applications to perform super-computing or other high-performance computing (HPC) tasks. In at least one embodiment, compute service(s) 3716 may be leveraged to perform parallel processing (e.g., using a parallel computing platform 3730) for processing data through one or more of applications and/or one or more tasks of a single application,

substantially simultaneously. In at least one embodiment, parallel computing platform 3730 (e.g., NVIDIA's CUDA) may enable general purpose computing on GPUs (GPGPU) (e.g., GPUs 3722). In at least one embodiment, a software layer of parallel computing platform 3730 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 3730 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 3730 (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.

[0520] In at least one embodiment, AI services 3718 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 3718 may leverage AI system 3724 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) 3710 may use one or more of output models 3616 from training system 3604 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 3728 (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 during an emergency, or for a radiologist during diagnosis. In at least one embodiment, a 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 3728 may distribute resources (e.g., services 3620 and/or hardware 3622) based on priority paths for different inferencing tasks of AI services 3718.

[0521] In at least one embodiment, shared storage may be mounted to AI services 3718 within system 3700. 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 3606, 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 3624 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 3712) 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.

[0522] 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.

[0523] In at least one embodiment, during application execution, an inference request for a 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

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, 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 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.

[0524] In at least one embodiment, transfer of requests between services 3620 and inference applications may be hidden behind a software development kit (SDK), and robust 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 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 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 3726, and an inference service may perform inferencing on a GPU.

[0525] In at least one embodiment, visualization services 3720 may be leveraged to generate visualizations for viewing outputs of applications and/or deployment pipeline(s) 3710. In at least one embodiment, GPUs 3722 may be leveraged by visualization services 3720 to generate visualizations. In at least one embodiment, rendering effects, such as ray-tracing, may be implemented by visualization services 3720 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 3720 may include an internal visualizer, cinematics, and/or other rendering or image processing capabilities or functionality (e.g., ray tracing, rasterization, internal optics, etc.).

[0526] In at least one embodiment, hardware 3622 may include GPUs 3722, AI system 3724, cloud 3726, and/or any other hardware used for executing training system 3604 and/or deployment system 3606. In at least one embodiment, GPUs 3722 (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 3716, AI services 3718, visualization services 3720, other services, and/or any of features or functionality of software 3618. For example, with respect to AI services 3718, GPUs 3722 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 3726, AI system 3724, and/or other components of system 3700 may use GPUs 3722. In at least one embodiment, cloud 3726 may include a GPU-optimized platform for deep learning tasks. In at least one embodiment, AI system 3724 may use GPUs, and cloud 3726 – or at least a portion tasked with deep learning or inferencing – may be executed using one or more AI systems 3724. As such, although hardware 3622 is illustrated as discrete components, this is not intended to be limiting, and any components of hardware 3622 may be combined with, or leveraged by, any other components of hardware 3622.

[0527] In at least one embodiment, AI system 3724 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 system 3724 (e.g., NVIDIA's DGX) may include GPU-optimized software (e.g., a software stack) that may be executed using a plurality of GPUs 3722, in addition to CPUs, RAM, storage, and/or other components, features, or functionality. In at least one embodiment, one or more AI systems 3724 may be implemented in cloud 3726 (e.g., in a data center) for performing some or all of AI-based processing tasks of system 3700.

[0528] In at least one embodiment, cloud 3726 may include a GPU-accelerated infrastructure (e.g., NVIDIA's NGC) that may provide a GPU-optimized platform for executing processing tasks of system 3700. In at least one embodiment, cloud 3726 may

include an AI system(s) 3724 for performing one or more of AI-based tasks of system 3700 (e.g., as a hardware abstraction and scaling platform). In at least one embodiment, cloud 3726 may integrate with application orchestration system 3728 leveraging multiple GPUs to enable seamless scaling and load balancing between and among applications and services 3620. In at least one embodiment, cloud 3726 may tasked with executing at least some of services 3620 of system 3700, including compute services 3716, AI services 3718, and/or visualization services 3720, as described herein. In at least one embodiment, cloud 3726 may perform small and large batch inference (e.g., executing NVIDIA's TENSOR RT), provide an accelerated parallel computing API and platform 3730 (e.g., NVIDIA's CUDA), execute application orchestration system 3728 (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 3700.

[0529] 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 3726 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 3726 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.

[0530] FIG. 38 includes an example illustration of a deployment pipeline 3710A for processing imaging data, in accordance with at least one embodiment. In at least one embodiment, system 3700 – and specifically deployment system 3606 – may be used to customize, update, and/or integrate deployment pipeline(s) 3710A into one or more production environments. In at least one embodiment, deployment pipeline 3710A of FIG. 38 includes a non-limiting example of a deployment pipeline 3710A 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 3710A for a CT scanner 3802, 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 3802. In at least one embodiment, applications may be applied to deployment pipeline 3710A as containers that may leverage services 3620 and/or hardware 3622 of system 3700. In addition, deployment pipeline 3710A may include additional processing tasks or applications that may be implemented to prepare data for use by applications (e.g., DICOM adapter 3702B and DICOM reader 3806 may be used in deployment pipeline 3710A to prepare data for use by CT reconstruction 3808, organ segmentation 3810, etc.). In at least one embodiment, deployment pipeline 3710A 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 3808 and organ segmentation 3810 for several subjects over a specific interval, and thus may deploy pipeline 3710A for that period of time. In at least one embodiment, a user may select, for each request from system 3700, applications that a user wants to perform processing on that data for that request. In at least one embodiment, deployment pipeline 3710A may be adjusted at any interval and, because of adaptability and scalability of a container structure within system 3700, this may be a seamless process.

[0531] In at least one embodiment, deployment pipeline 3710A of FIG. 38 may include CT scanner 3802 generating imaging data of a patient or subject. In at least one embodiment, imaging data from CT scanner 3802 may be stored on a PACS server(s) 3804 associated with a facility housing CT scanner 3802. In at least one embodiment, PACS server(s) 3804 may include software and/or hardware components that may directly interface with imaging modalities (e.g., CT scanner 3802) at a facility. In at least one embodiment, DICOM adapter 3702B may enable sending and receipt of DICOM objects using DICOM protocols. In at least one embodiment, DICOM adapter 3702B may aid in preparation or configuration of DICOM data from PACS server(s) 3804 for use by deployment pipeline 3710A. In at least one embodiment, once DICOM data is processed through DICOM adapter 3702B, pipeline manager 3712 may route data through to deployment pipeline 3710A. In at least one embodiment, DICOM reader 3806 may extract image files and any associated metadata from DICOM data (e.g., raw sinogram data, as illustrated in visualization 3816A). 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 3710A. In at least one embodiment, once DICOM reader 3806 has finished extracting and/or storing data, a signal of completion may be communicated to pipeline manager 3712. In at least one embodiment, pipeline manager 3712 may then initiate or call upon one or more other applications or containers in deployment pipeline 3710A.

[0532] In at least one embodiment, CT reconstruction 3808 application and/or container may be executed once data (e.g., raw sinogram data) is available for processing by CT reconstruction 3808 application. In at least one embodiment, CT reconstruction 3808 may read raw sinogram data from a cache, reconstruct an image file out of raw sinogram data (e.g., as illustrated in visualization 3816B), and store resulting image file in a cache. In at least one embodiment, at completion of reconstruction, pipeline manager 3712 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 3810 application and/or container may be triggered by pipeline manager 3712. In at least one embodiment, organ segmentation 3810 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 3810 application and/or container may rely on services 3620, and pipeline manager 3712 and/or application orchestration system 3728 may facilitate use of services 3620 by organ segmentation 3810 application and/or container. In at least one embodiment, for example, organ segmentation 3810 application and/or container may leverage AI services 3718 to perform inference on a normalized image, and AI services 3718 may leverage hardware 3622 (e.g., AI system 3724) to execute AI services 3718. In at least one embodiment, a result of an inference may be a mask file (e.g., as illustrated in visualization 3816C) that may be stored in a cache (or other storage device).

[0533] In at least one embodiment, once applications that process DICOM data and/or data extracted from DICOM data have completed processing, a signal may be generated for pipeline manager 3712. In at least one embodiment, pipeline manager 3712 may then execute DICOM writer 3812 to read results from a cache (or other storage device), package results into a DICOM format (e.g., as DICOM output 3814) for use by users at a facility who generated a request. In at least one embodiment, DICOM output 3814 may then be transmitted to DICOM adapter 3702B to prepare DICOM output 3814 for storage on PACS server(s) 3804 (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 3816B and 3816C may be generated and available to a user for diagnoses, research, and/or for other purposes.

[0534] Although illustrated as consecutive application in deployment pipeline 3710A, CT reconstruction 3808 and organ segmentation 3810 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 3806 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 3620, a scheduler of system 3700 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 3730 may be used to perform parallel processing for applications to decrease run-time of deployment pipeline 3710A to provide real-time results.

[0535] In at least one embodiment, and with reference to FIGS. 39A-39B, deployment system 3606 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 3700 may allow for creation and provision of virtual instruments that may include a software-defined deployment pipeline 3710 that may receive raw/unprocessed input data generated by a device(s) and output processed/reconstructed data. In at least one embodiment, deployment pipelines 3710 (e.g., 3710A and 3710B) 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 embodiment, virtual instruments may execute any number of containers each including instantiations of applications. In at least one embodiment, such as where real-time processing is desired, deployment pipelines 3710 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).

[0536] In at least one embodiment, system 3700 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 3726). In at least one embodiment, deployment system 3606, 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 3604. 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.

[0537] In at least one embodiment, a computing system may include some or all of hardware 3622 described herein, and hardware 3622 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 3726. In at least one embodiment, because deployment system 3606 and associated applications or containers are 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.

[0538] FIG. 39A 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 3710B may leverage one or more of services 3620 of system 3700. In at least one embodiment, deployment pipeline 3710B and services 3620 may leverage hardware 3622 of a system either locally or in cloud 3726. In at least one embodiment, although not illustrated, process 3900 may be facilitated by pipeline manager 3712, application orchestration system 3728, and/or parallel computing platform 3730.

[0539] In at least one embodiment, process 3900 may include receipt of imaging data from an ultrasound device 3902. 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 3700 for processing through deployment pipeline 3710 selected or customized as a virtual instrument (e.g., a virtual ultrasound) for ultrasound device 3902. In at least one embodiment, imaging data may be received directly from an imaging device (e.g., ultrasound device 3902) 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 3806 to extract data for use by applications or containers of deployment pipeline 3710B. In at least one embodiment, DICOM reader 3806 may leverage data augmentation library 3914 (e.g., NVIDIA's DALI) as a service 3620 (e.g., as one of compute service(s) 3716) for extracting, resizing, rescaling, and/or otherwise preparing data for use by applications or containers.

[0540] In at least one embodiment, once data is prepared, a reconstruction 3906 application and/or container may be executed to reconstruct data from ultrasound device 3902 into an image file. In at least one embodiment, after reconstruction 3906, or at a same time as reconstruction 3906, a detection 3908 application and/or container may be executed for anomaly detection, object detection, feature detection, and/or other detection tasks related to data. In at least one embodiment, an image file generated during reconstruction 3906 may be used during detection 3908 to identify anomalies, objects, features, etc. In at least one embodiment, detection 3908 application may leverage an inference engine 3916 (e.g., as one of AI service(s) 3718) to perform inference on data to generate detections. In at least one embodiment, one or more machine learning models (e.g., from training system 3604) may be executed or called by detection 3908 application.

[0541] In at least one embodiment, once reconstruction 3906 and/or detection 3908 is/are complete, data output from these application and/or containers may be used to generate visualizations 3910, such as visualization 3912 (e.g., a grayscale output) displayed on a workstation or display terminal. In at least one embodiment, visualization may allow a technician or other user to visualize results of deployment pipeline 3710B with respect to ultrasound device 3902. In at least one embodiment, visualization 3910 may be executed by leveraging a render component 3918 of system 3700 (e.g., one of visualization service(s)

3720). In at least one embodiment, render component 3918 may execute a 2D, OpenGL, or ray-tracing service to generate visualization 3912.

[0542] FIG. 39B 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 3710C may leverage one or more of services 3620 of system 3700. In at least one embodiment, deployment pipeline 3710C and services 3620 may leverage hardware 3622 of a system either locally or in cloud 3726. In at least one embodiment, although not illustrated, process 3980 may be facilitated by pipeline manager 3712, application orchestration system 3728, and/or parallel computing platform 3730.

[0543] In at least one embodiment, process 3920 may include CT scanner 3922 generating raw data that may be received by DICOM reader 3806 (e.g., directly, via a PACS server 3804, after processing, etc.). In at least one embodiment, a Virtual CT (instantiated by deployment pipeline 3710C) may include a first, real-time pipeline for monitoring a patient (e.g., patient movement detection AI 3926) and/or for adjusting or optimizing exposure of CT scanner 3922 (e.g., using exposure control AI 3924). In at least one embodiment, one or more of applications (e.g., 3924 and 3926) may leverage a service 3620, such as AI service(s) 3718. In at least one embodiment, outputs of exposure control AI 3924 application (or container) and/or patient movement detection AI 3926 application (or container) may be used as feedback to CT scanner 3922 and/or a technician for adjusting exposure (or other settings of CT scanner 3922) and/or informing a patient to move less.

[0544] In at least one embodiment, deployment pipeline 3710C may include a non-real-time pipeline for analyzing data generated by CT scanner 3922. In at least one embodiment, a second pipeline may include CT reconstruction 3808 application and/or container, a coarse detection AI 3928 application and/or container, a fine detection AI 3932 application and/or container (e.g., where certain results are detected by coarse detection AI 3928), a visualization 3930 application and/or container, and a DICOM writer 3812 (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 3922 may be passed through pipelines of deployment pipeline 3710C (instantiated as a virtual CT instrument) to generate results. In at least one embodiment, results from DICOM writer 3812 may be transmitted for display and/or may be stored on PACS server(s) 3804 for later retrieval, analysis, or display by a technician, practitioner, or other user.

[0545] At least one embodiment of the disclosure can be described in view of the following clauses:

[0546] In clause 1, a processor comprising: one or more circuits to use one or more neural networks to generate a dataset including labels of one or more objects within one or more images generated by one or more generative networks.

[0547] In clause 2, the processor of claim 1, wherein the labels of the one or more objects comprise pixel-level labels of the one or more objects within the one or more images.

[0548] In clause 3, the processor of claim 1, wherein the one or more generative networks comprise one or more generative adversarial networks (GANs), wherein the one or more images comprise one or more synthetic images generated by a GAN of the one or more GANs, wherein each synthetic image of the one or more synthetic images is generated using a combination of data output by a plurality of layers of the GAN.

[0549] In clause 4, the processor of claim 3, wherein the data output by the plurality of layers of the GAN comprises a plurality of feature maps associated with the respective synthetic image, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

[0550] In clause 5, the processor of claim 3, wherein the one or more circuits are further to: use the GAN to generate the one or more synthetic images, wherein for each synthetic image of the one or more synthetic images the data output by the plurality of layers of the GAN is produced in generation of the synthetic image, and wherein the dataset is usable to train an additional machine learning model to perform pixel-level segmentation of images.

[0551] In clause 6, the processor of claim 1, wherein to generate a synthetic image of the one or more images in the dataset the one or more circuits are further to: extract a plurality of intermediate feature maps from intermediate layers of a generative network of the one or more generative networks, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image; for each intermediate feature map of the plurality of intermediate feature maps, resize the respective intermediate feature map to a same certain resolution; and concatenate data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

[0552] In clause 7, the processor of claim 6, wherein the one or more circuits are further to: input the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map; determine, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and generate a mask for the synthetic image, wherein each

entry in the mask is associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

[0553] In clause 8, a processor comprising: one or more circuits to train one or more neural networks to perform pixel-level labeling of synthetic images generated by a generative network, wherein the one or more neural networks are trained using a plurality of synthetic images generated by the generative network, each of the plurality of synthetic images comprising a) pixel-level labels and b) a combination of data output by a plurality of layers of the generative network.

[0554] In clause 9, the processor of claim 8, wherein the plurality of synthetic images comprises 50 or fewer images.

[0555] In clause 10, the processor of claim 8, wherein the combination of data output by the plurality of layers of the generative network comprises a plurality of feature maps associated with each of the plurality of synthetic images, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

[0556] In clause 11, the processor of claim 8, wherein the one or more circuits are further to: use the generative network to generate each of the plurality of synthetic images, wherein the data output by the plurality of layers of the generative network is produced in generation of the respective synthetic image.

[0557] In clause 12, the processor of claim 8, wherein the trained one or more neural networks is trained to perform operations comprising: extracting a plurality of intermediate feature maps from the plurality of layers of the generative network, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image; for each intermediate feature map of the plurality of intermediate feature maps, resizing the respective intermediate feature map to a same certain resolution; and concatenating data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

[0558] In clause 13, the processor of claim 12, wherein the trained one or more neural networks is trained to perform further operations comprising: inputting the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map; determining, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and generating a mask for the synthetic image, wherein each entry in the mask is

associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

[0559] In clause 14, a method comprising: generating, using one or more neural networks, a dataset including labels of one or more objects within one or more images generated by one or more generative networks.

[0560] In clause 15, the method of claim 14, wherein the one or more images comprise one or more synthetic images generated by the generative network, wherein each synthetic image of the one or more synthetic images is generated using a combination of data output by a plurality of layers of the generative network.

[0561] In clause 16, the method of claim 15, wherein the data output by the plurality of layers of the generative network comprises a plurality of feature maps associated with the respective synthetic image, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

[0562] In clause 17, the method of claim 16, further comprising: using the generative network to generate the one or more synthetic images, wherein for each synthetic image of the one or more synthetic images the data output by the plurality of layers of the generative network is produced in generation of the synthetic image, and wherein the dataset is usable to train an additional machine learning model to perform pixel-level segmentation of images.

[0563] In clause 18, the method of claim 14, wherein generating a synthetic image of the one or more images in the dataset further comprises: extracting a plurality of intermediate feature maps from intermediate layers of the generative network, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image; for each intermediate feature map of the plurality of intermediate feature maps, resizing the respective intermediate feature map to a same certain resolution; and concatenating data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

[0564] In clause 19, the method of claim 18, further comprising: inputting the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map; determining, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and generating a mask for the synthetic image, wherein each entry in the mask is associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

[0565] In clause 20, a system comprising: one or more processors to use one or more neural networks to generate a dataset including labels of one or more objects within one or more images generated by one or more generative networks; and one or more memories to store parameters associated with the one or more neural networks.

[0566] In clause 21, the system of claim 20, wherein the labels of the one or more objects comprise pixel-level labels of the one or more objects within the one or more images.

[0567] In clause 22, the system of claim 20, wherein the one or more generative networks comprise one or more generative adversarial networks (GANs), wherein the one or more images comprise one or more synthetic images generated by a GAN of the one or more GANs, wherein each synthetic image of the one or more synthetic images is generated using a combination of data output by a plurality of layers of the GAN.

[0568] In clause 23, the system of claim 22, wherein the data output by the plurality of layers of the GAN comprises a plurality of feature maps associated with the respective synthetic image, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

[0569] In clause 24, the system of claim 22, wherein the one or more processors are further to: use the GAN to generate the one or more synthetic images, wherein for each synthetic image of the one or more synthetic images the data output by the plurality of layers of the GAN is produced in generation of the synthetic image, and wherein the dataset is usable to train an additional machine learning model to perform pixel-level segmentation of images.

[0570] In clause 25, the system of claim 20, wherein to generate a synthetic image of the one or more images in the dataset the one or more processors are further to: extract a plurality of intermediate feature maps from intermediate layers of the generative network, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image; for each intermediate feature map of the plurality of intermediate feature maps, resize the respective intermediate feature map to a same certain resolution; and concatenate data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

[0571] In clause 26, the system of claim 20, wherein the one or more processors are further to: input the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map; determine, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and generate a mask for the synthetic image, wherein each

entry in the mask is associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

[0572] In clause 27, a method comprising: generating, using one or more neural networks, a dataset including labels of one or more parts of an automobile within one or more automobile images generated by one or more generative networks, wherein each automobile image of the one or more automobile images is generated using a combination of data output by a plurality of layers of the generative network.

[0573] In clause 28, the method of claim 27, wherein the one or more generative networks comprise one or more generative adversarial networks (GANs), and wherein generating an automobile image of the one or more automobile images in the dataset further comprises: extracting a plurality of intermediate feature maps from intermediate layers of a GAN of the one or more GANs, the plurality of intermediate feature maps generated by the GAN during generation of the automobile image; for each intermediate feature map of the plurality of intermediate feature maps, resizing the respective intermediate feature map to a same certain resolution; and concatenating data from the plurality of intermediate feature maps and the automobile image to generate a combined automobile feature map, the combined feature map having a plurality of channels.

[0574] In clause 29, the method of claim 28 further comprises: inputting the combined automobile feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined automobile feature map; determining, for each pixel in the combined automobile feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications corresponding to automobile parts of the combined automobile feature map; and generating a mask for the automobile image, wherein each entry in the mask is associated with a specific pixel in the automobile image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

[0575] In clause 30, the method of claim 29 further comprises: using the dataset to train an additional machine learning model to perform pixel-level segmentation of automobile images.

[0576] In clause 31, a system comprising: one or more processors to train one or more neural networks to perform pixel-level labeling of synthetic images generated by a generative network, wherein the one or more neural networks are trained using a plurality of synthetic images generated by the generative network, each of the plurality of synthetic images

comprising a) pixel-level labels and b) a combination of data output by a plurality of layers of the generative network; and one or more memories to store parameters associated with the one or more neural networks.

[0577] 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.

[0578] In at least one embodiment, referring back to FIG. 13, computer programs in form of machine-readable executable code or computer control logic algorithms are stored in main memory 1304 and/or secondary storage. Computer programs, if executed by one or more processors, enable system 1300 to perform various functions in accordance with at least one embodiment. In at least one embodiment, memory 1304, 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 (“USB”) flash memory, etc. In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of CPU 1302, parallel processing system 1312, an integrated circuit capable of at least a portion of capabilities of both CPU 1302, parallel processing system 1312, 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 combination of integrated circuit(s).

[0579] 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 1300 may take form of a desktop 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.

[0580] In at least one embodiment, parallel processing system 1312 includes, without limitation, a plurality of parallel processing units (“PPUs”) 1314 and associated memories

1316. In at least one embodiment, PPUs 1314 are connected to a host processor or other peripheral devices via an interconnect 1318 and a switch 1320 or multiplexer. In at least one embodiment, parallel processing system 712 distributes computational tasks across PPUs 1314 which can be parallelizable — for example, as part of distribution of computational 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 1314, although such shared memory may incur performance penalties relative to use of local memory and registers resident to a PPU 1314. In at least one embodiment, operation of PPUs 1314 is synchronized through use of a command such as `__syncthreads()`, wherein all threads in a block (e.g., executed across multiple PPUs 1314) to reach a certain point of execution of code before proceeding.

[0581] 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.

[0582] 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.

[0583] 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 that an item, term, etc., may be either A or B or C, or any nonempty subset of set of A and B 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.”

[0584] 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 embodiment, executable instructions are executed such that different instructions are 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.

[0585] 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.

[0586] 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.

[0587] 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.

[0588] 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.

[0589] 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

computing device, that manipulate and/or transform data represented as physical, such as 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.

[0590] 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, 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.

[0591] 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 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, 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.

[0592] 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 may be defined above for purposes of description, various functions and

responsibilities might be distributed and divided in different ways, depending on circumstances.

[0593] 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 processor comprising:
one or more circuits to use one or more neural networks to generate a dataset including labels of one or more objects within one or more images generated by one or more generative networks.
2. The processor of claim 1, wherein the labels of the one or more objects comprise pixel-level labels of the one or more objects within the one or more images.
3. The processor of claim 1, wherein the one or more generative networks comprise one or more generative adversarial networks (GANs), wherein the one or more images comprise one or more synthetic images generated by a GAN of the one or more GANs, wherein each synthetic image of the one or more synthetic images is generated using a combination of data output by a plurality of layers of the GAN.
4. The processor of claim 3, wherein the data output by the plurality of layers of the GAN comprises a plurality of feature maps associated with the respective synthetic image, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.
5. The processor of claim 3, wherein the one or more circuits are further to:
use the GAN to generate the one or more synthetic images, wherein for each synthetic image of the one or more synthetic images the data output by the plurality of layers of the GAN is produced in generation of the synthetic image, and wherein the dataset is usable to train an additional machine learning model to perform pixel-level segmentation of images.
6. The processor of claim 1, wherein to generate a synthetic image of the one or more images in the dataset the one or more circuits are further to:
extract a plurality of intermediate feature maps from intermediate layers of a generative network of the one or more generative networks, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image;

for each intermediate feature map of the plurality of intermediate feature maps, resize the respective intermediate feature map to a same certain resolution; and

concatenate data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

7. The processor of claim 6, wherein the one or more circuits are further to:

input the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map;

determine, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and

generate a mask for the synthetic image, wherein each entry in the mask is associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

8. A processor comprising:

one or more circuits to train one or more neural networks to perform pixel-level labeling of synthetic images generated by a generative network, wherein the one or more neural networks are trained using a plurality of synthetic images generated by the generative network, each of the plurality of synthetic images comprising a) pixel-level labels and b) a combination of data output by a plurality of layers of the generative network.

9. The processor of claim 8, wherein the plurality of synthetic images comprises 50 or fewer images.

10. The processor of claim 8, wherein the combination of data output by the plurality of layers of the generative network comprises a plurality of feature maps associated with each of the plurality of synthetic images, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

11. The processor of claim 8, wherein the one or more circuits are further to:
use the generative network to generate each of the plurality of synthetic images,
wherein the data output by the plurality of layers of the generative network is produced in
generation of the respective synthetic image.
12. The processor of claim 8, wherein the trained one or more neural networks is trained
to perform operations comprising:
extracting a plurality of intermediate feature maps from the plurality of layers of the
generative network, the plurality of intermediate feature maps generated by the generative
network during generation of the synthetic image;
for each intermediate feature map of the plurality of intermediate feature maps,
resizing the respective intermediate feature map to a same certain resolution; and
concatenating data from the plurality of intermediate feature maps and the synthetic
image to generate a combined feature map, the combined feature map having a plurality of
channels.
13. The processor of claim 12, wherein the trained one or more neural networks is trained
to perform further operations comprising:
inputting the combined feature map to a trained pixel-level classifier to perform pixel-
level classification of pixels of the combined feature map;
determining, for each pixel in the combined feature map and using the plurality of
channels, a classification associated with the respective pixel, wherein the classification is
one of a plurality of classifications associated with the combined feature map; and
generating a mask for the synthetic image, wherein each entry in the mask is
associated with a specific pixel in the synthetic image and indicates for the specific pixel an
association between the specific pixel and a classification of the plurality of classifications.
14. A method comprising:
generating, using one or more neural networks, a dataset including labels of one or
more objects within one or more images generated by one or more generative networks.
15. The method of claim 14, wherein the one or more images comprise one or more
synthetic images generated by the generative network, wherein each synthetic image of the

one or more synthetic images is generated using a combination of data output by a plurality of layers of the generative network.

16. The method of claim 15, wherein the data output by the plurality of layers of the generative network comprises a plurality of feature maps associated with the respective synthetic image, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

17. The method of claim 16, further comprising:
using the generative network to generate the one or more synthetic images, wherein for each synthetic image of the one or more synthetic images the data output by the plurality of layers of the generative network is produced in generation of the synthetic image, and wherein the dataset is usable to train an additional machine learning model to perform pixel-level segmentation of images.

18. The method of claim 14, wherein generating a synthetic image of the one or more images in the dataset further comprises:

extracting a plurality of intermediate feature maps from intermediate layers of the generative network, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image;

for each intermediate feature map of the plurality of intermediate feature maps, resizing the respective intermediate feature map to a same certain resolution; and

concatenating data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

19. The method of claim 18, further comprising:

inputting the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map;

determining, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and

generating a mask for the synthetic image, wherein each entry in the mask is associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

20. A system comprising:

one or more processors to use one or more neural networks to generate a dataset including labels of one or more objects within one or more images generated by one or more generative networks; and

one or more memories to store parameters associated with the one or more neural networks.

21. The system of claim 20, wherein the labels of the one or more objects comprise pixel-level labels of the one or more objects within the one or more images.

22. The system of claim 20, wherein the one or more generative networks comprise one or more generative adversarial networks (GANs), wherein the one or more images comprise one or more synthetic images generated by a GAN of the one or more GANs, wherein each synthetic image of the one or more synthetic images is generated using a combination of data output by a plurality of layers of the GAN.

23. The system of claim 22, wherein the data output by the plurality of layers of the GAN comprises a plurality of feature maps associated with the respective synthetic image, wherein each of the plurality of feature maps is output by a different layer of the plurality of layers.

24. The system of claim 22, wherein the one or more processors are further to:

use the GAN to generate the one or more synthetic images, wherein for each synthetic image of the one or more synthetic images the data output by the plurality of layers of the GAN is produced in generation of the synthetic image, and wherein the dataset is usable to train an additional machine learning model to perform pixel-level segmentation of images.

25. The system of claim 20, wherein to generate a synthetic image of the one or more images in the dataset the one or more processors are further to:

extract a plurality of intermediate feature maps from intermediate layers of the generative network, the plurality of intermediate feature maps generated by the generative network during generation of the synthetic image;

for each intermediate feature map of the plurality of intermediate feature maps, resize the respective intermediate feature map to a same certain resolution; and

concatenate data from the plurality of intermediate feature maps and the synthetic image to generate a combined feature map, the combined feature map having a plurality of channels.

26. The system of claim 20, wherein the one or more processors are further to:

input the combined feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined feature map;

determine, for each pixel in the combined feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications associated with the combined feature map; and

generate a mask for the synthetic image, wherein each entry in the mask is associated with a specific pixel in the synthetic image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

27. A method comprising:

generating, using one or more neural networks, a dataset including labels of one or more parts of an automobile within one or more automobile images generated by one or more generative networks, wherein each automobile image of the one or more automobile images is generated using a combination of data output by a plurality of layers of the generative network.

28. The method of claim 27, wherein the one or more generative networks comprise one or more generative adversarial networks (GANs), and wherein generating an automobile image of the one or more automobile images in the dataset further comprises:

extracting a plurality of intermediate feature maps from intermediate layers of a GAN of the one or more GANs, the plurality of intermediate feature maps generated by the GAN during generation of the automobile image;

for each intermediate feature map of the plurality of intermediate feature maps, resizing the respective intermediate feature map to a same certain resolution; and

concatenating data from the plurality of intermediate feature maps and the automobile image to generate a combined automobile feature map, the combined feature map having a plurality of channels.

29. The method of claim 28 further comprises:

inputting the combined automobile feature map to a trained pixel-level classifier to perform pixel-level classification of pixels of the combined automobile feature map;

determining, for each pixel in the combined automobile feature map and using the plurality of channels, a classification associated with the respective pixel, wherein the classification is one of a plurality of classifications corresponding to automobile parts of the combined automobile feature map; and

generating a mask for the automobile image, wherein each entry in the mask is associated with a specific pixel in the automobile image and indicates for the specific pixel an association between the specific pixel and a classification of the plurality of classifications.

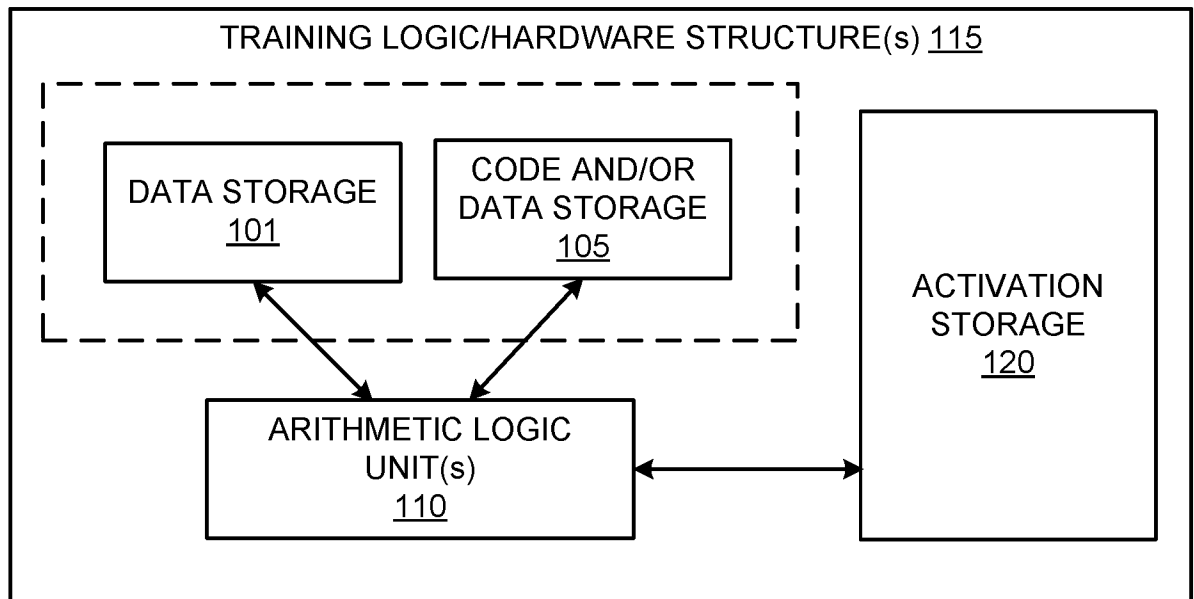
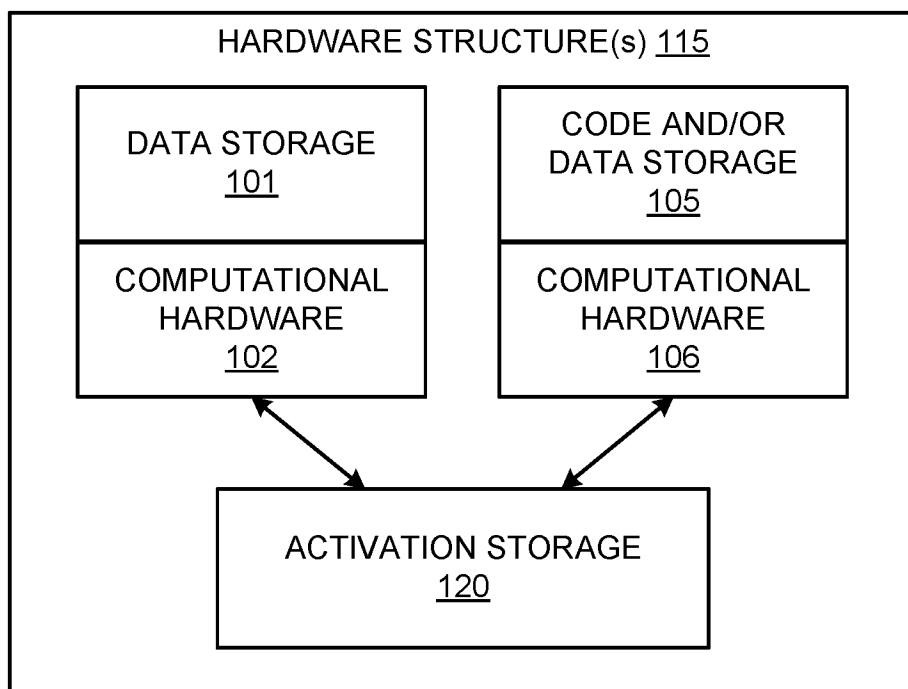
30. The method of claim 29 further comprises:

using the dataset to train an additional machine learning model to perform pixel-level segmentation of automobile images.

31. A system comprising:

one or more processors to train one or more neural networks to perform pixel-level labeling of synthetic images generated by a generative network, wherein the one or more neural networks are trained using a plurality of synthetic images generated by the generative network, each of the plurality of synthetic images comprising a) pixel-level labels and b) a combination of data output by a plurality of layers of the generative network; and

one or more memories to store parameters associated with the one or more neural networks.

**FIG. 1A****FIG. 1B**

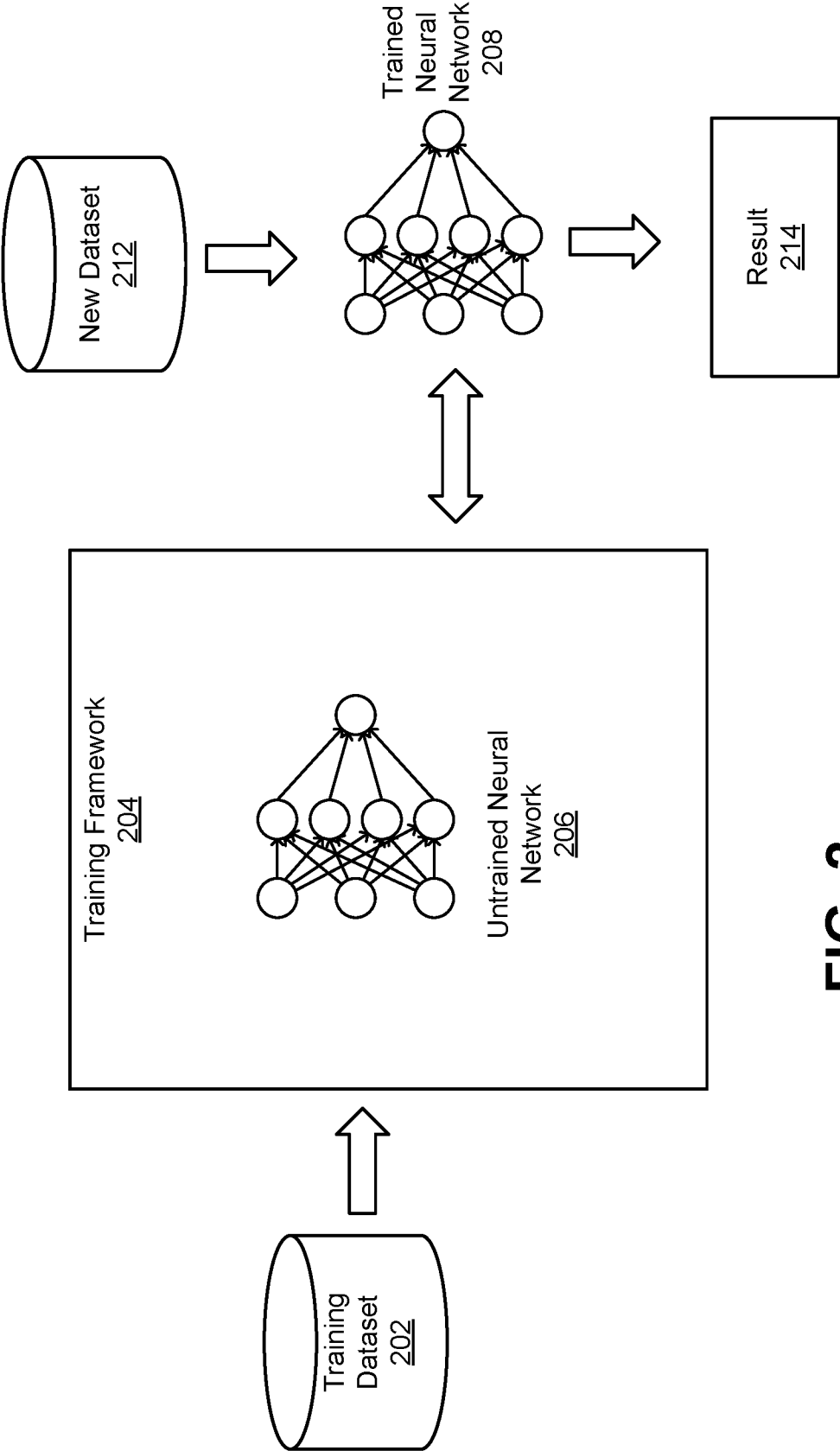
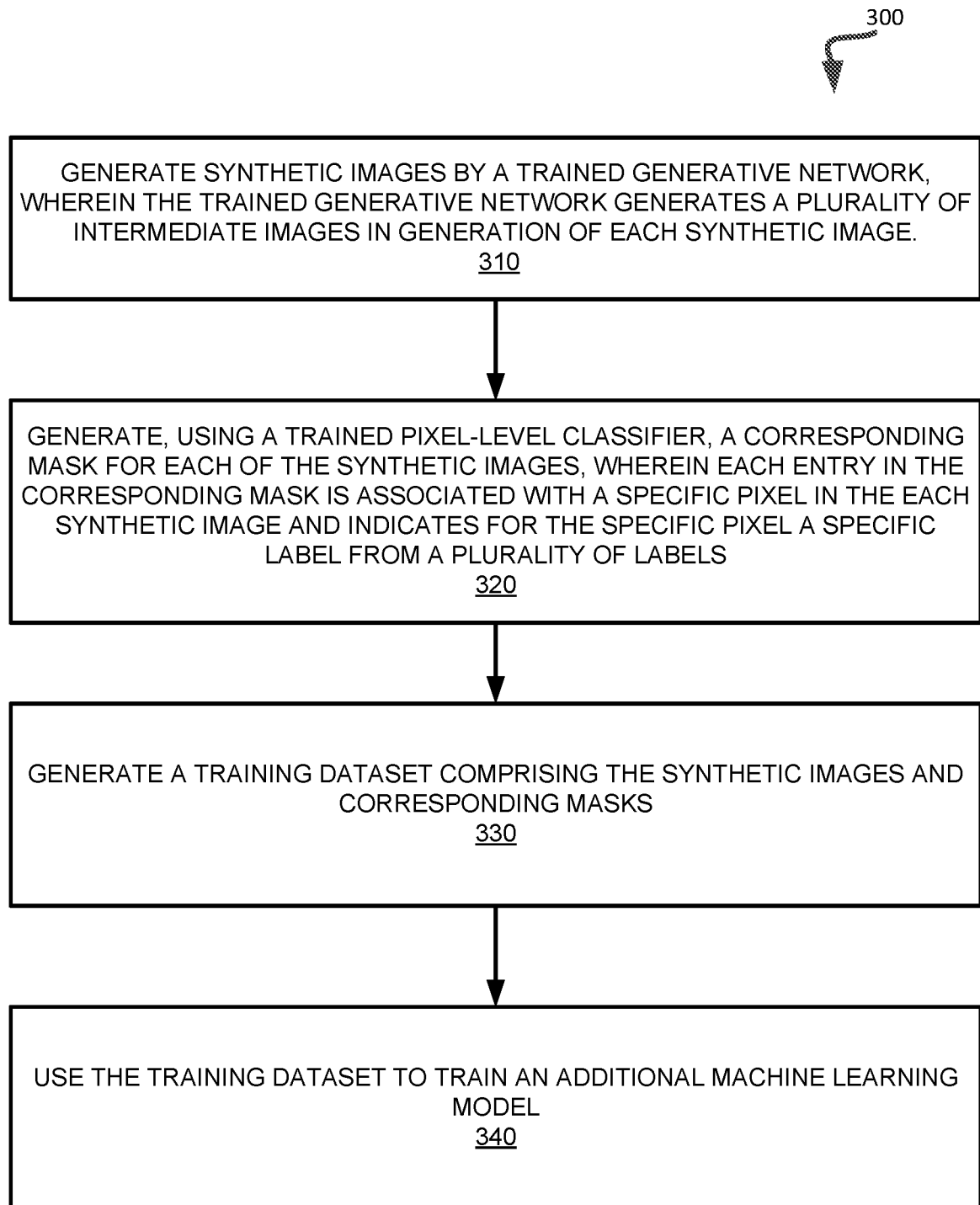
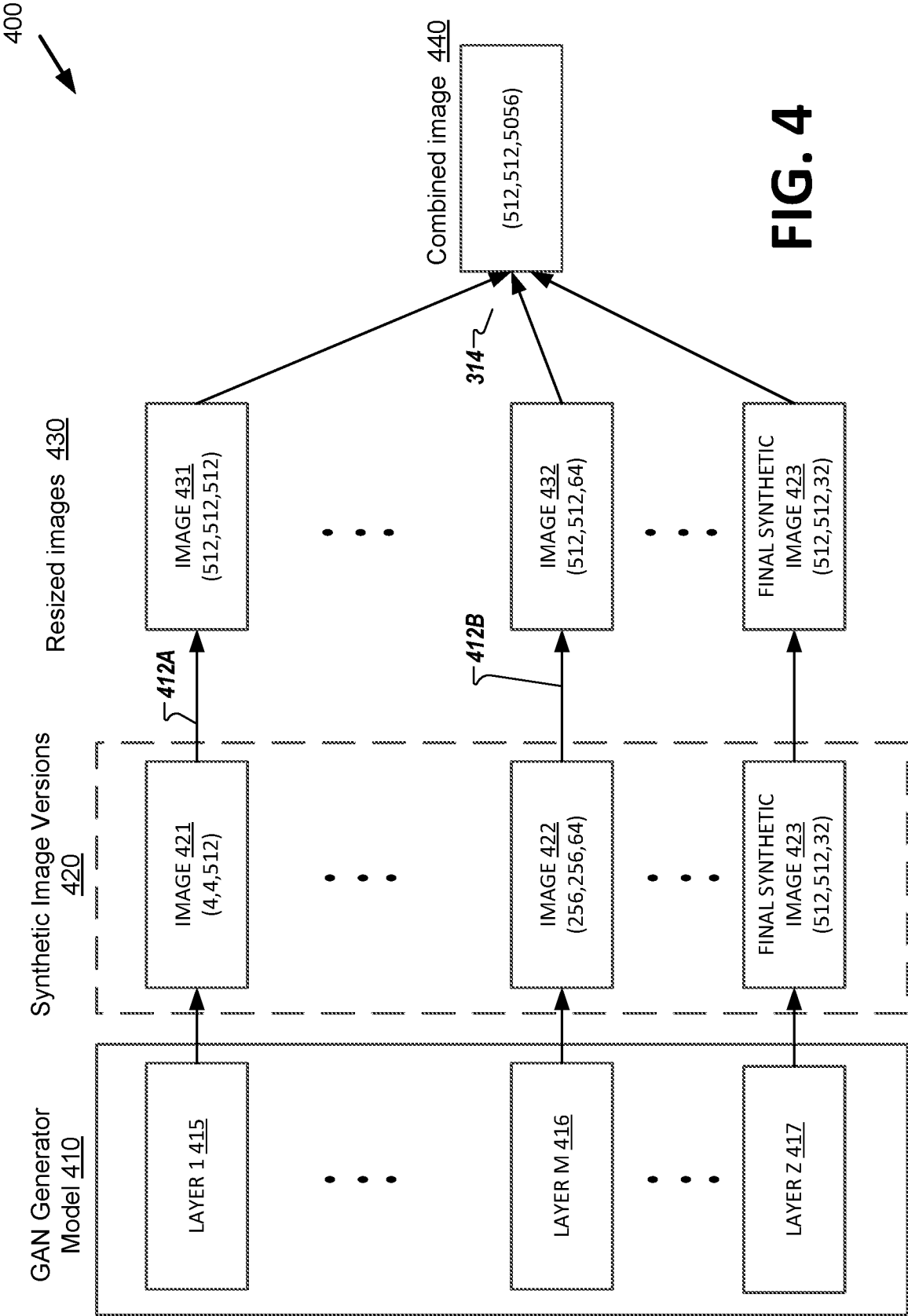
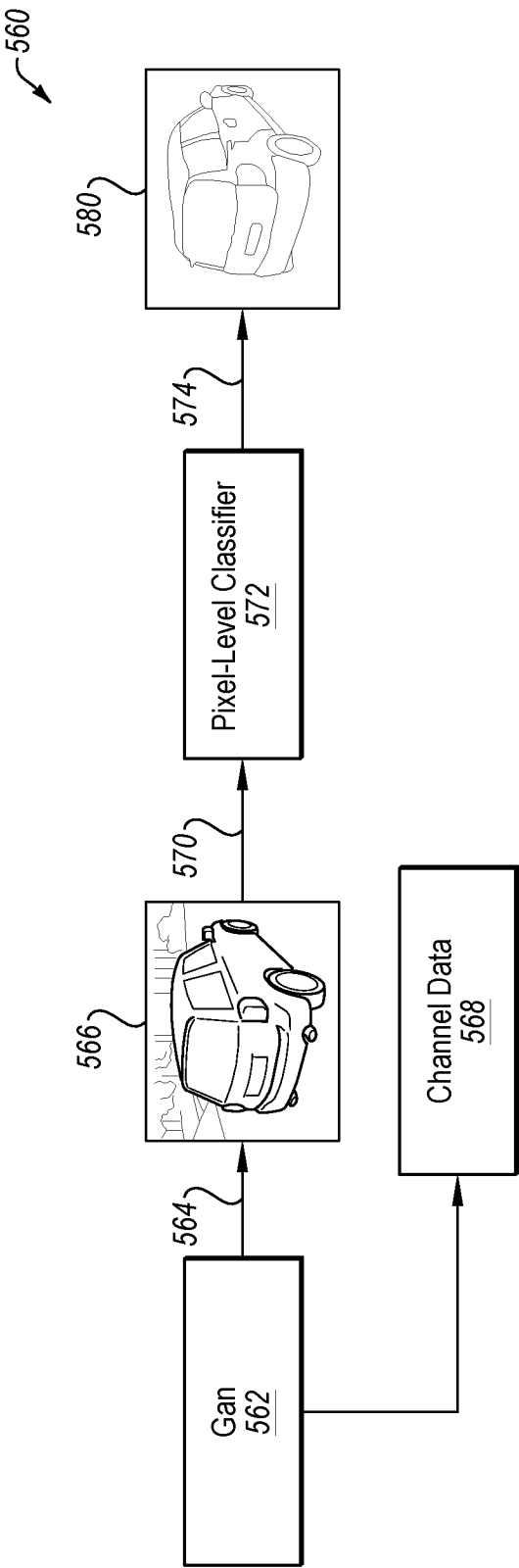
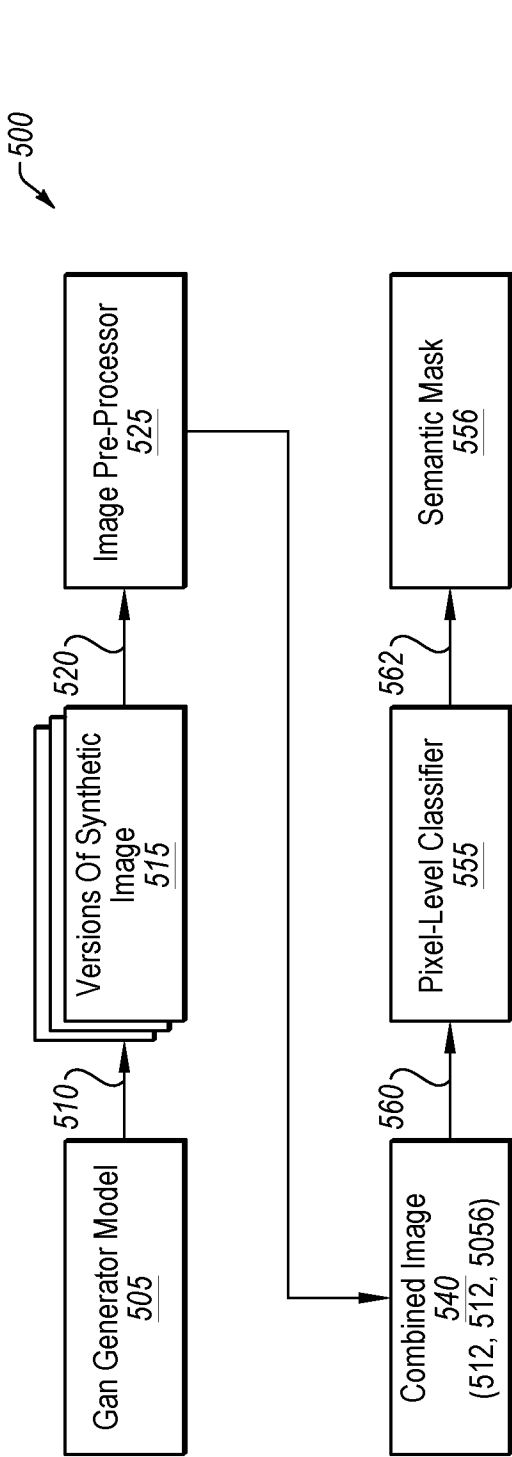


FIG. 2

**FIG. 3**





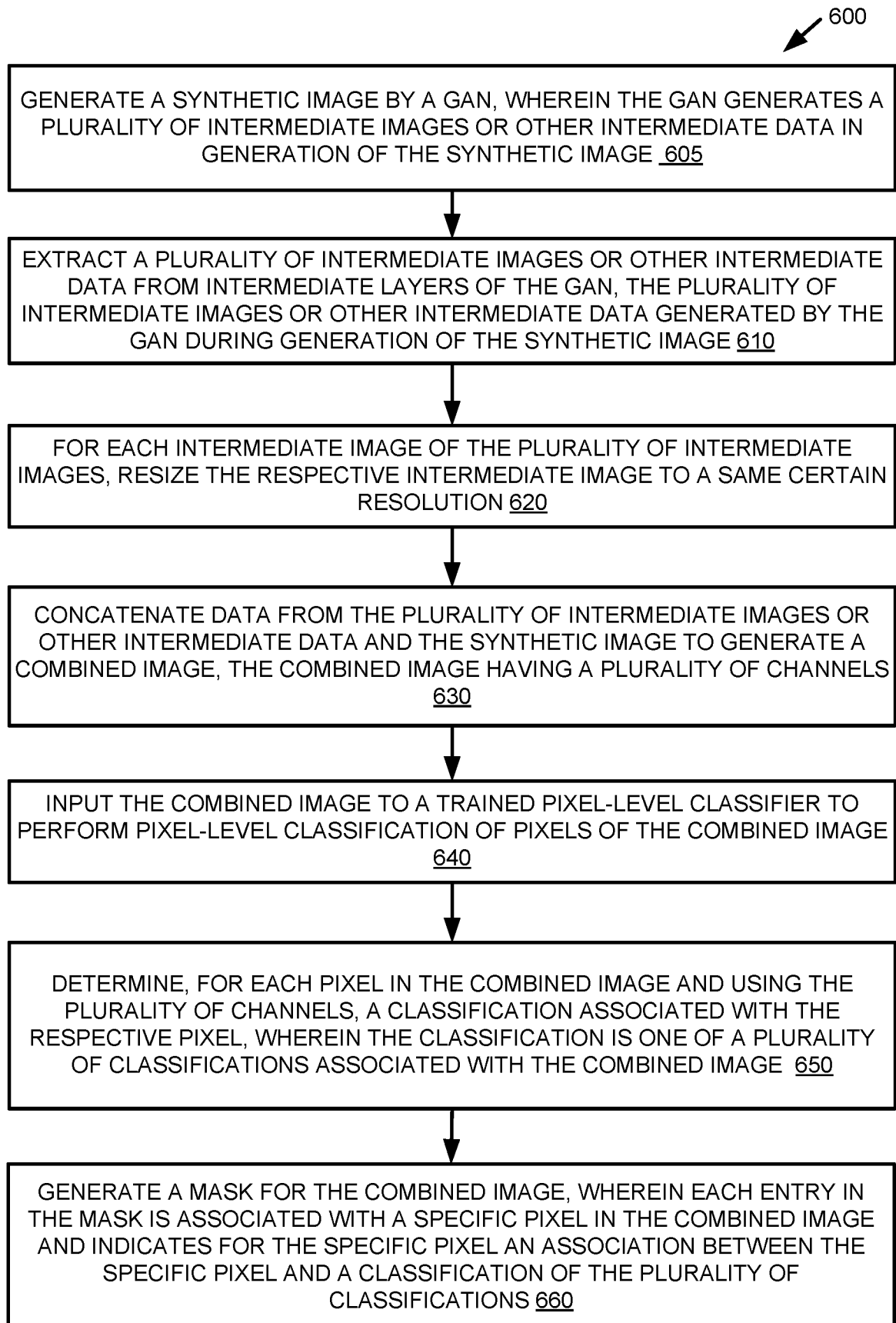
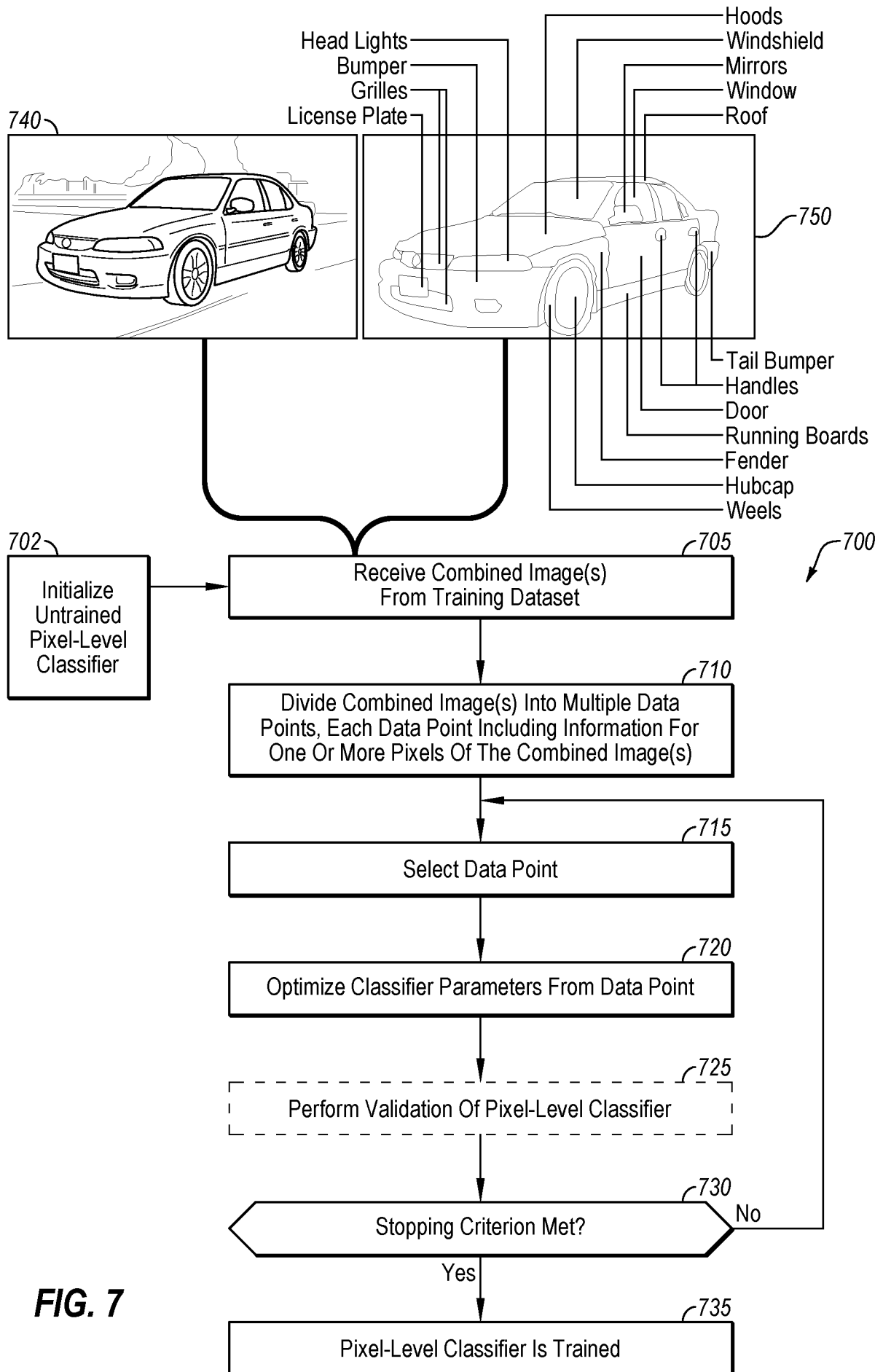
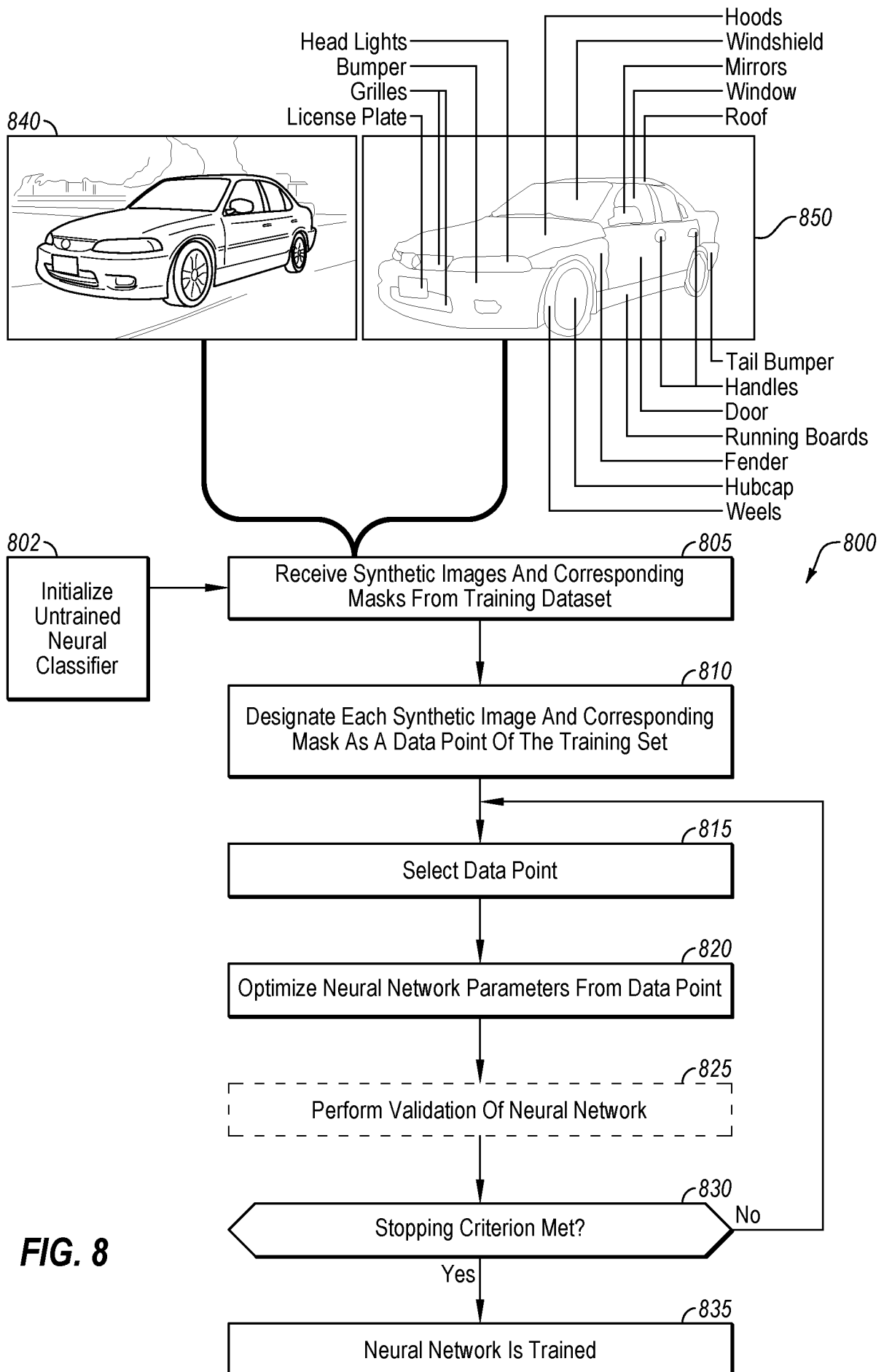


FIG. 6

**FIG. 7**



DATA CENTER
900

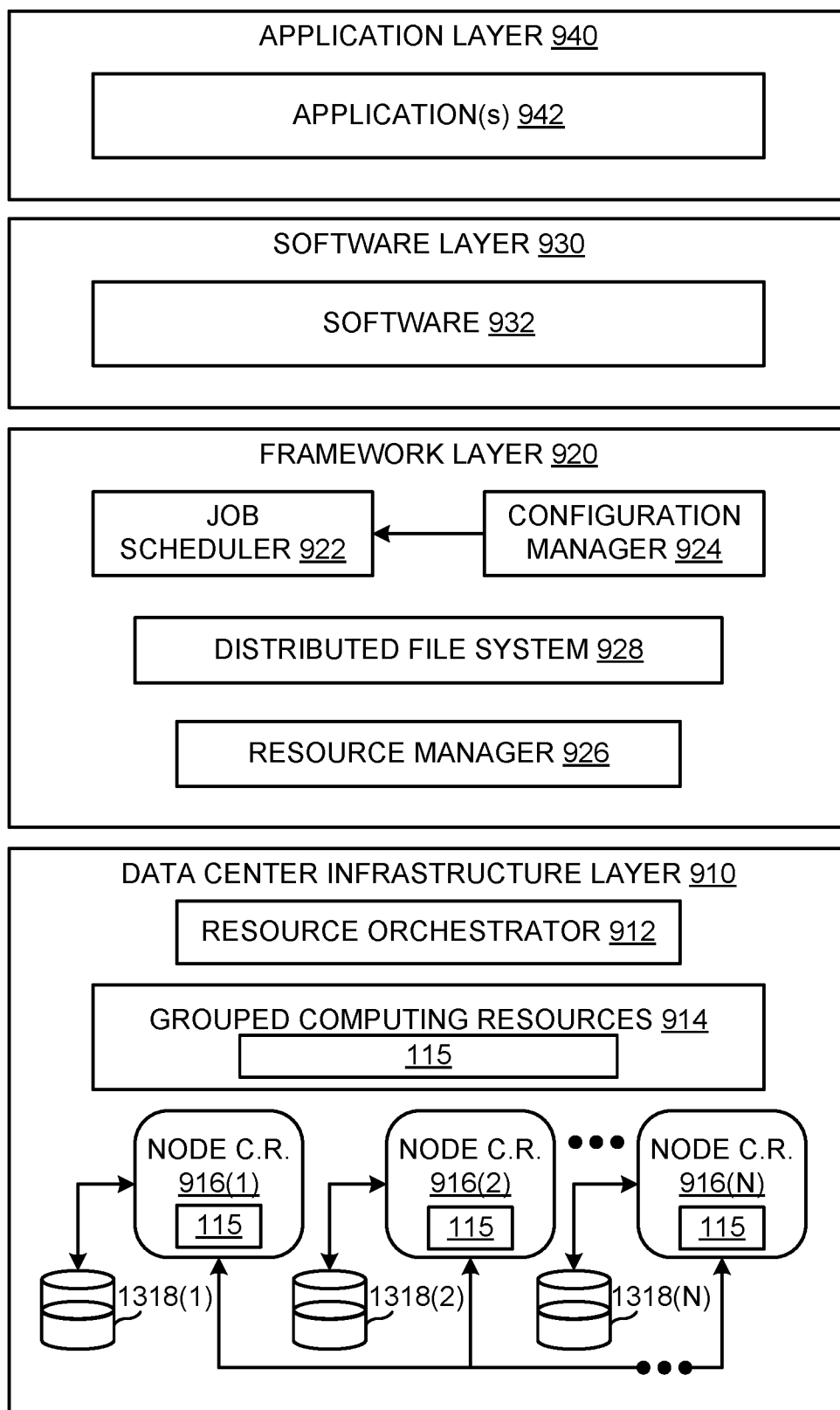


FIG. 9

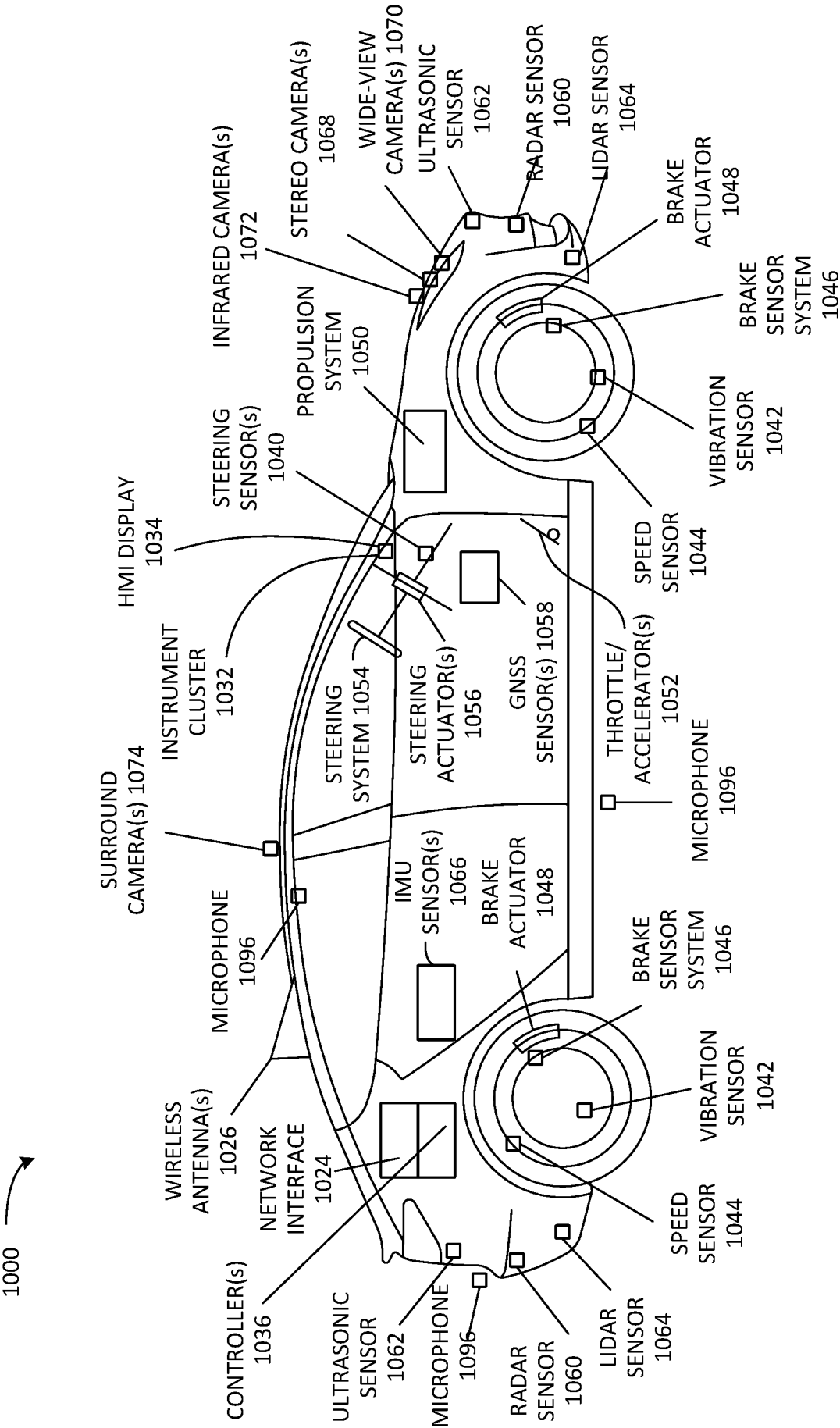


FIG. 10A

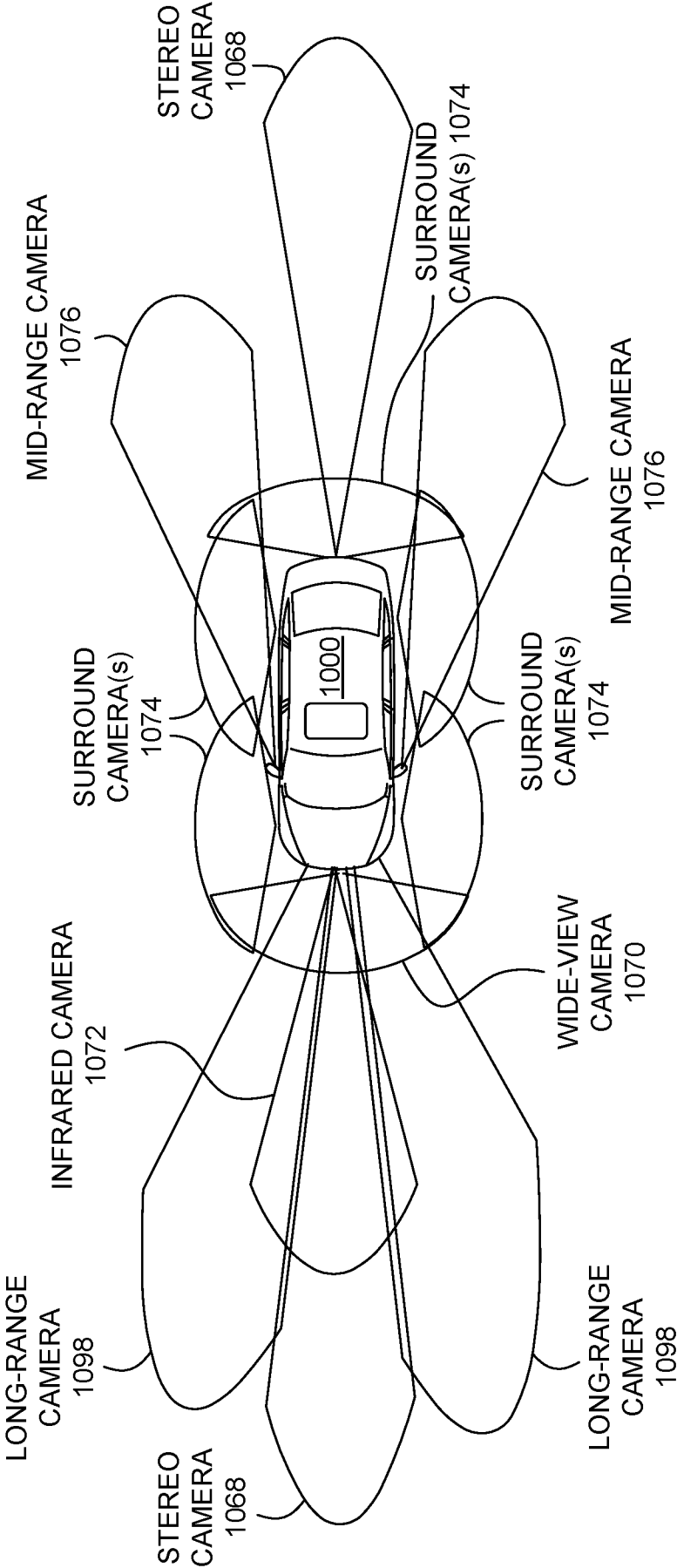


FIG. 10B

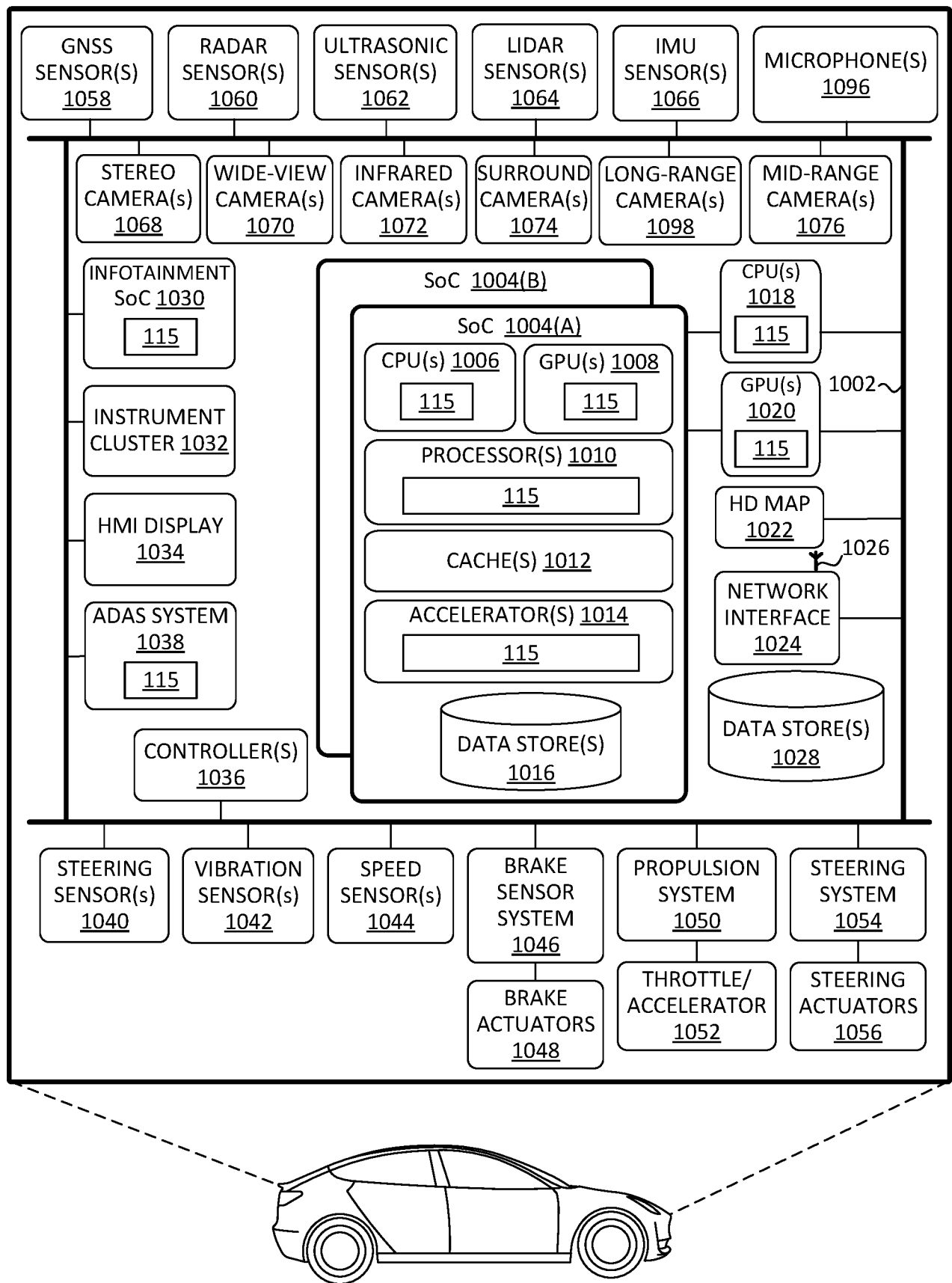
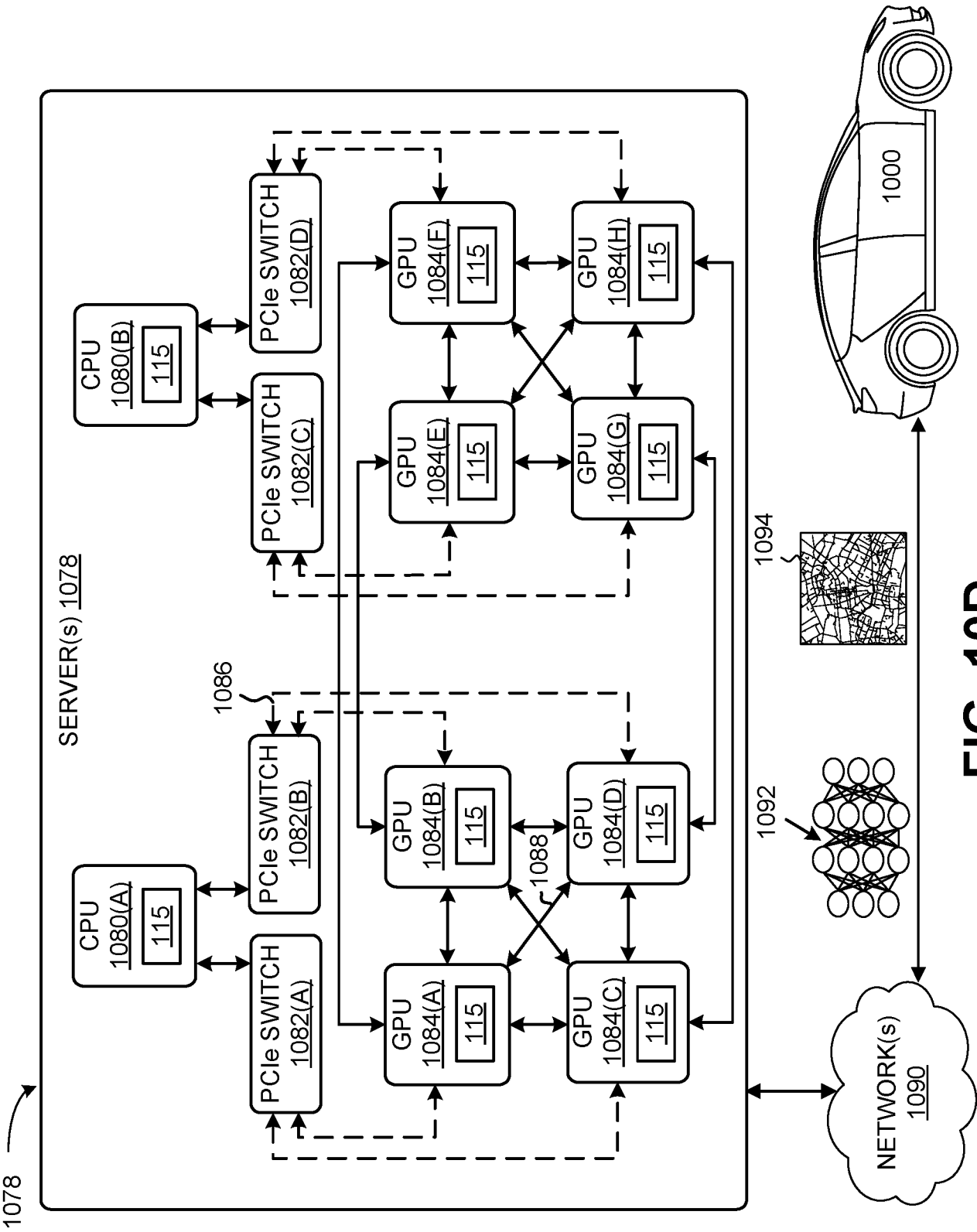
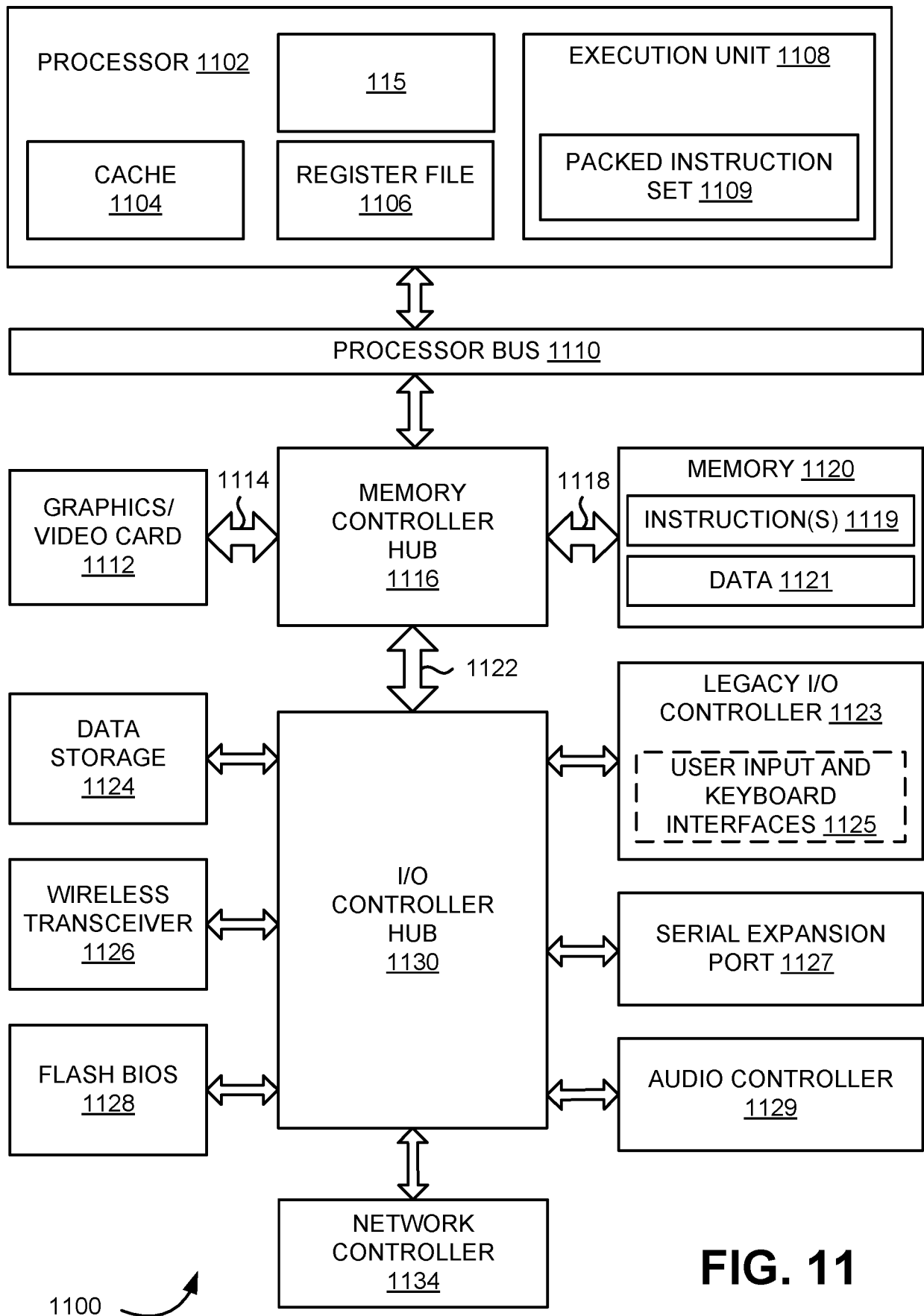
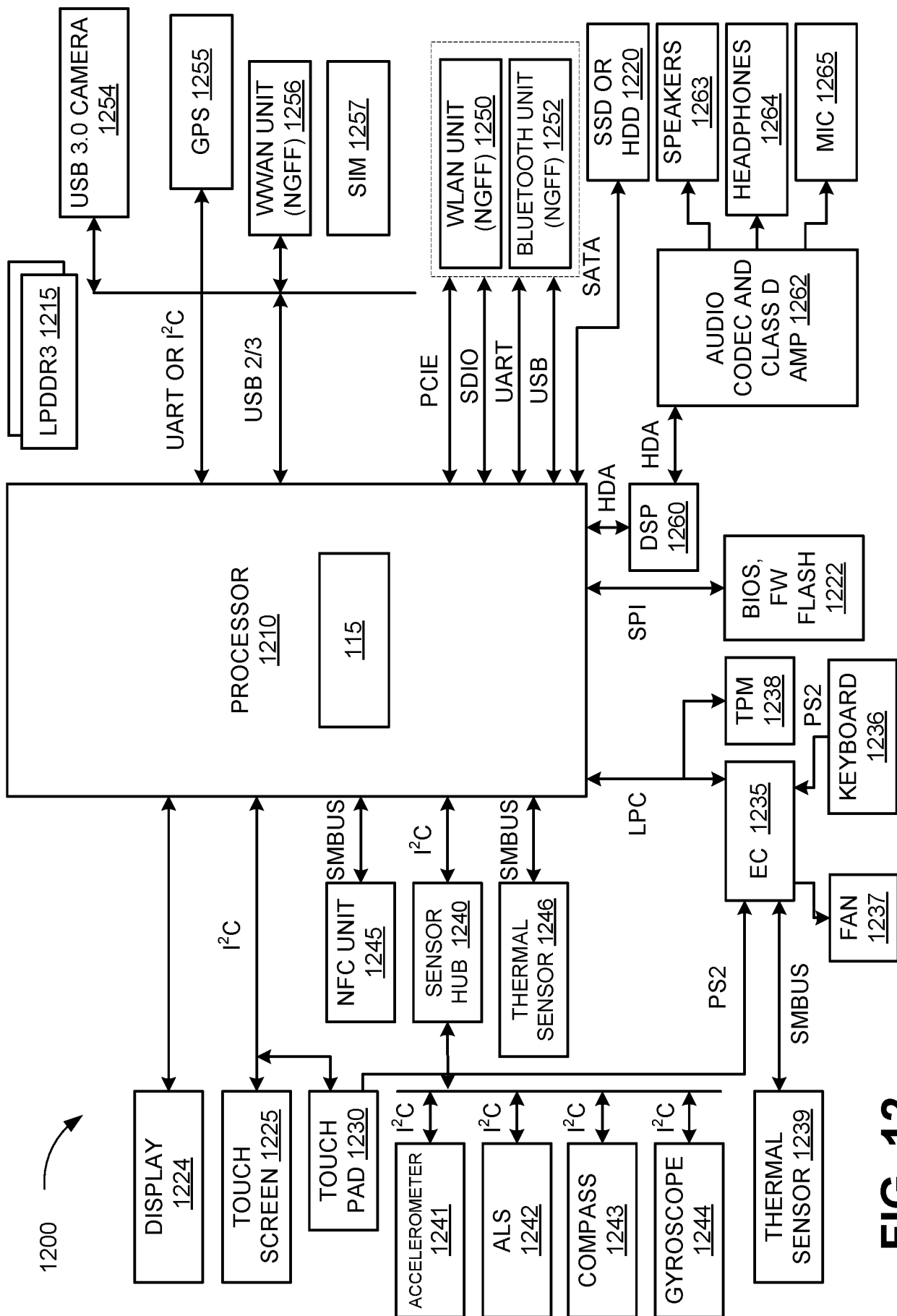
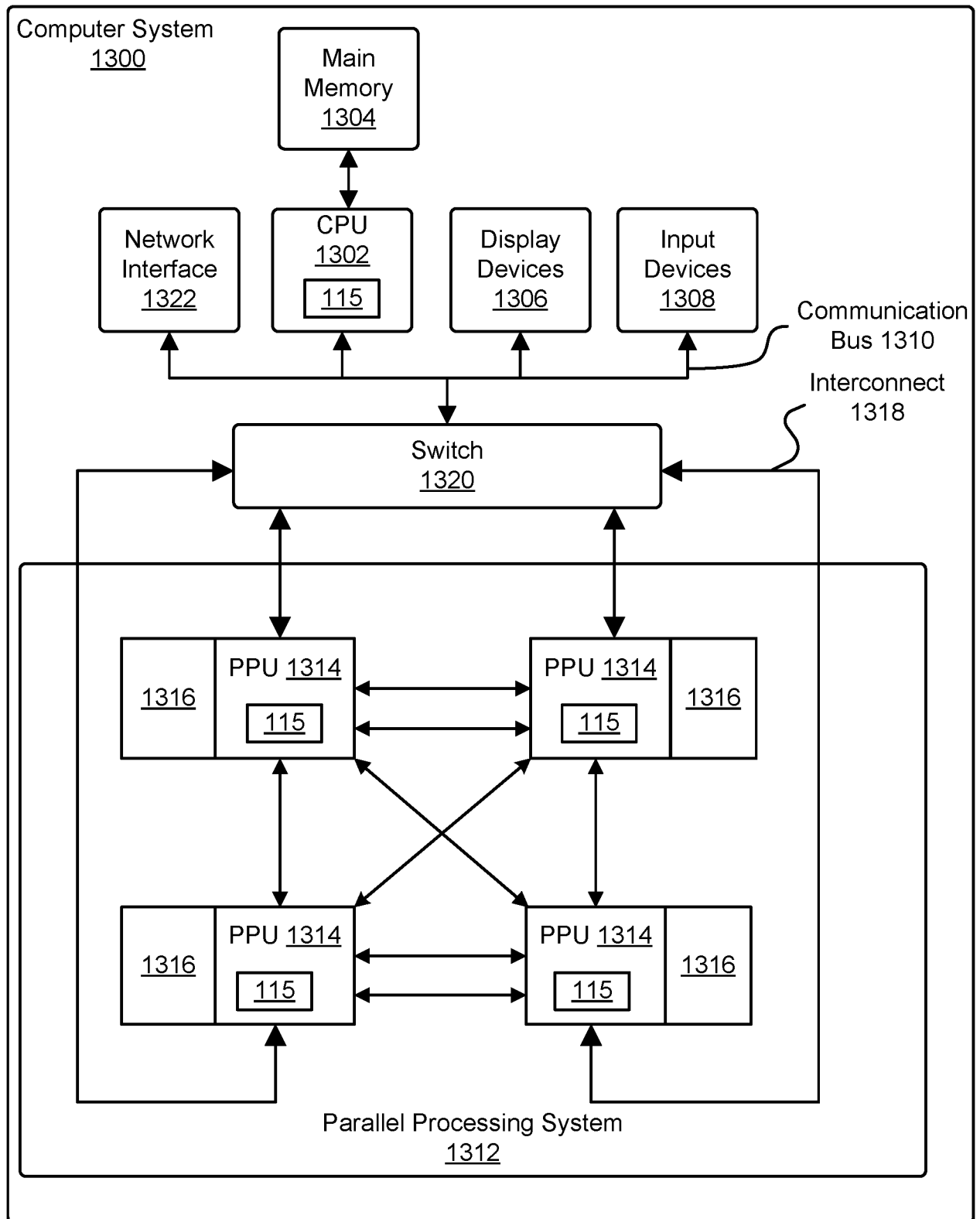


FIG. 10C



**FIG. 11**

**FIG. 12**

**FIG. 13**

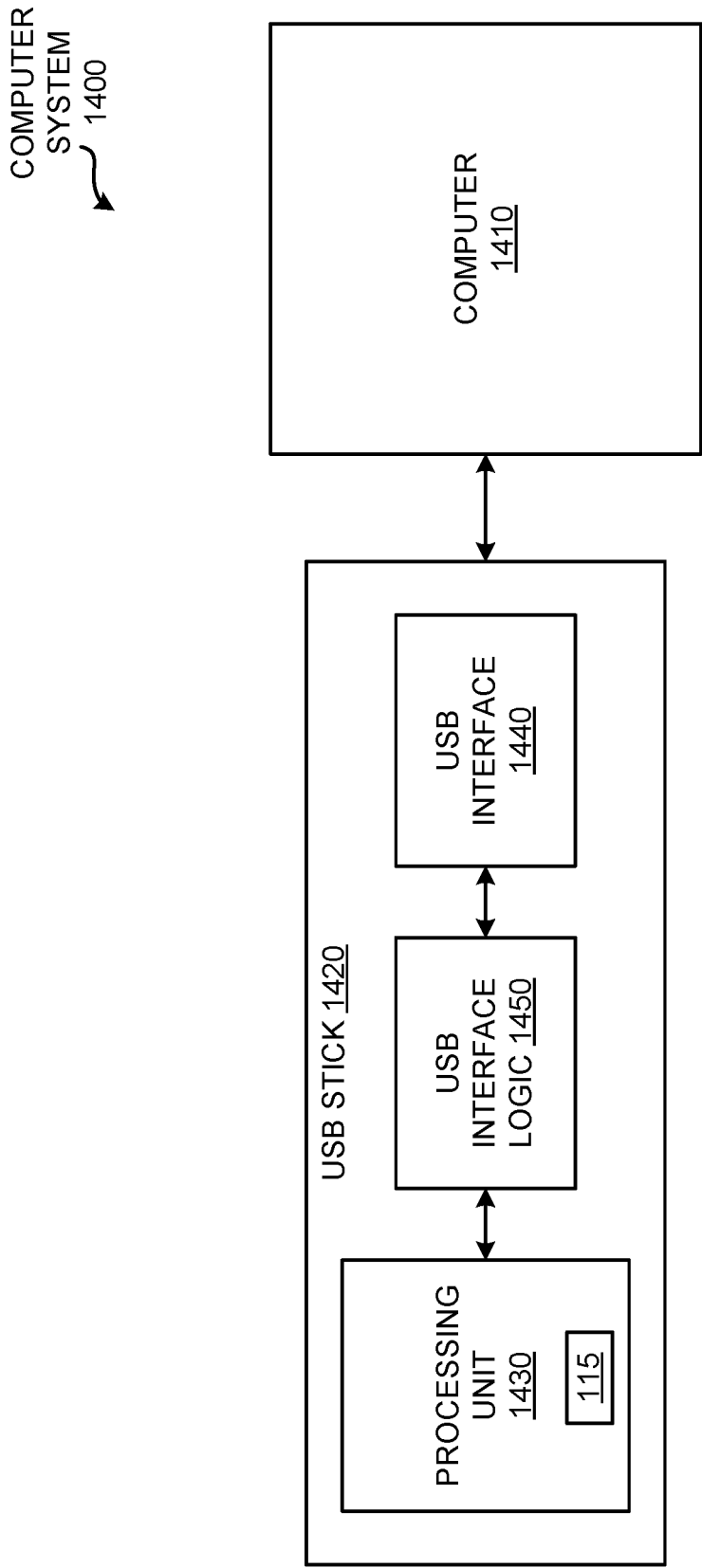


FIG. 14

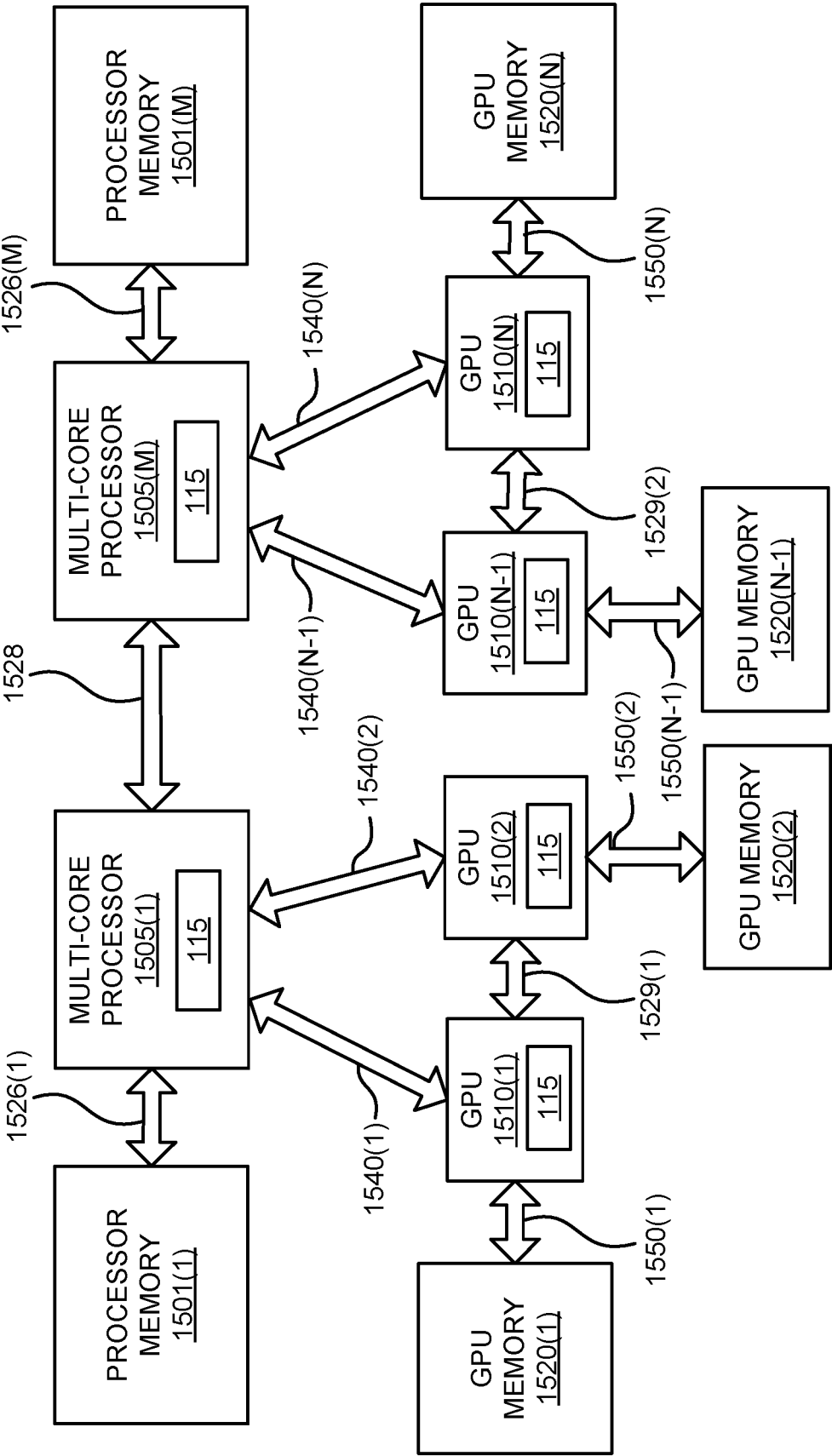


FIG. 15A

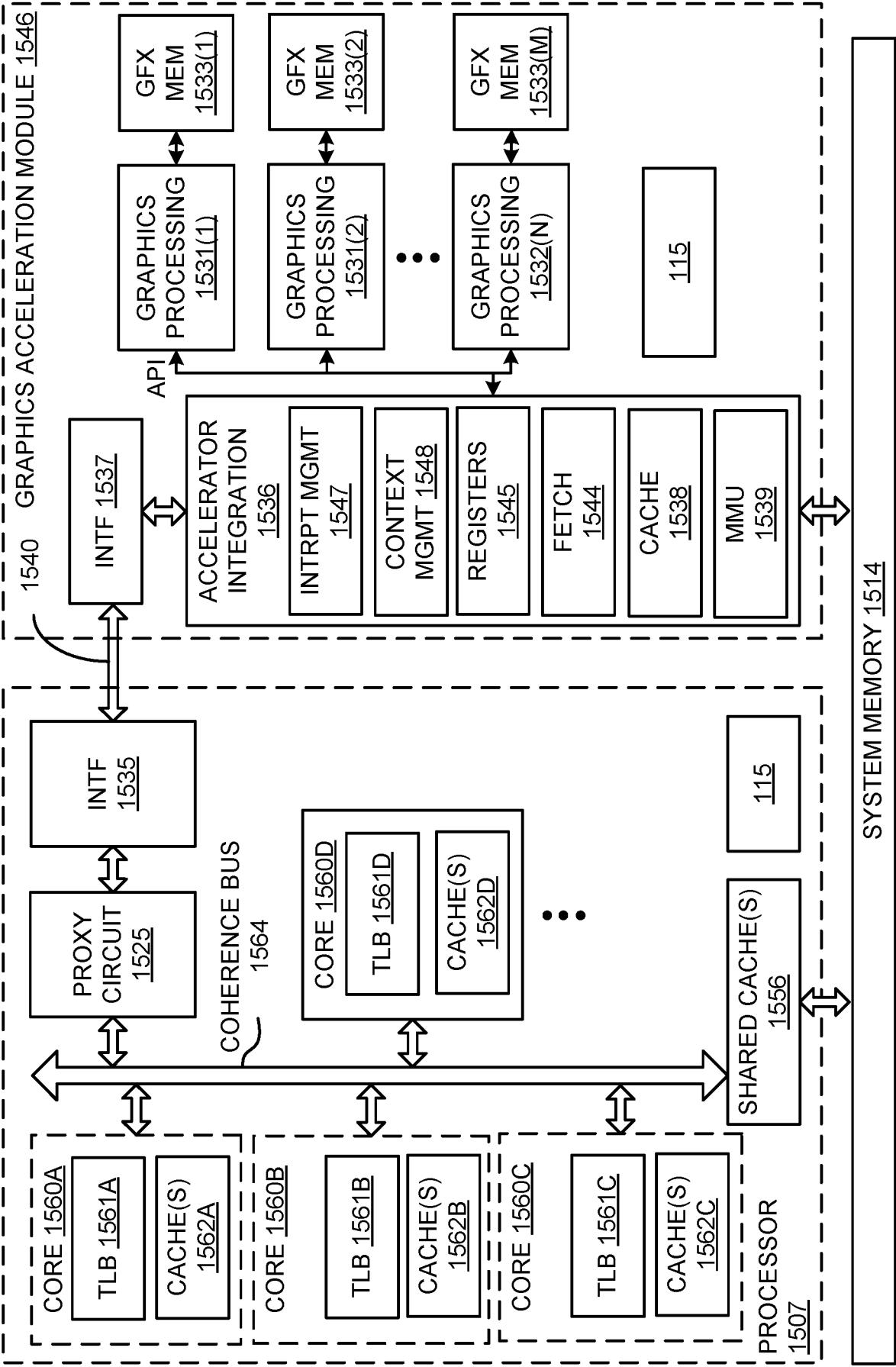


FIG. 15B

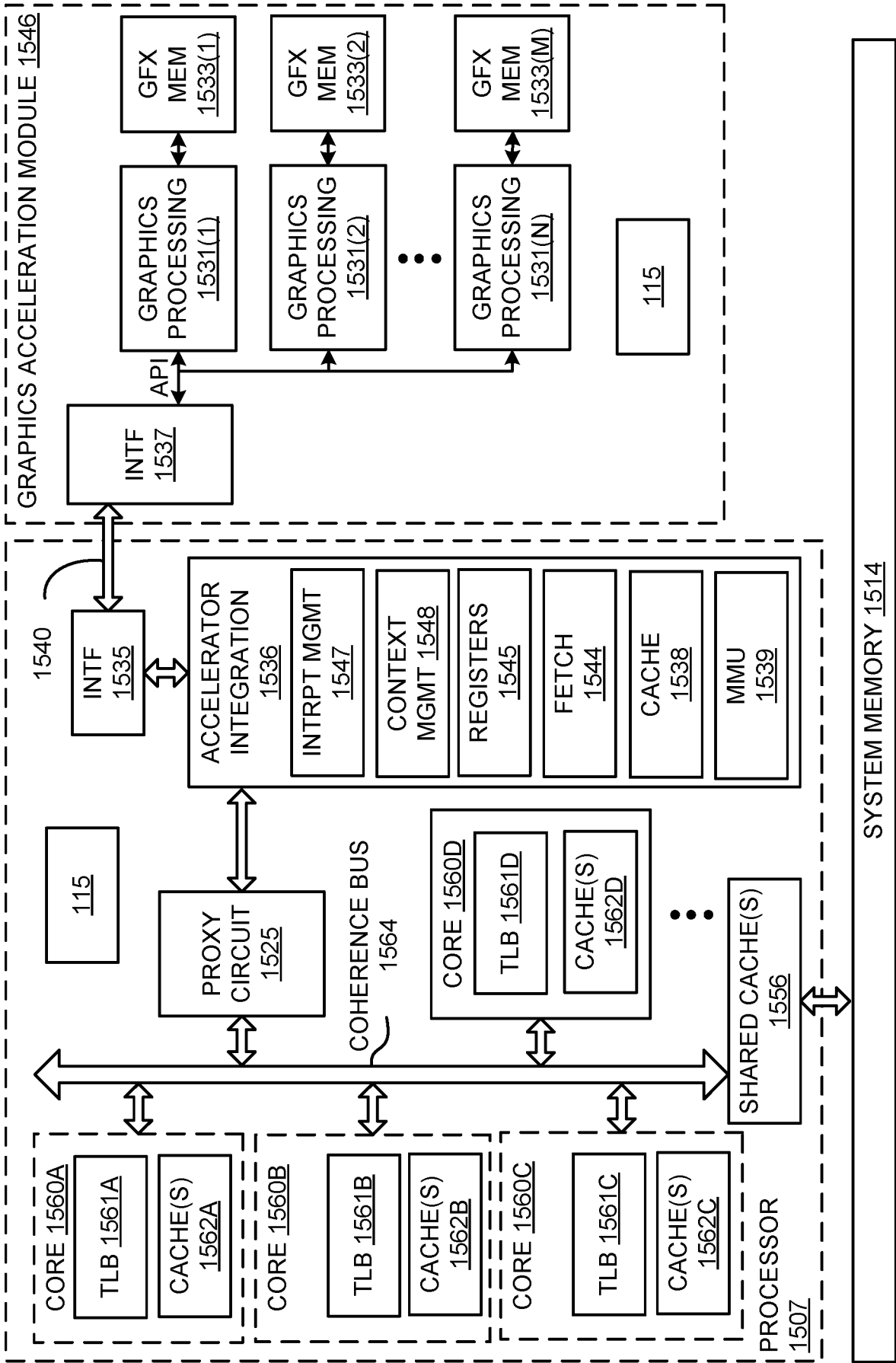
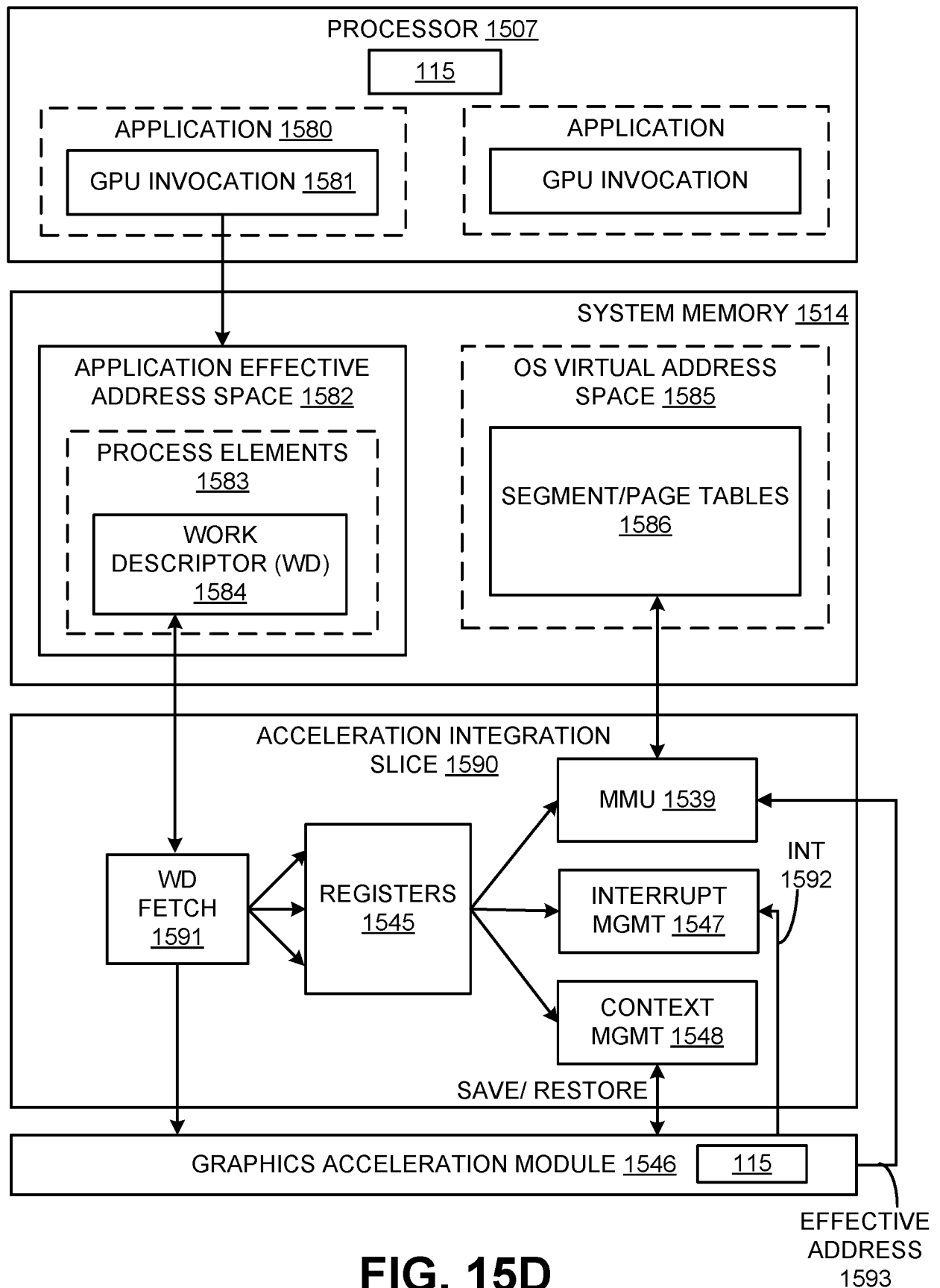


FIG. 15C

**FIG. 15D**

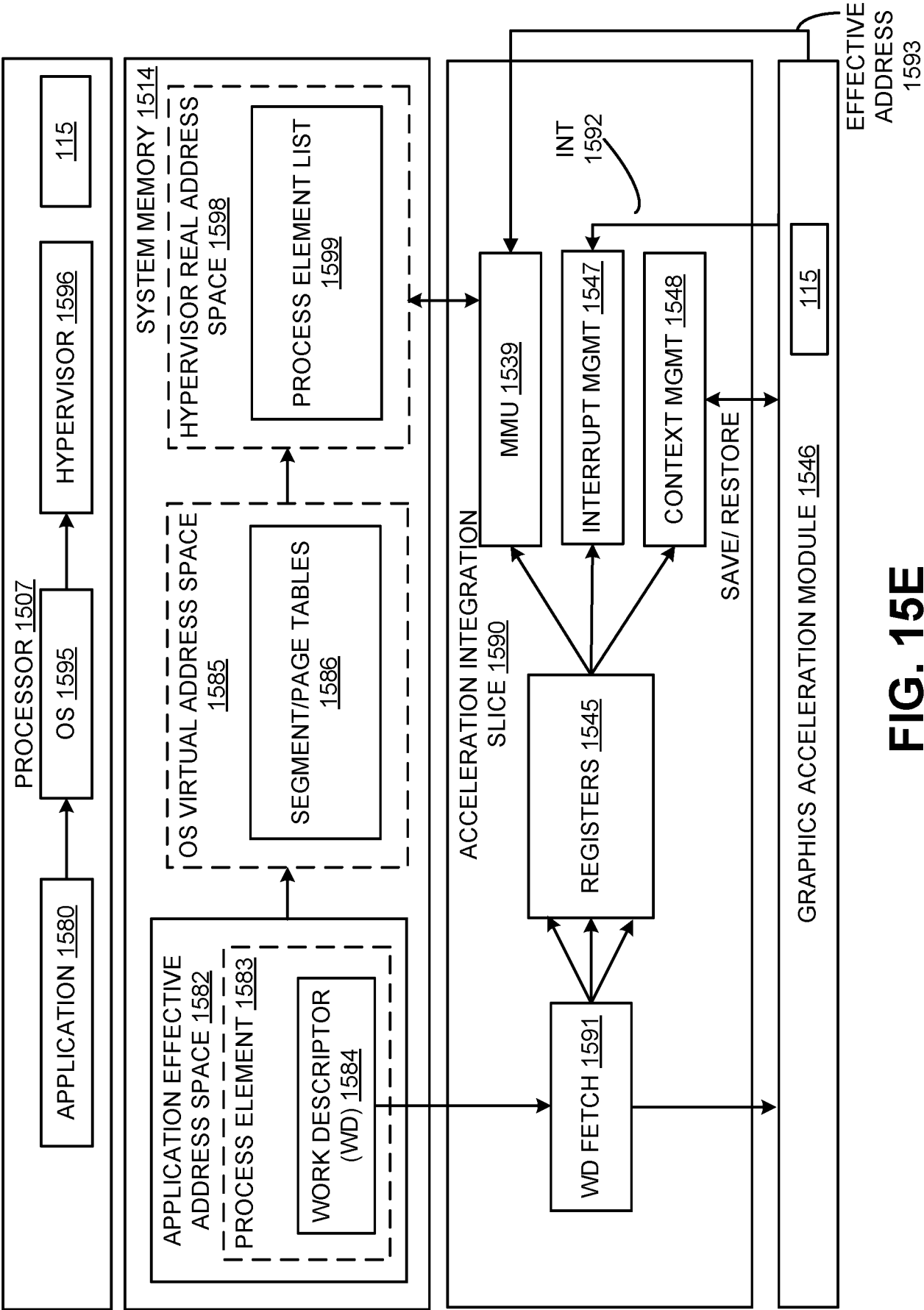


FIG. 15E

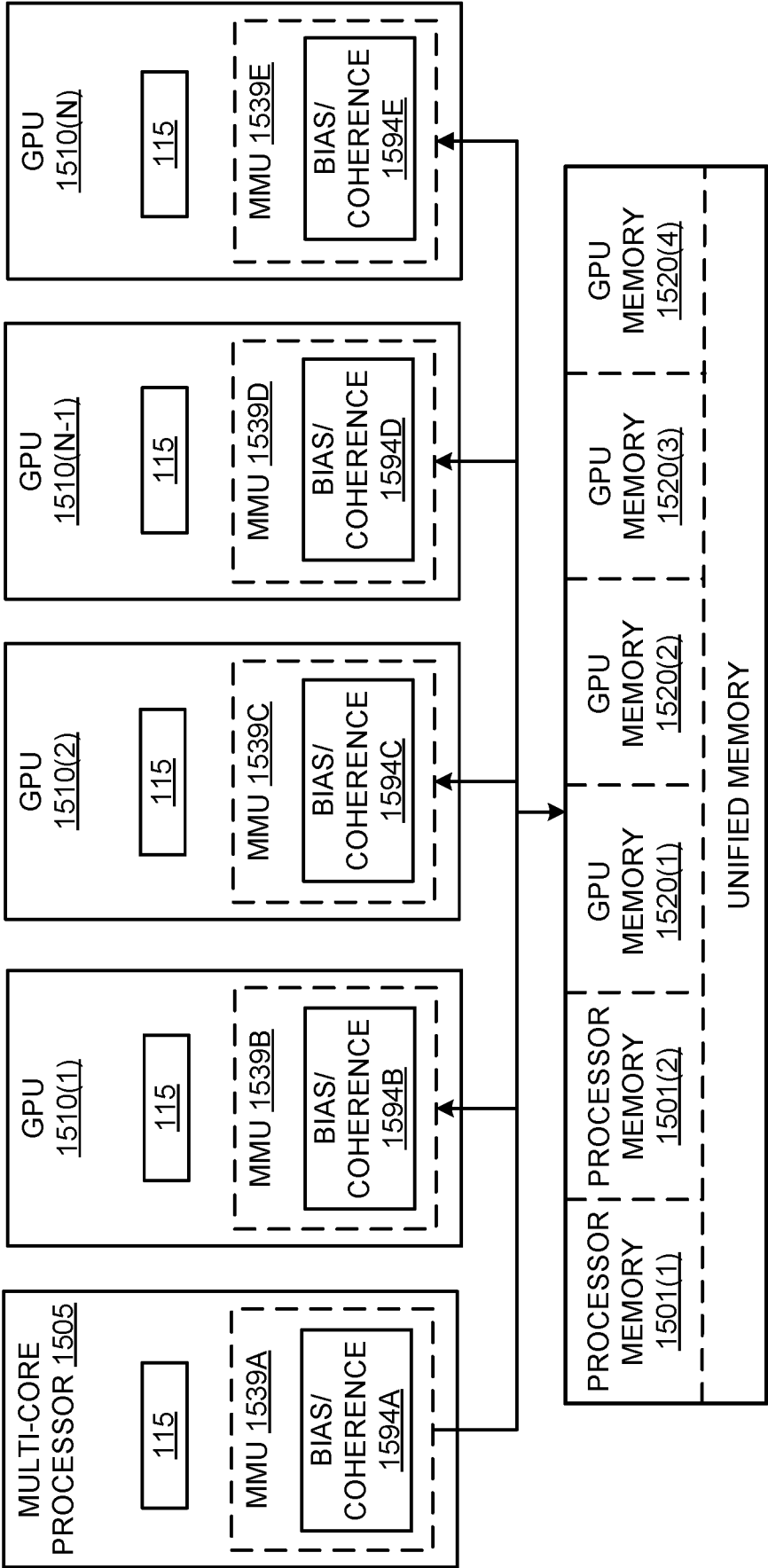


FIG. 15F

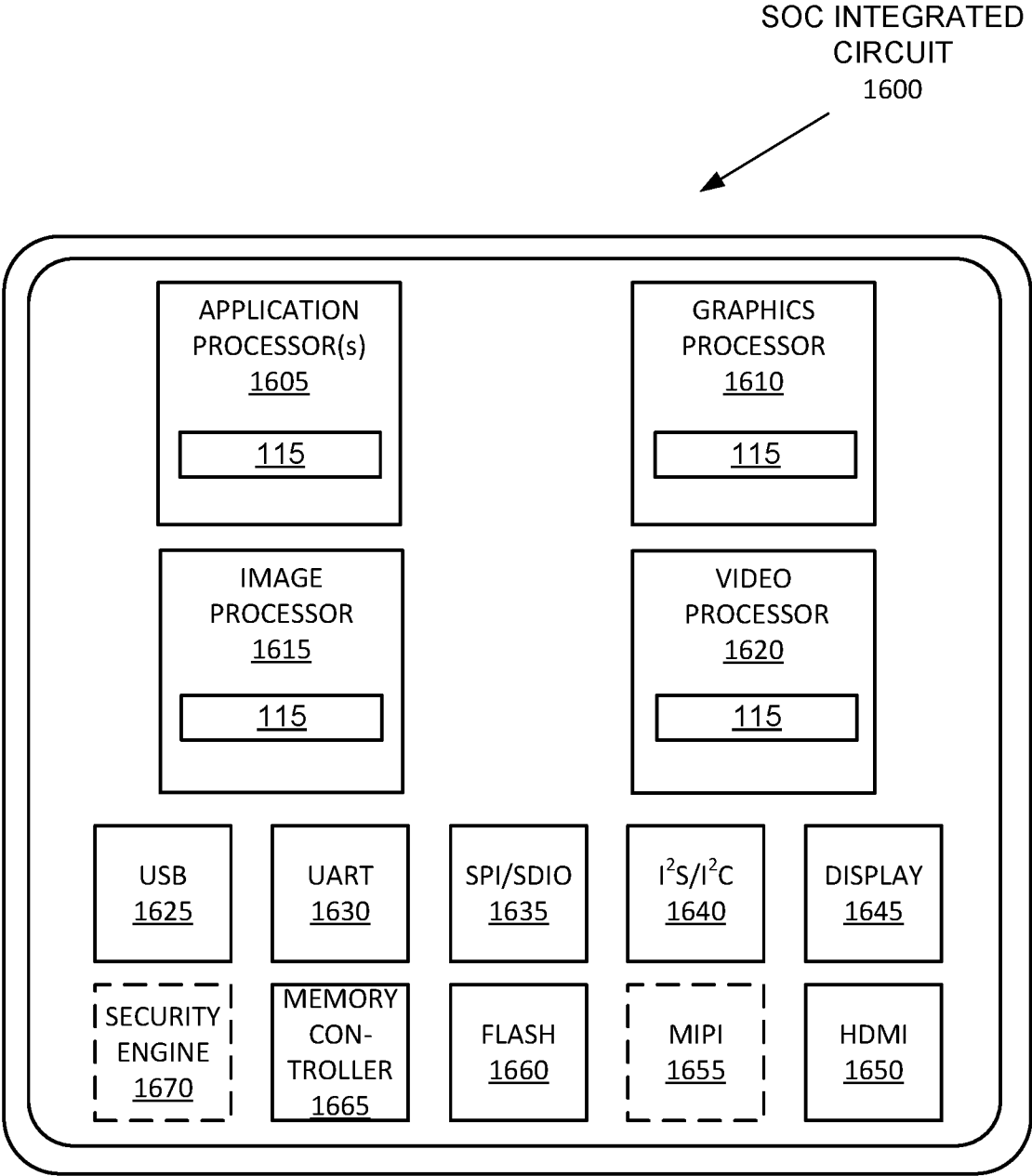


FIG. 16

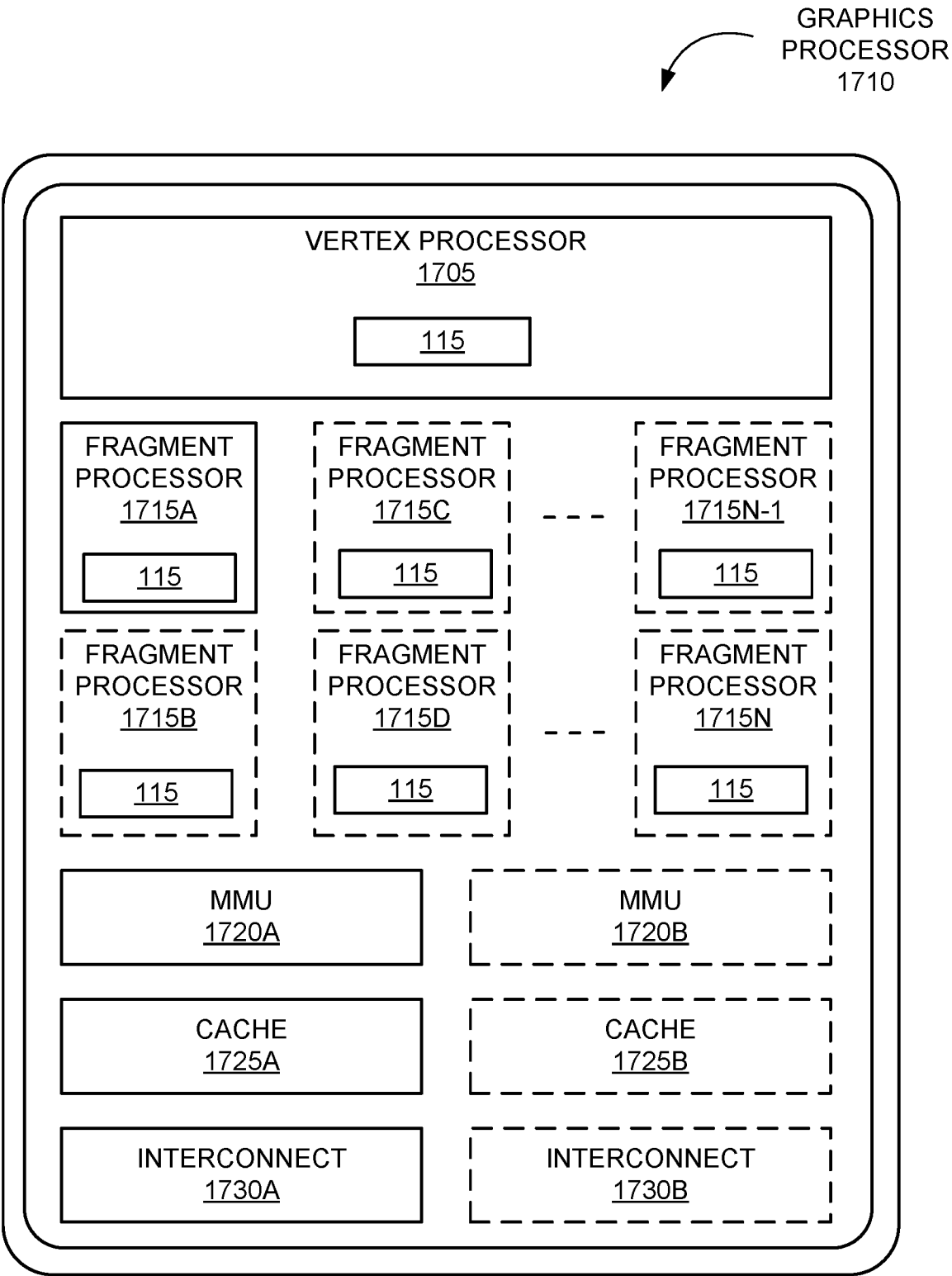


FIG. 17A

GRAPHICS
PROCESSOR
1740

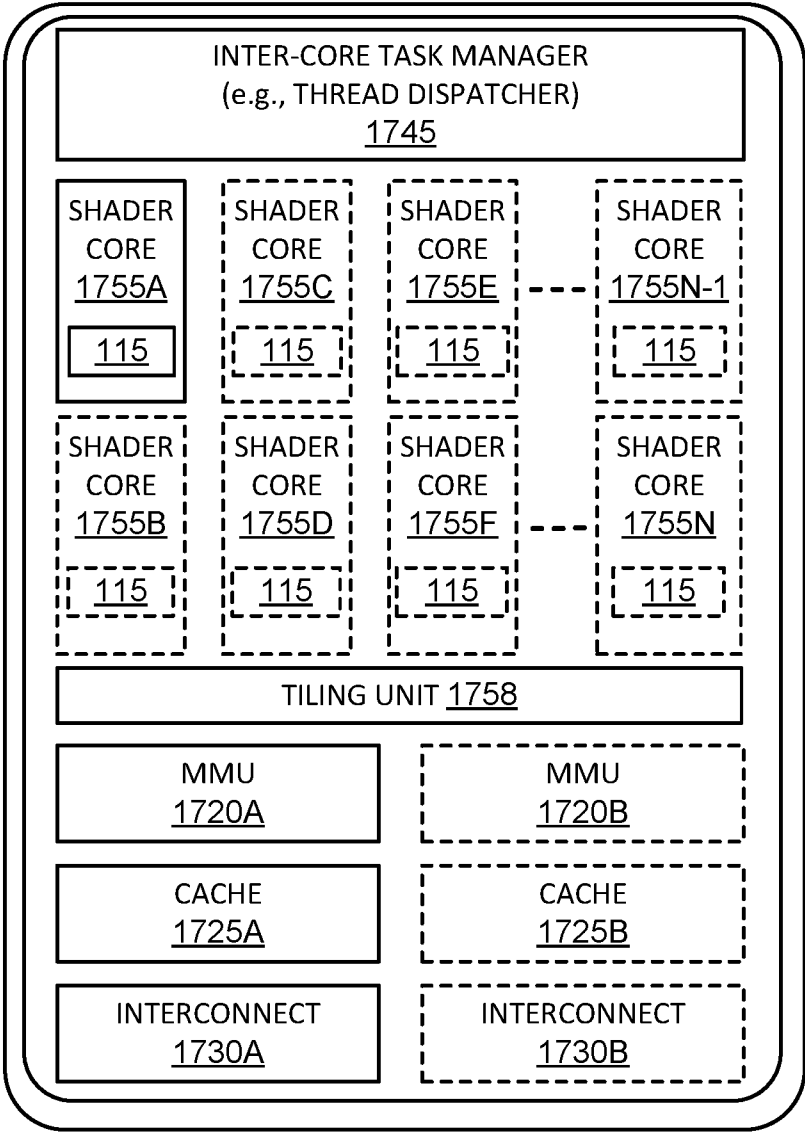
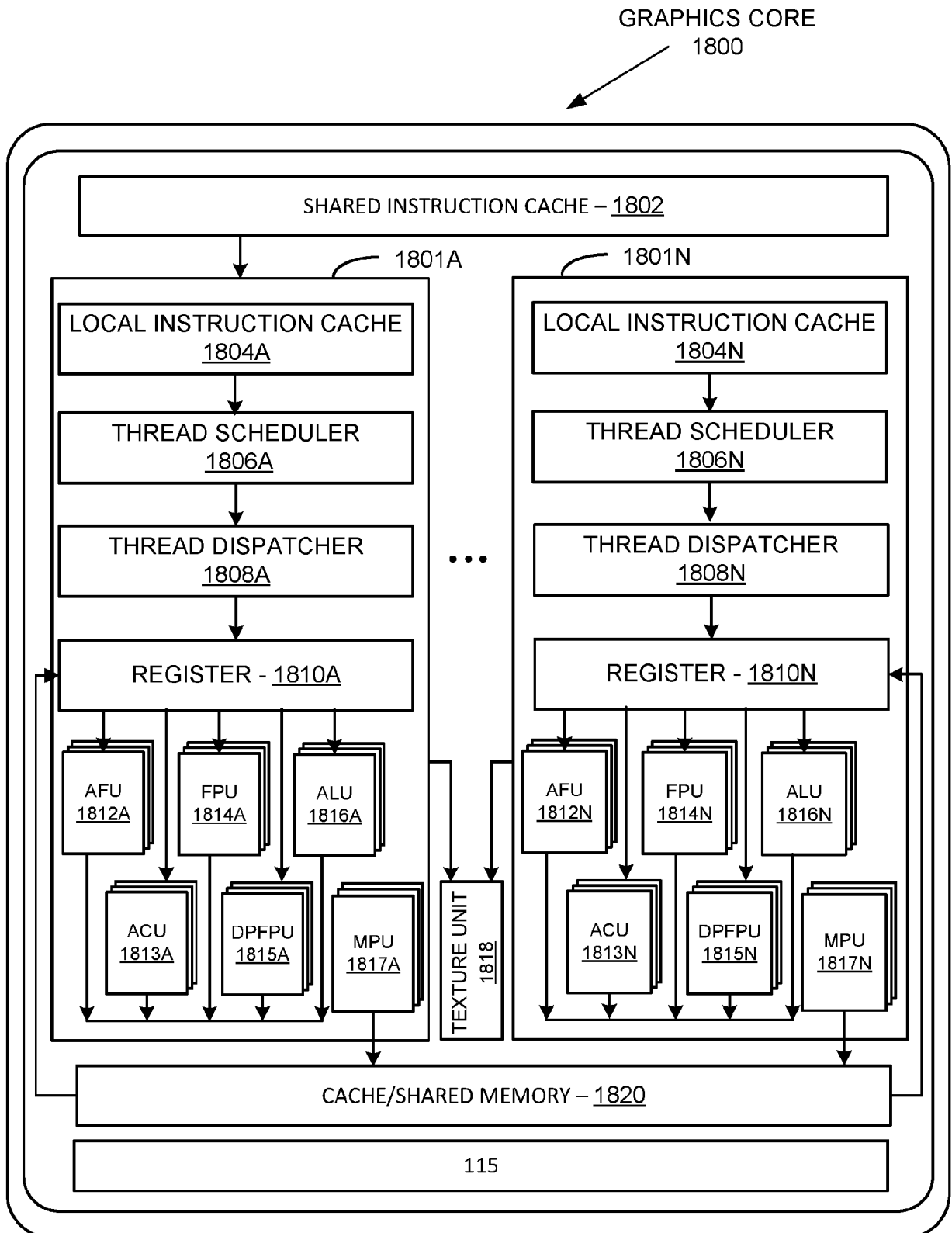


FIG. 17B

27 / 54

**FIG. 18A**

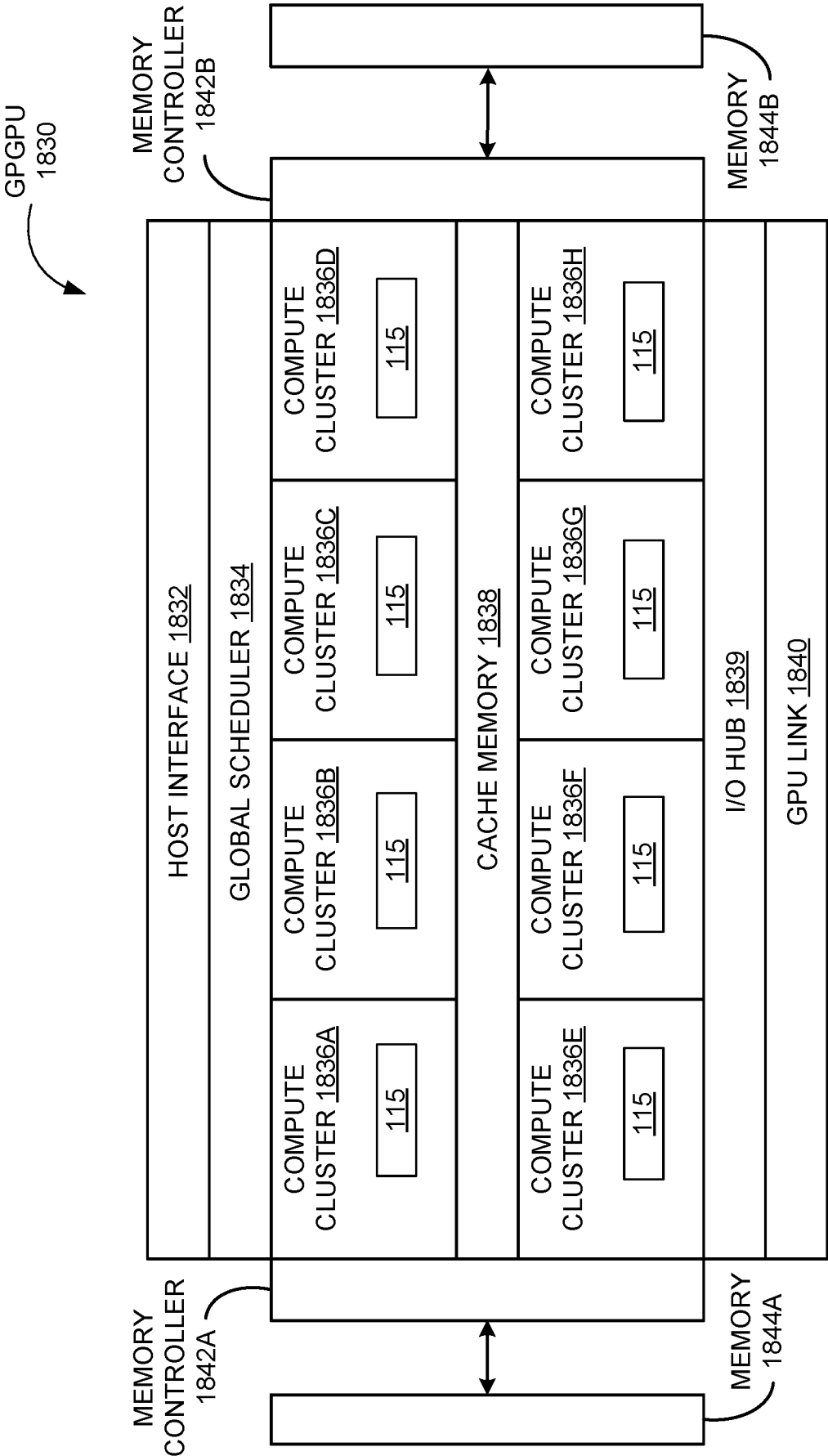
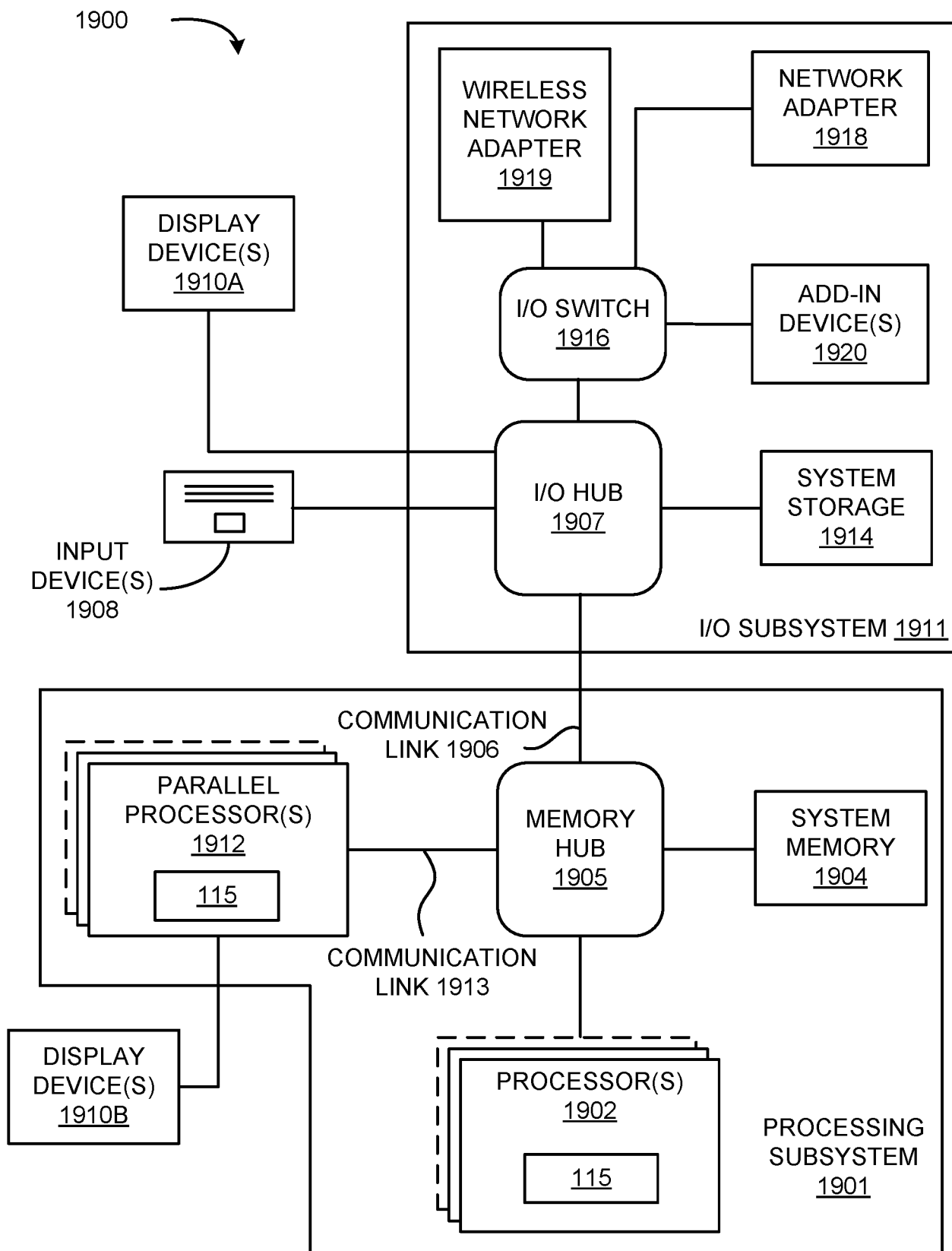
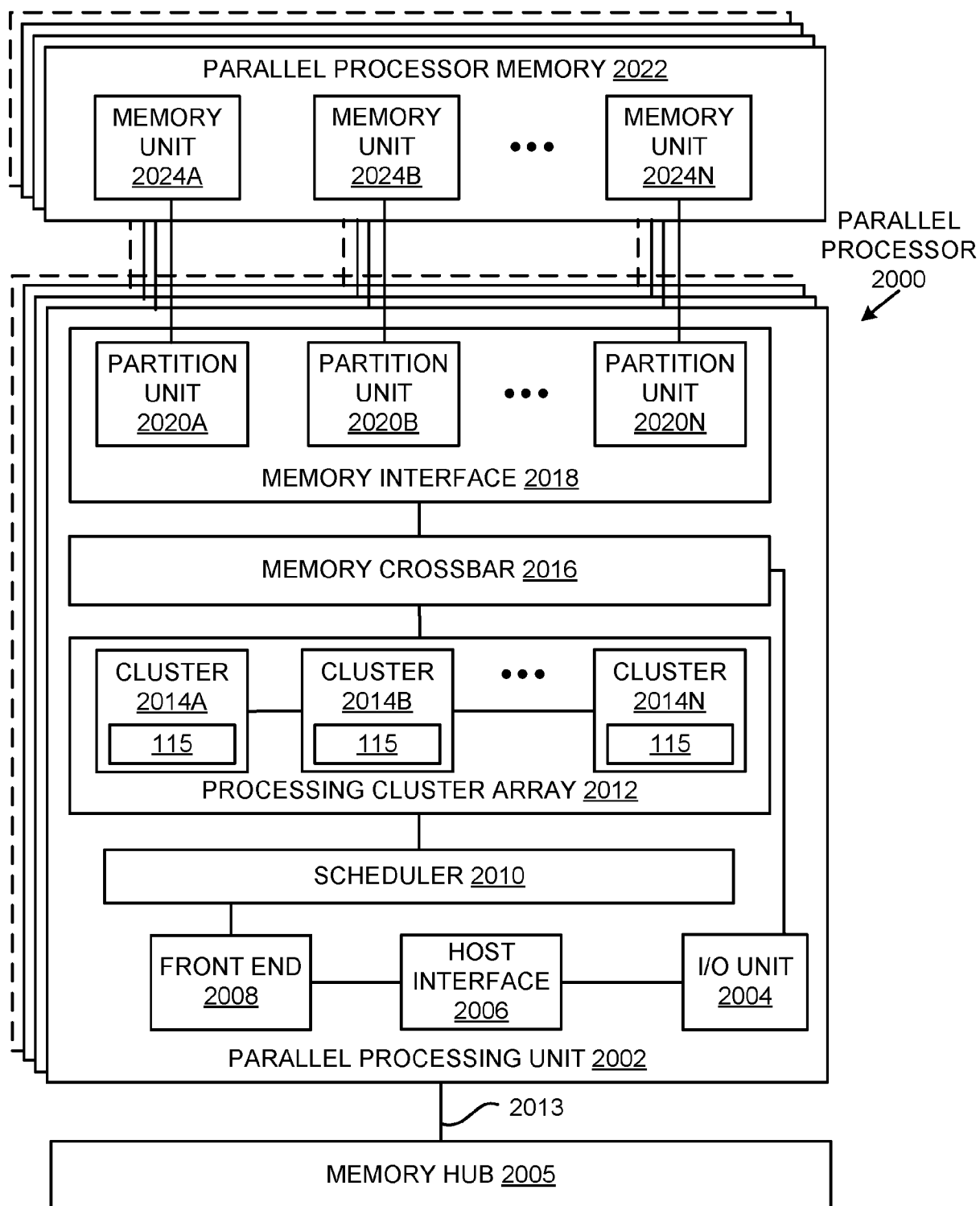
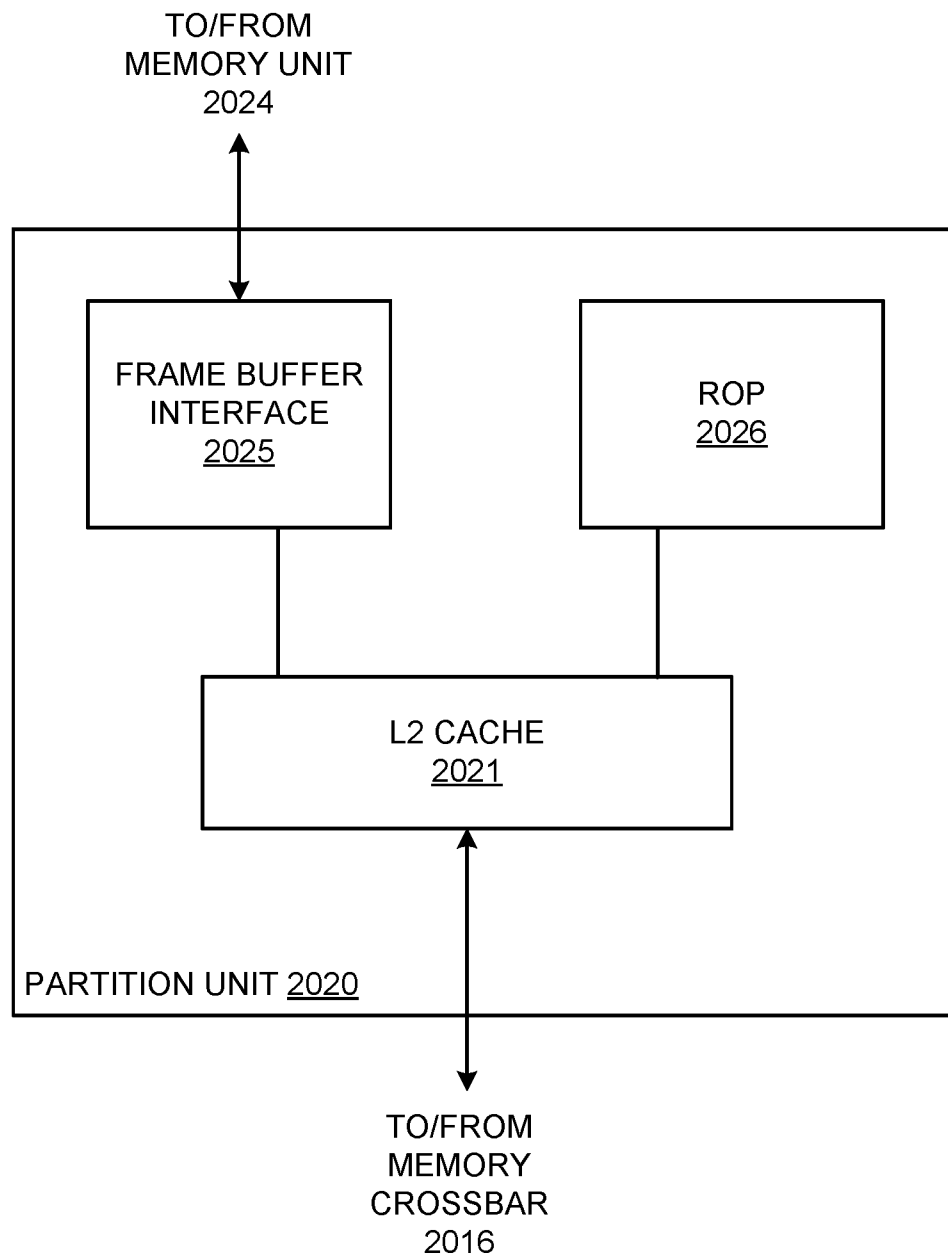


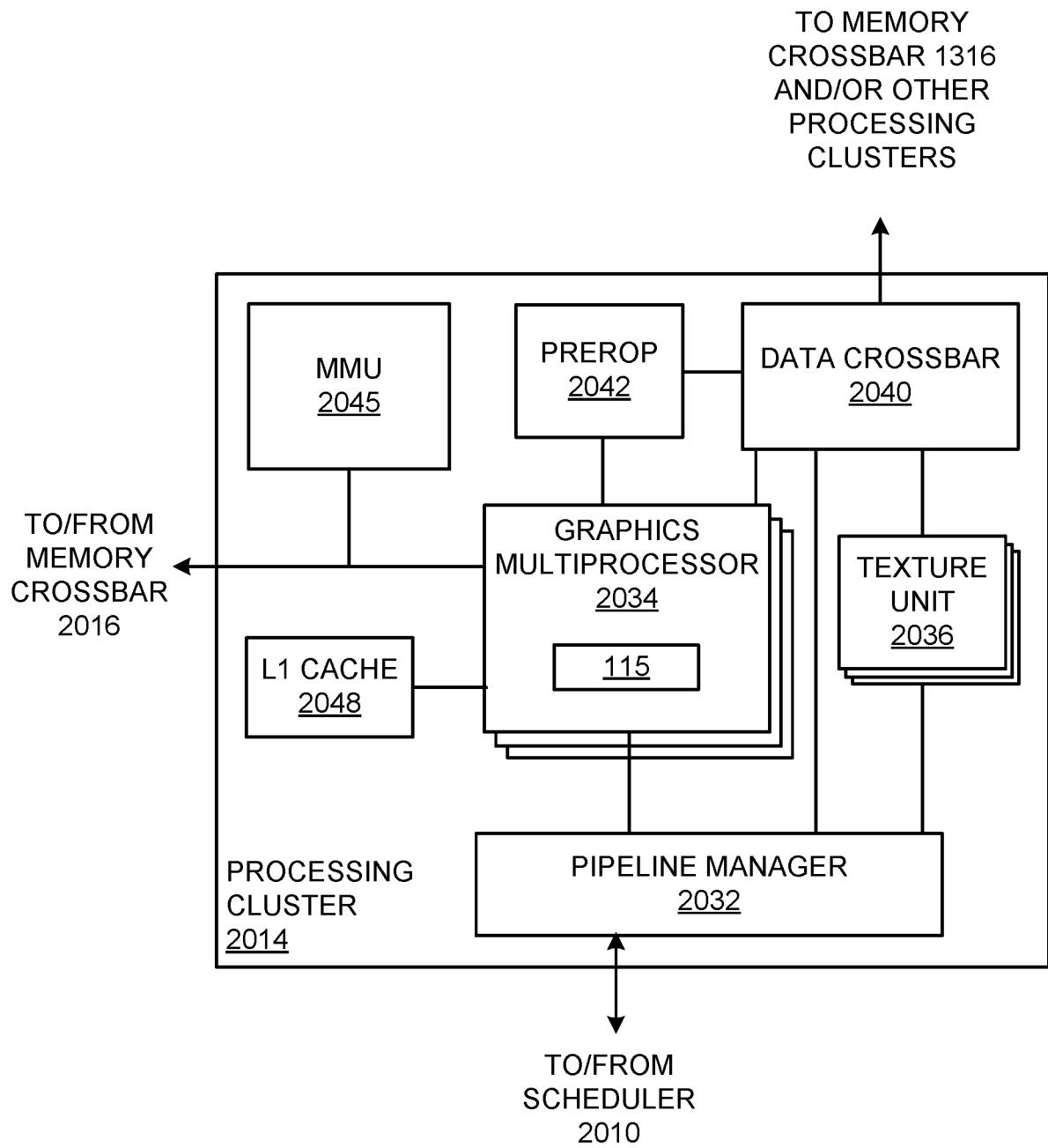
FIG. 18B

**FIG. 19**

30 / 54

**FIG. 20A**

**FIG. 20B**

**FIG. 20C**

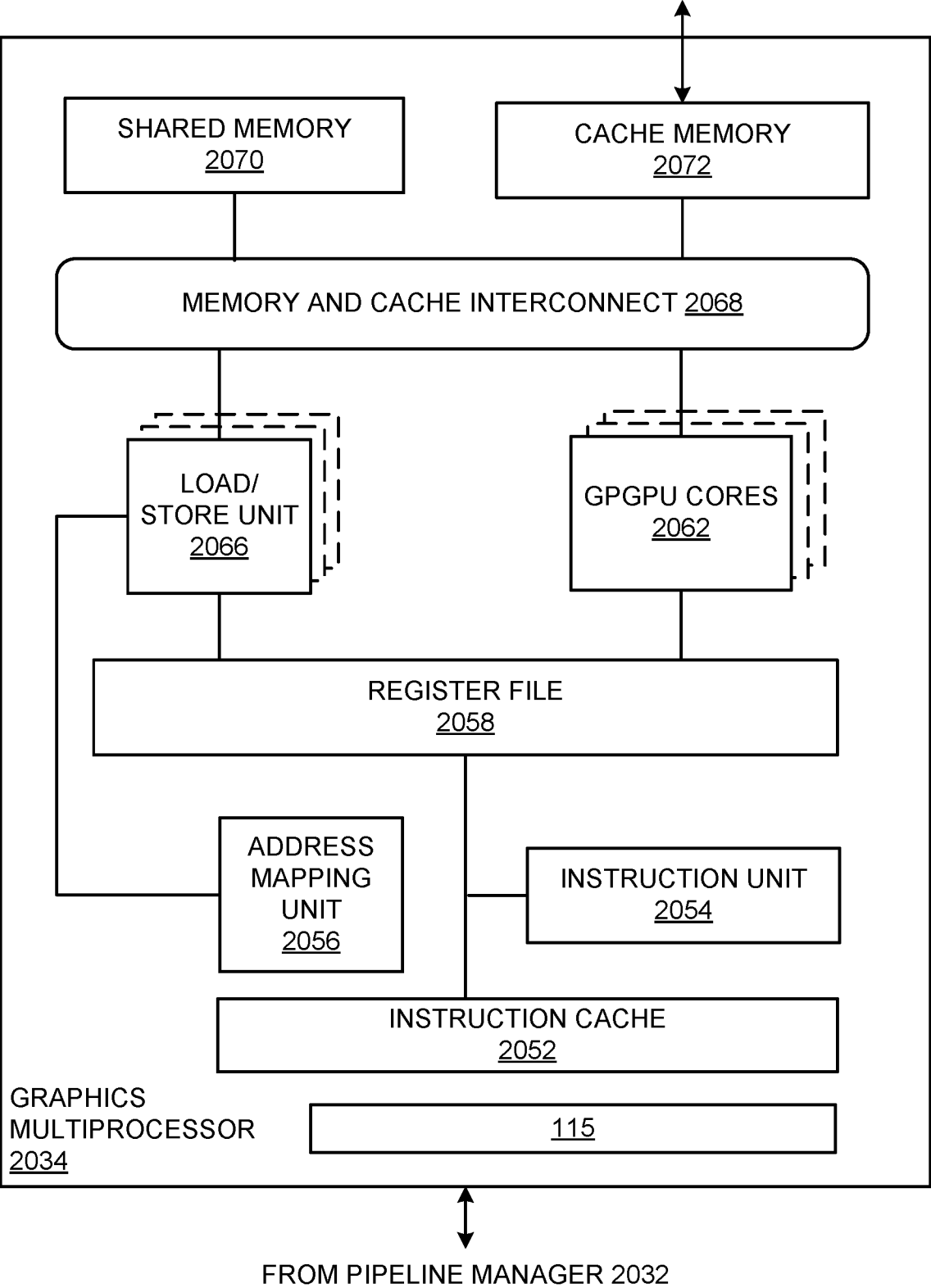
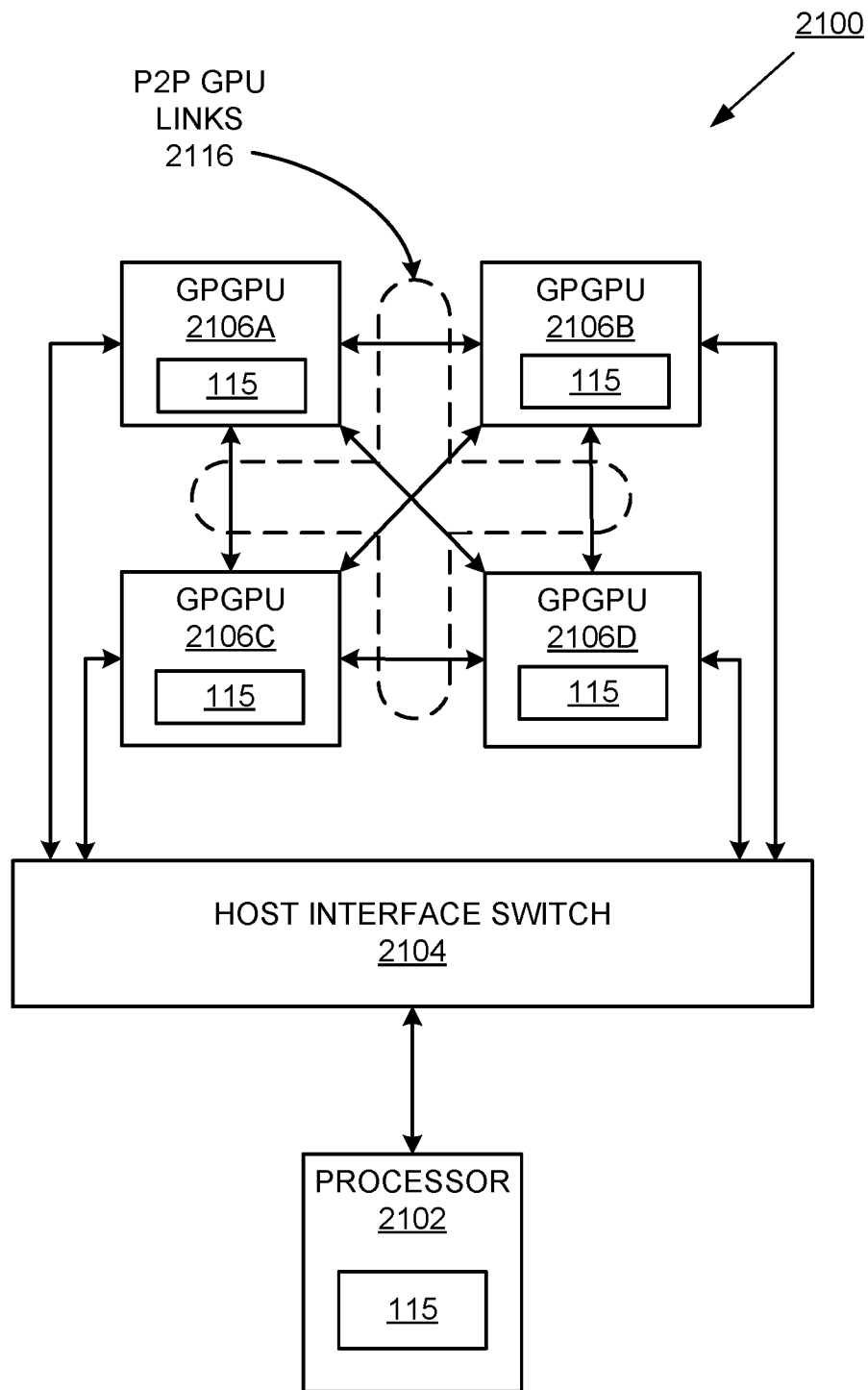


FIG. 20D

**FIG. 21**

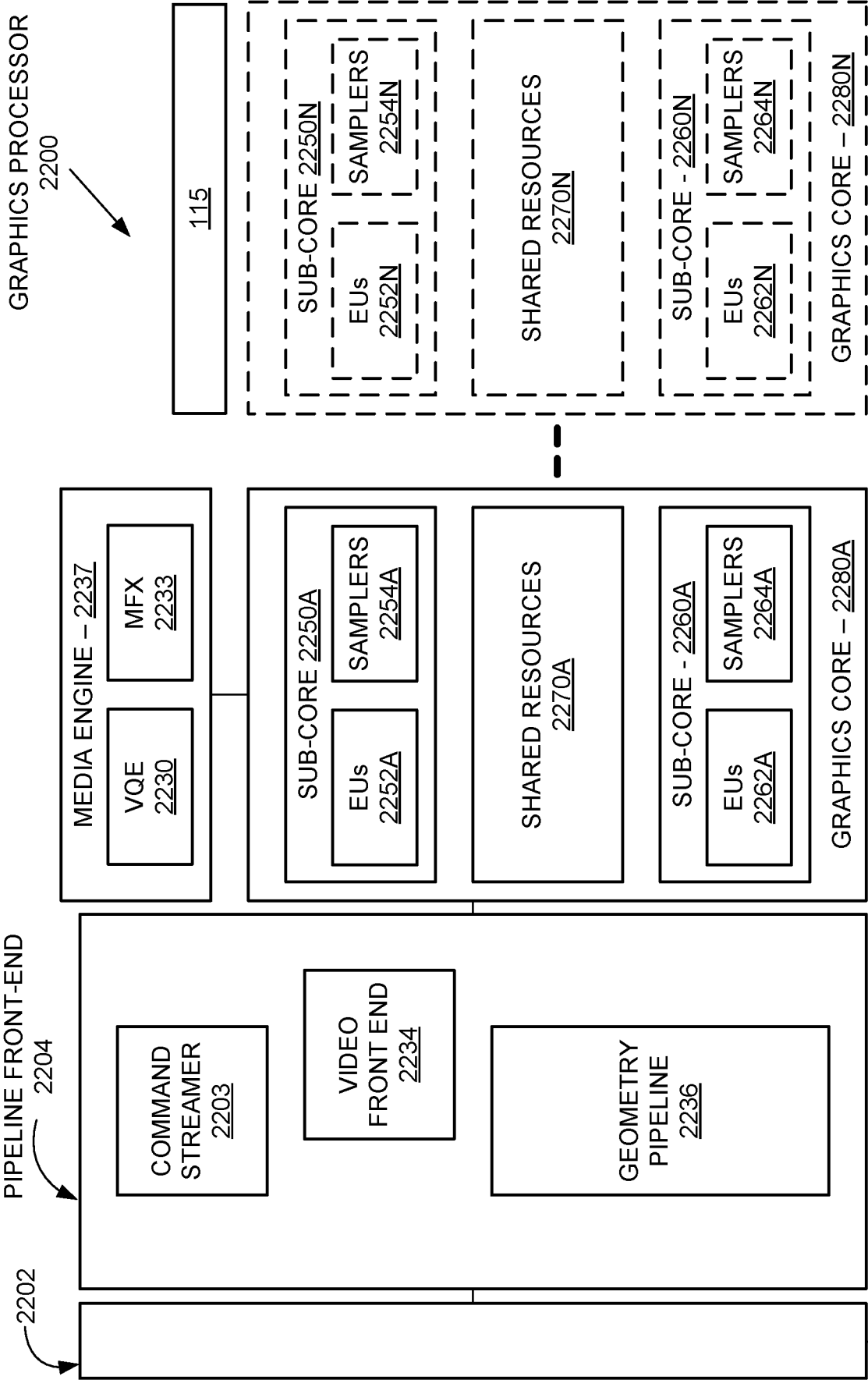


FIG. 22

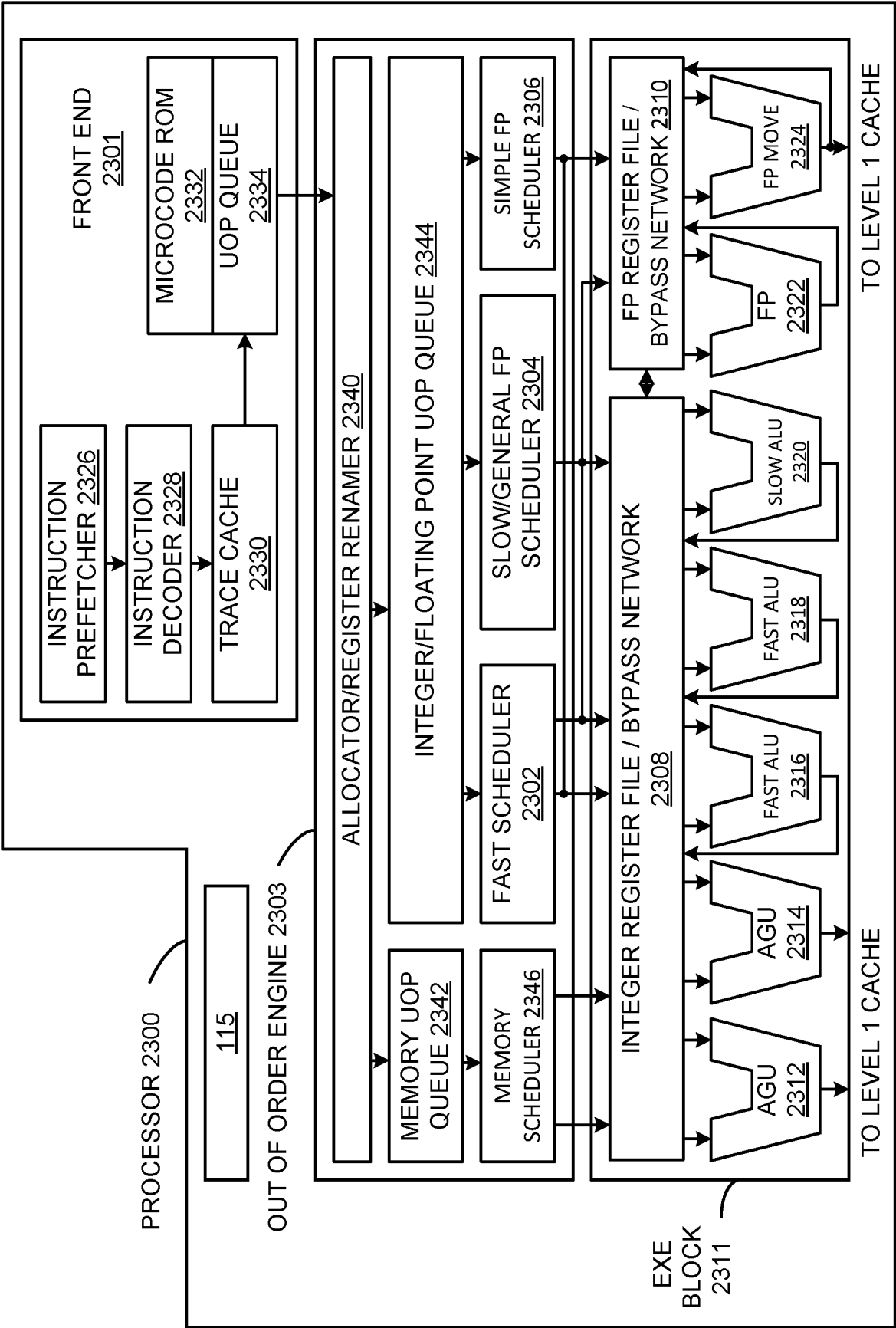


FIG. 23

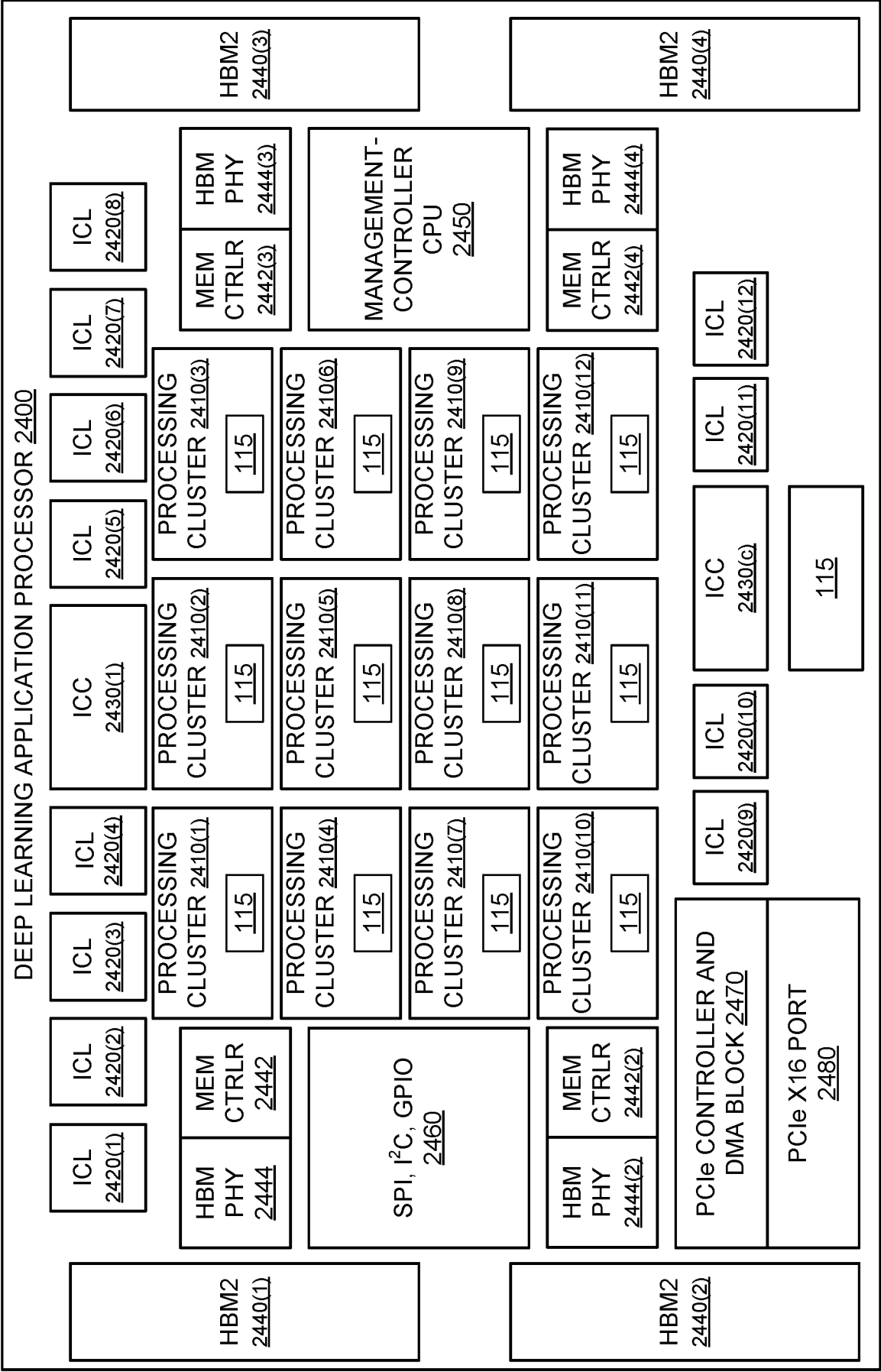


FIG. 24

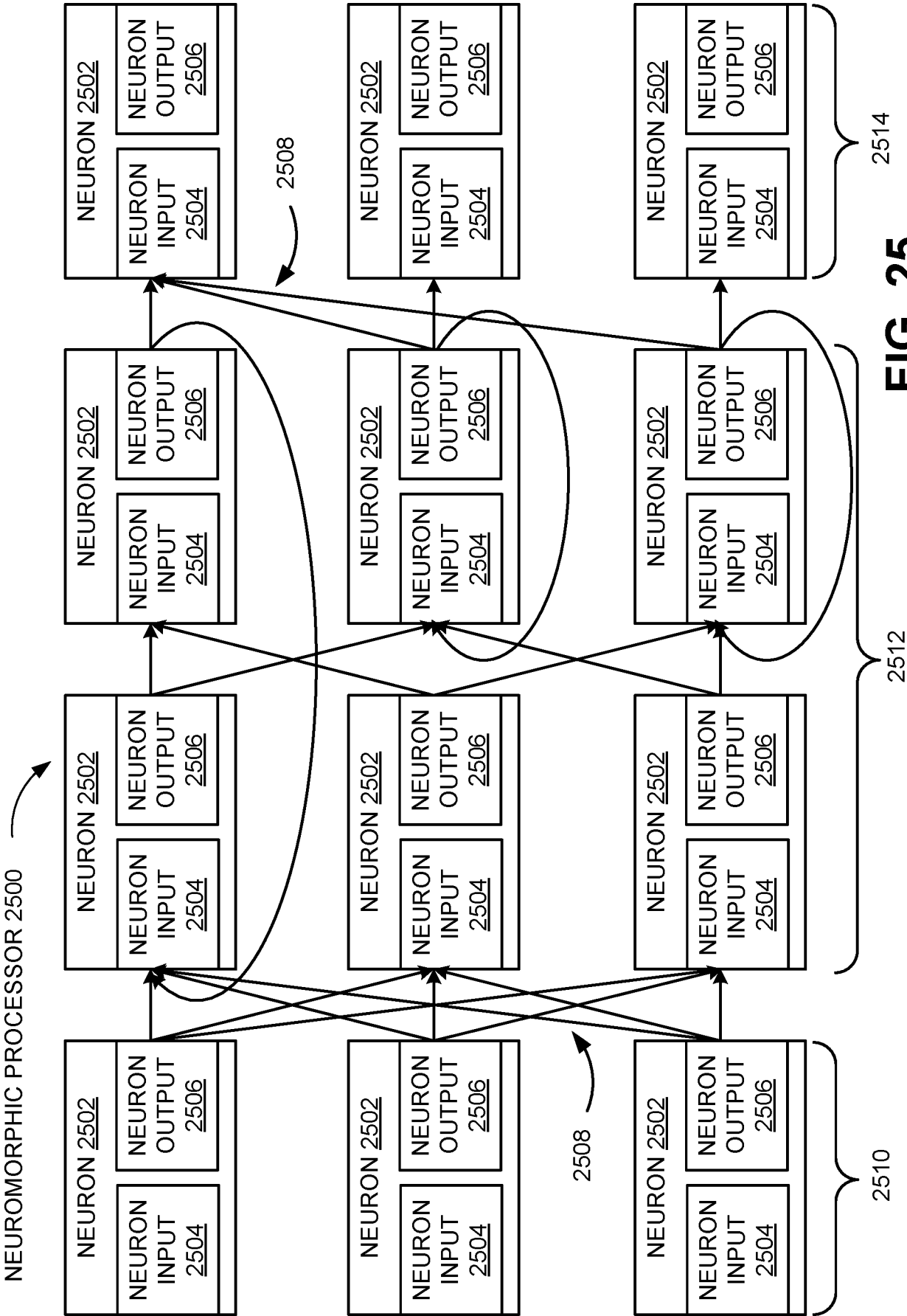
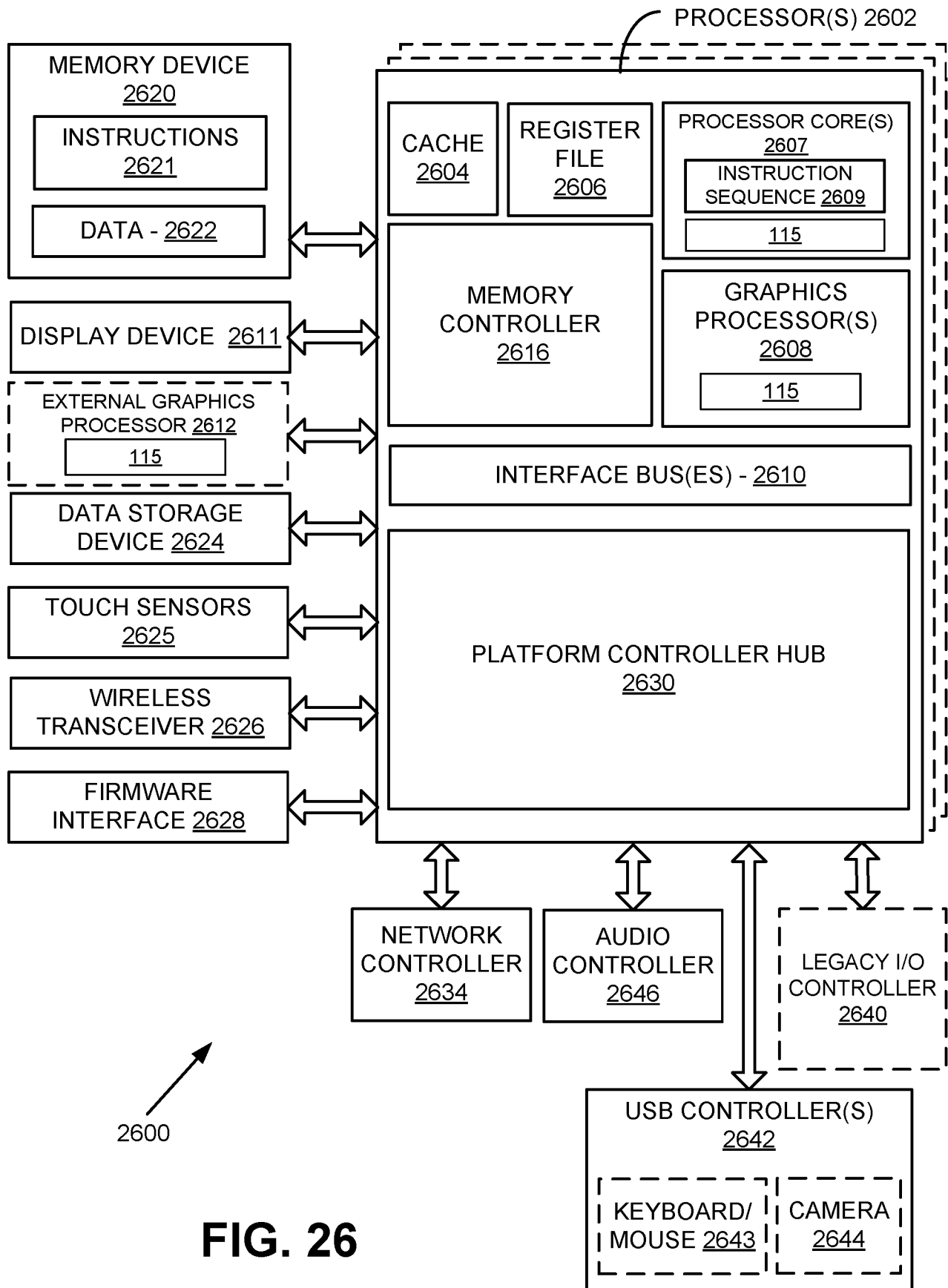


FIG. 25



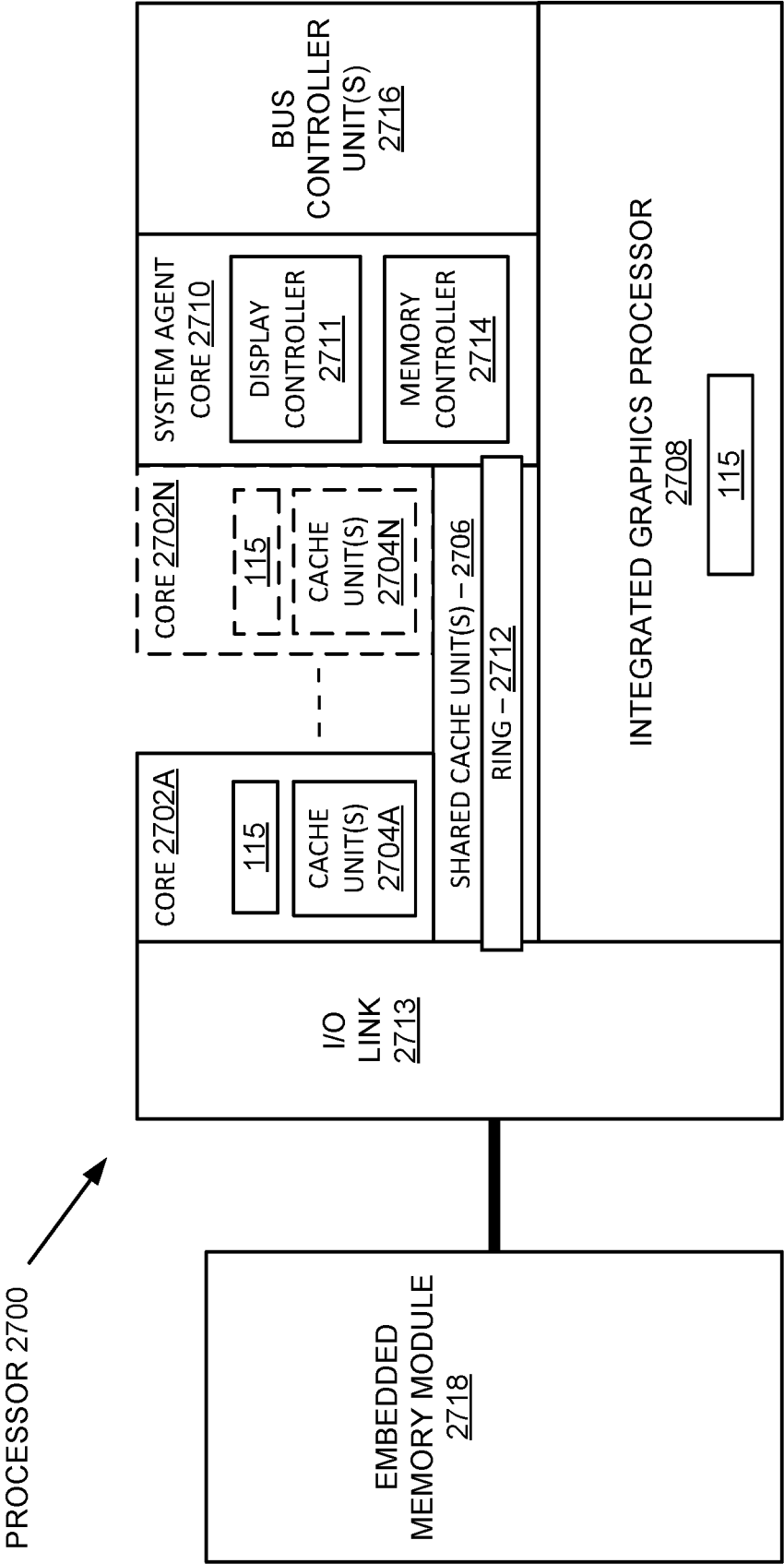
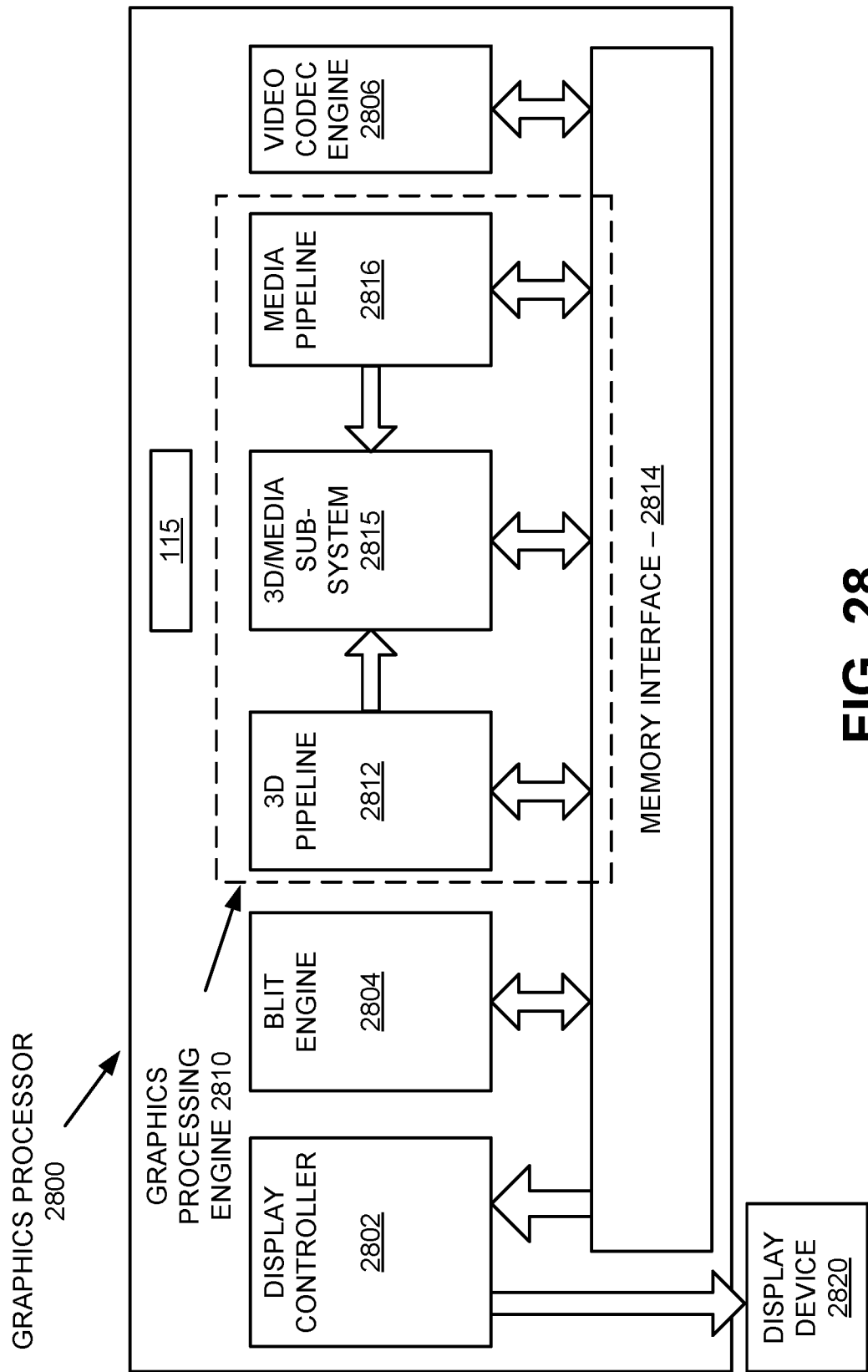


FIG. 27



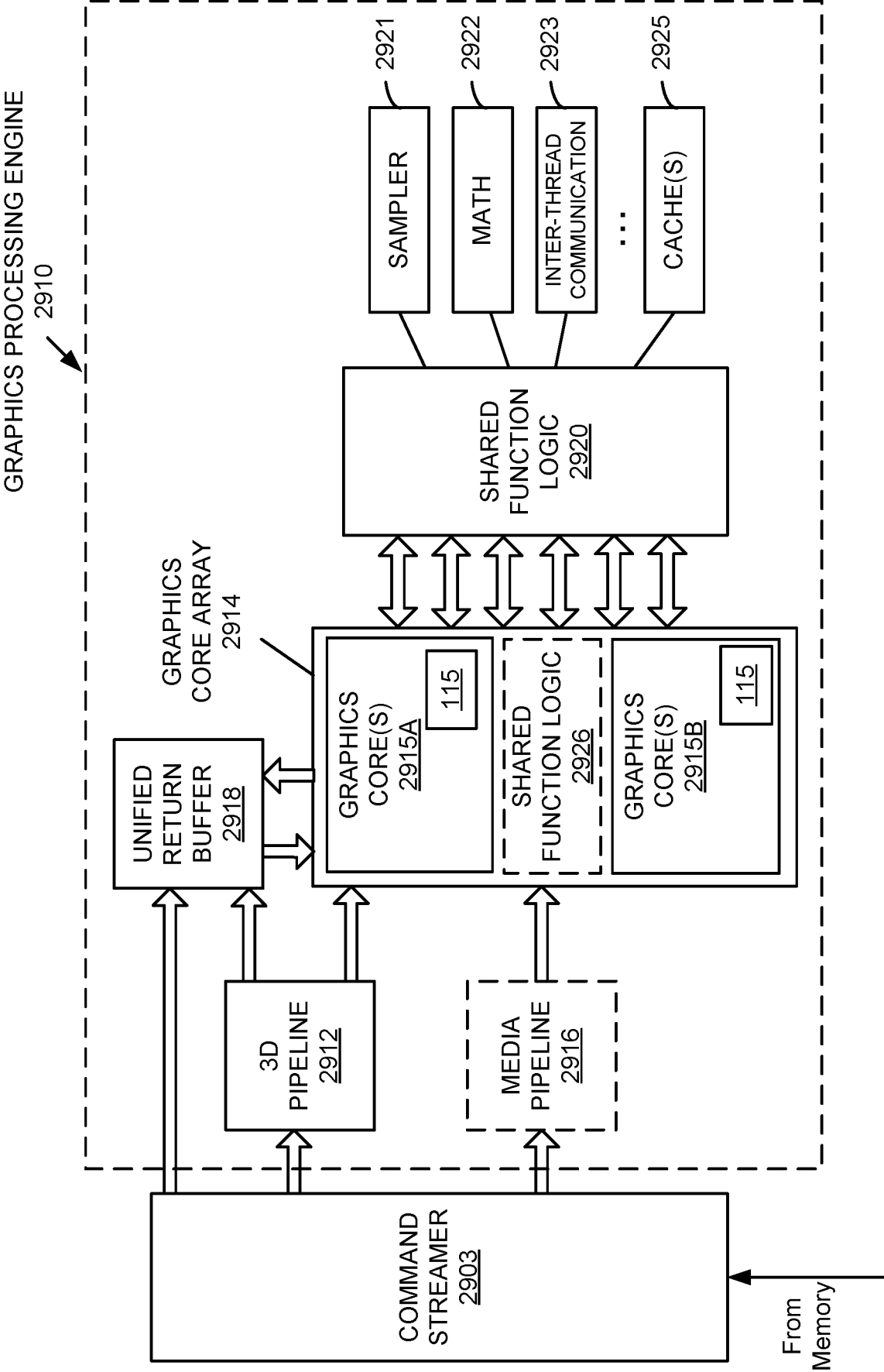


FIG. 29

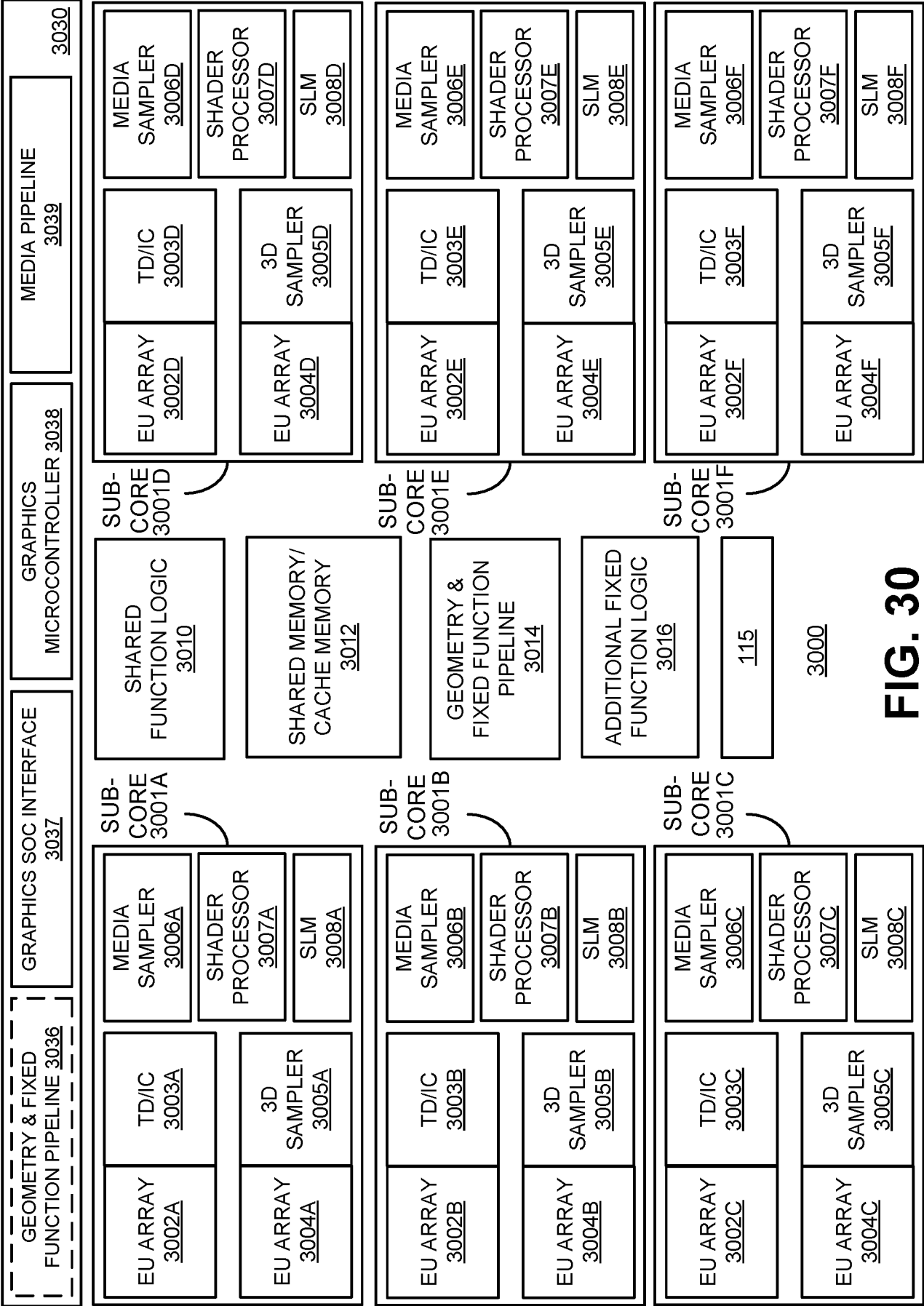


FIG. 30

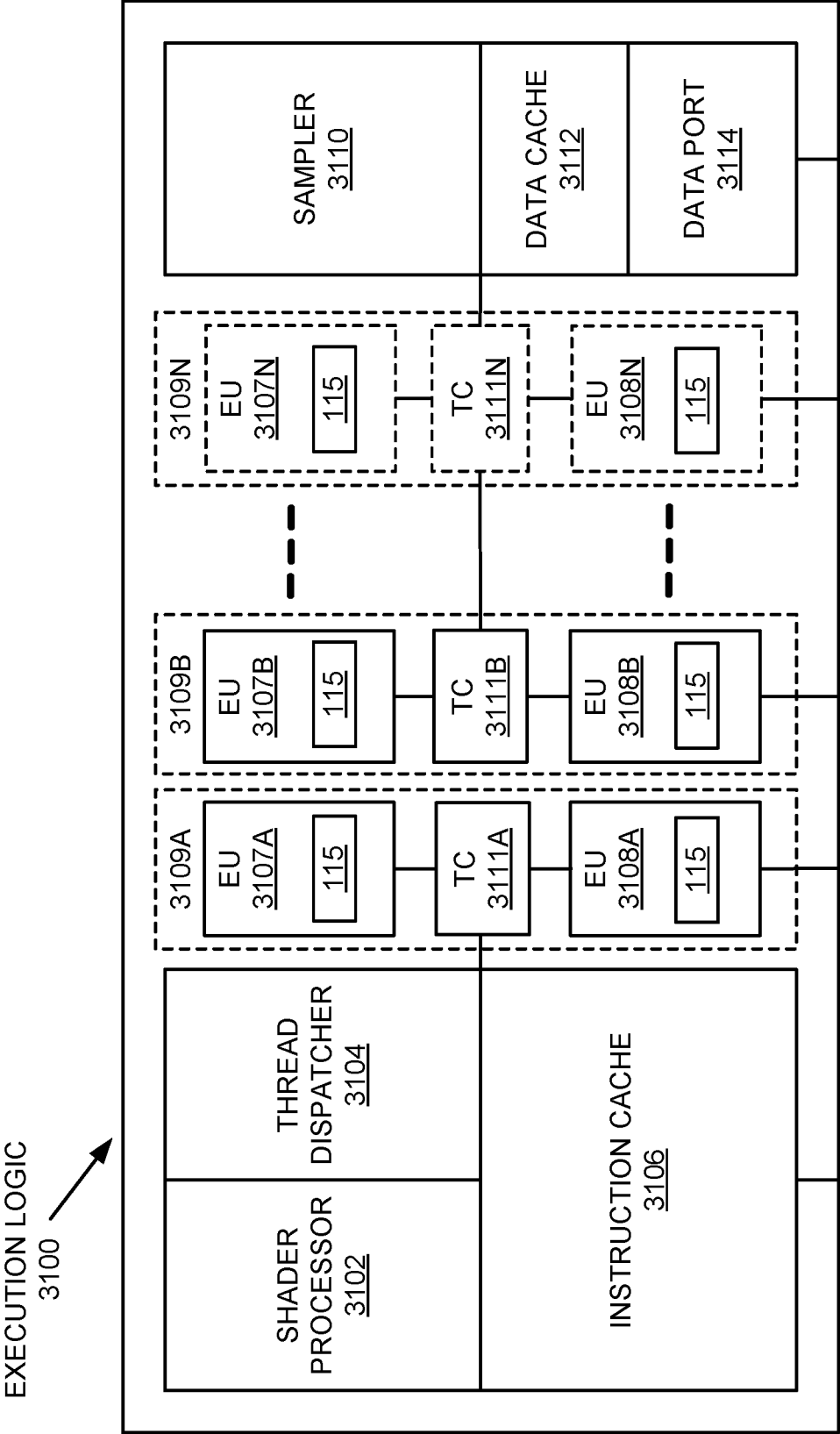


FIG. 31A

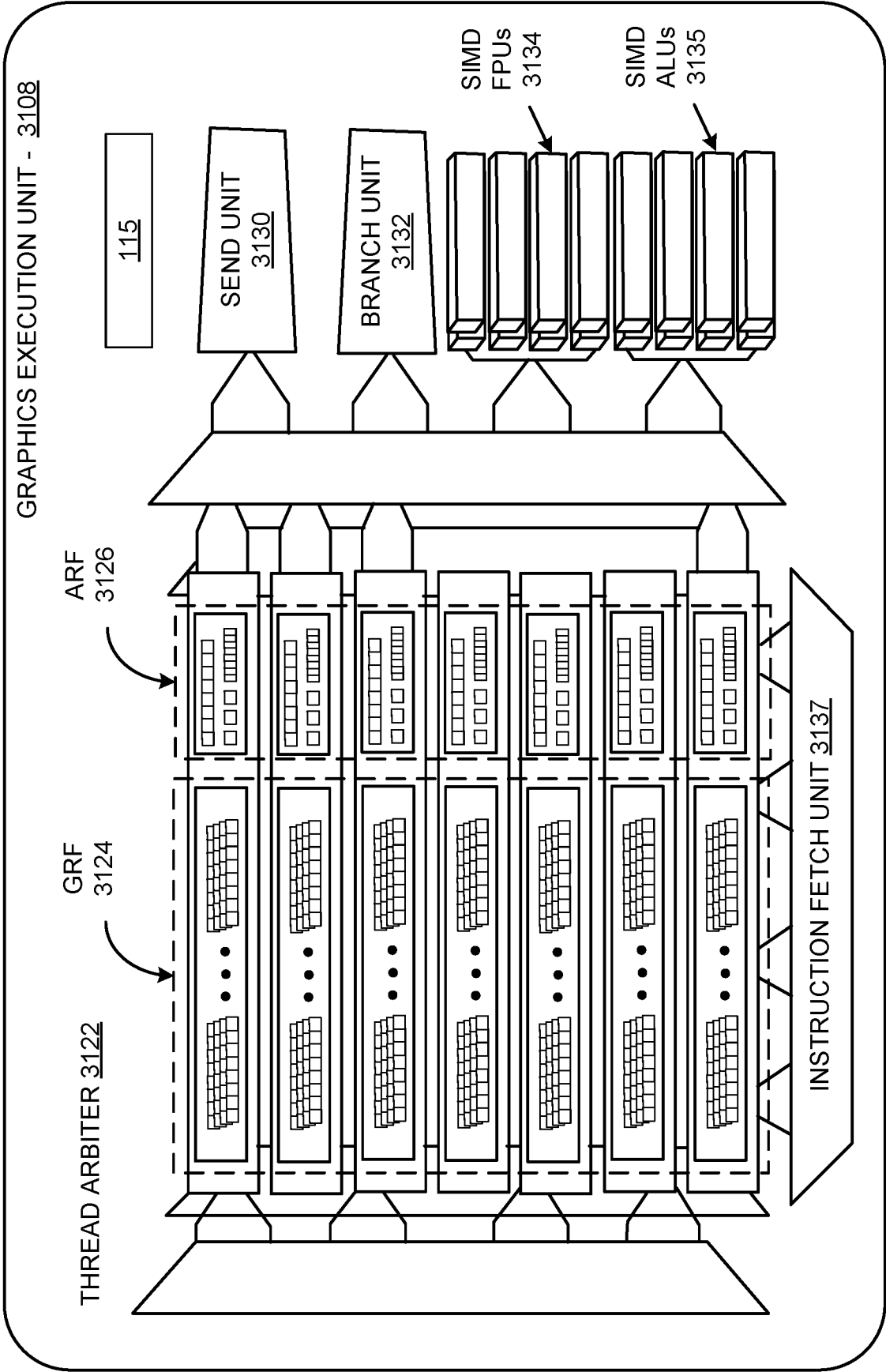
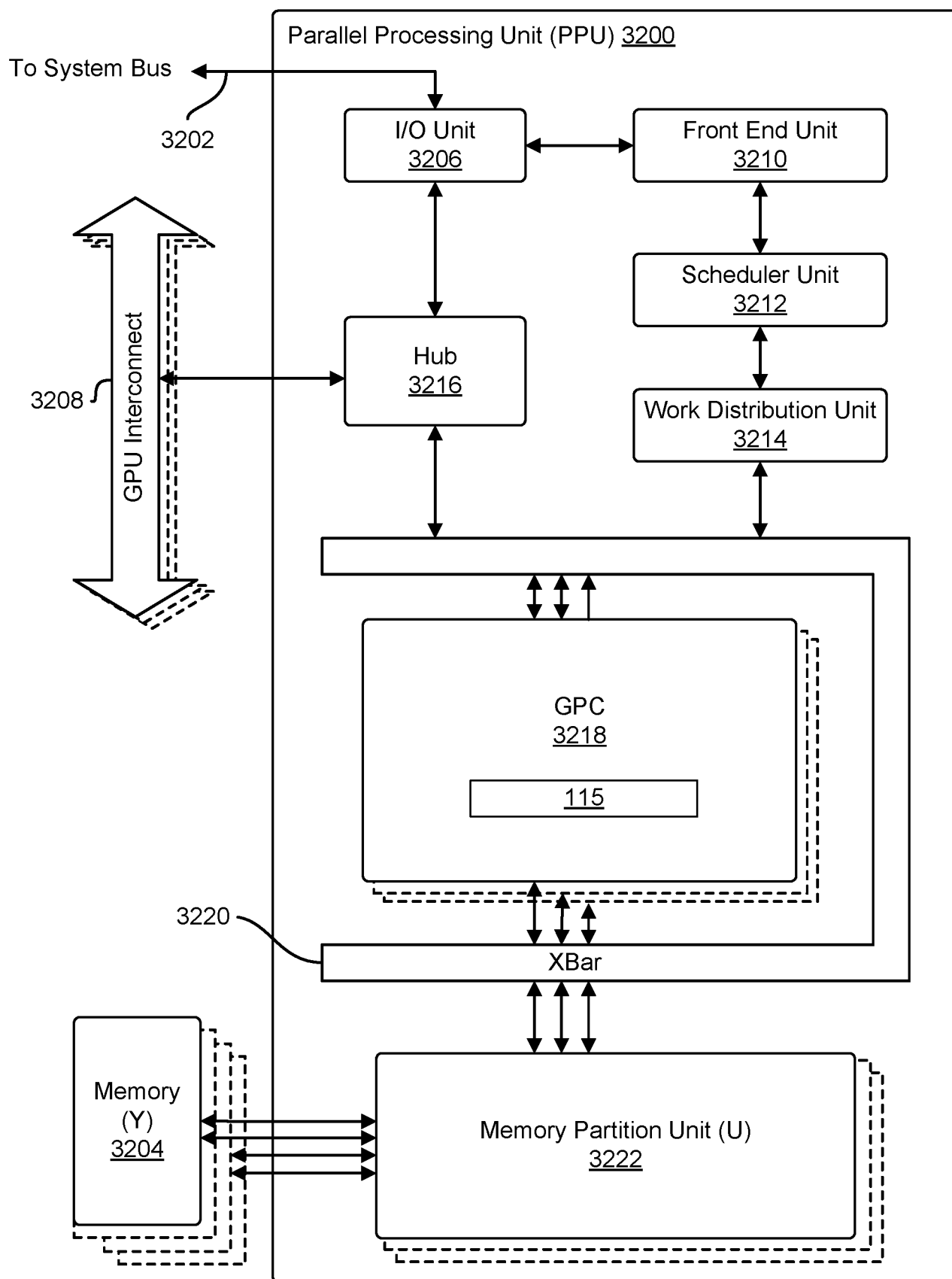
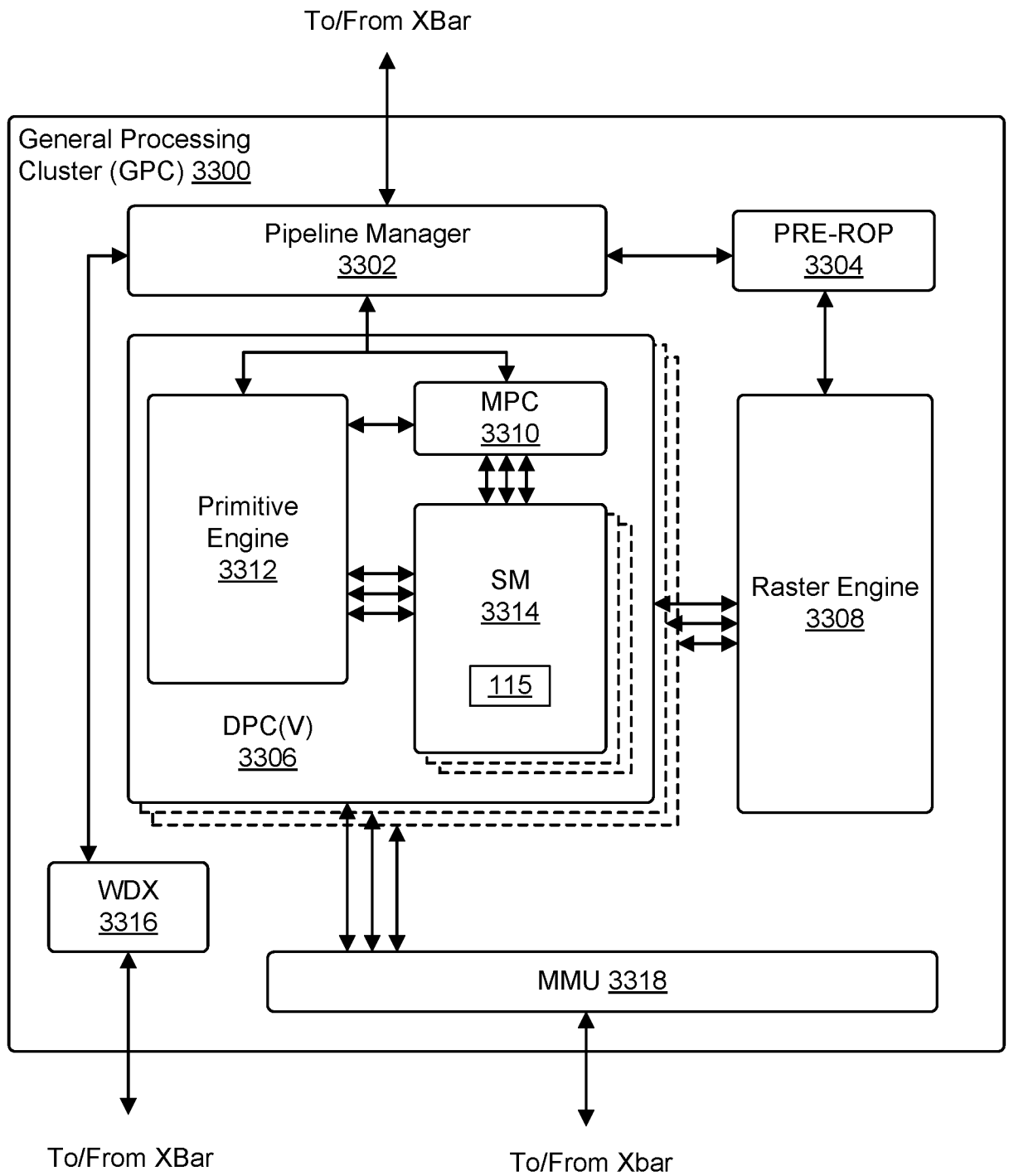
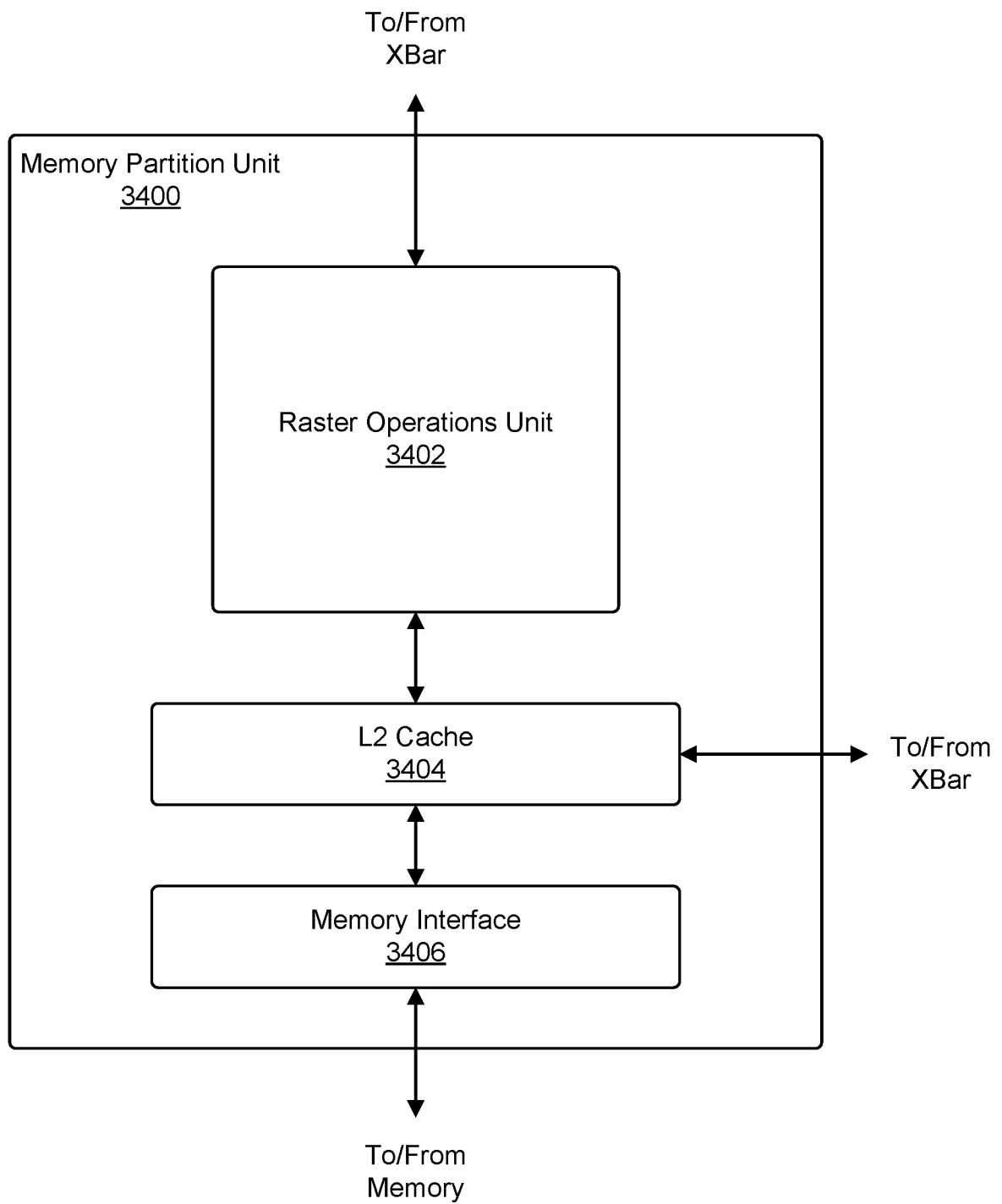
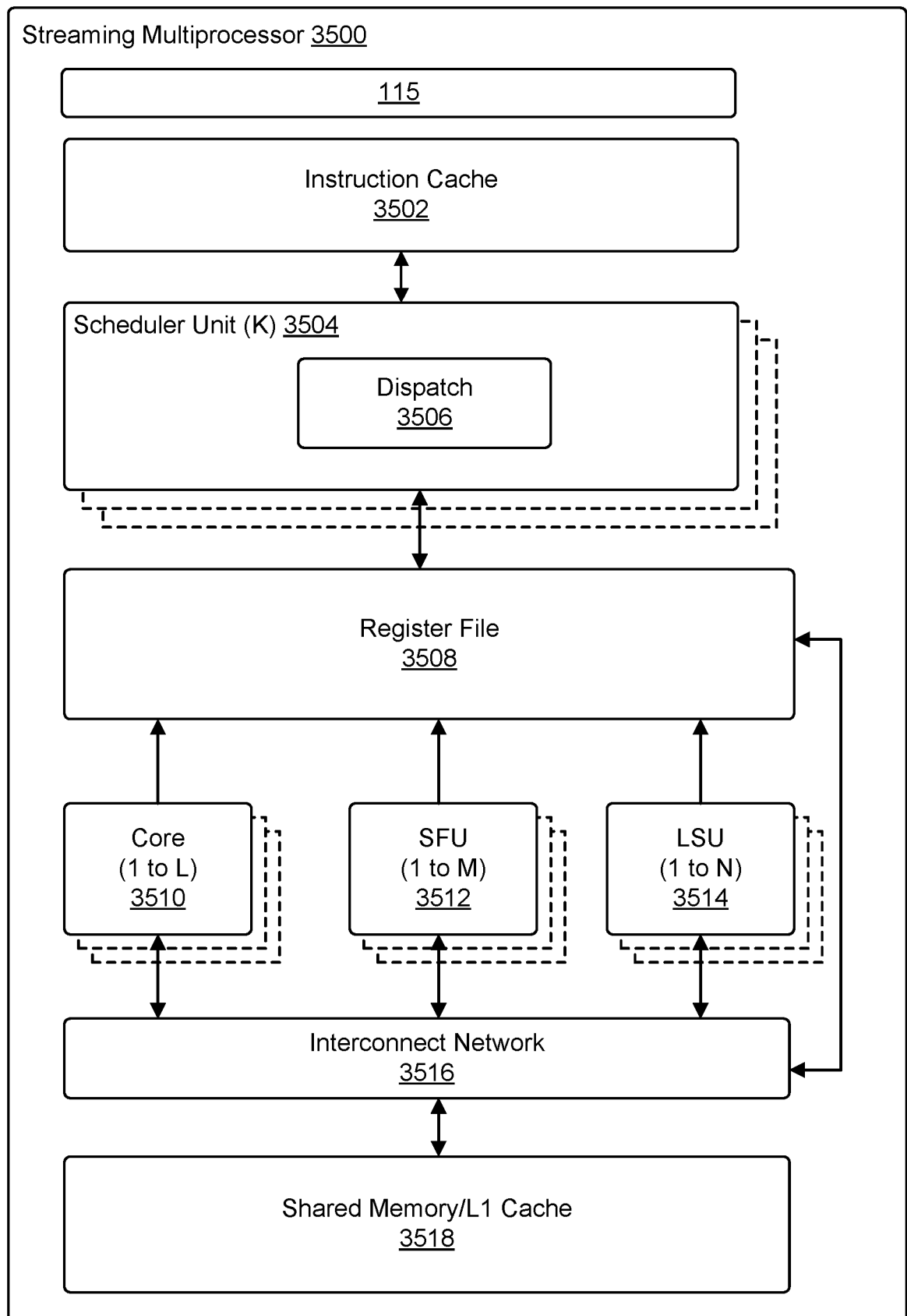


FIG. 31B

**FIG. 32**

**FIG. 33**

**FIG. 34**

**FIG. 35**

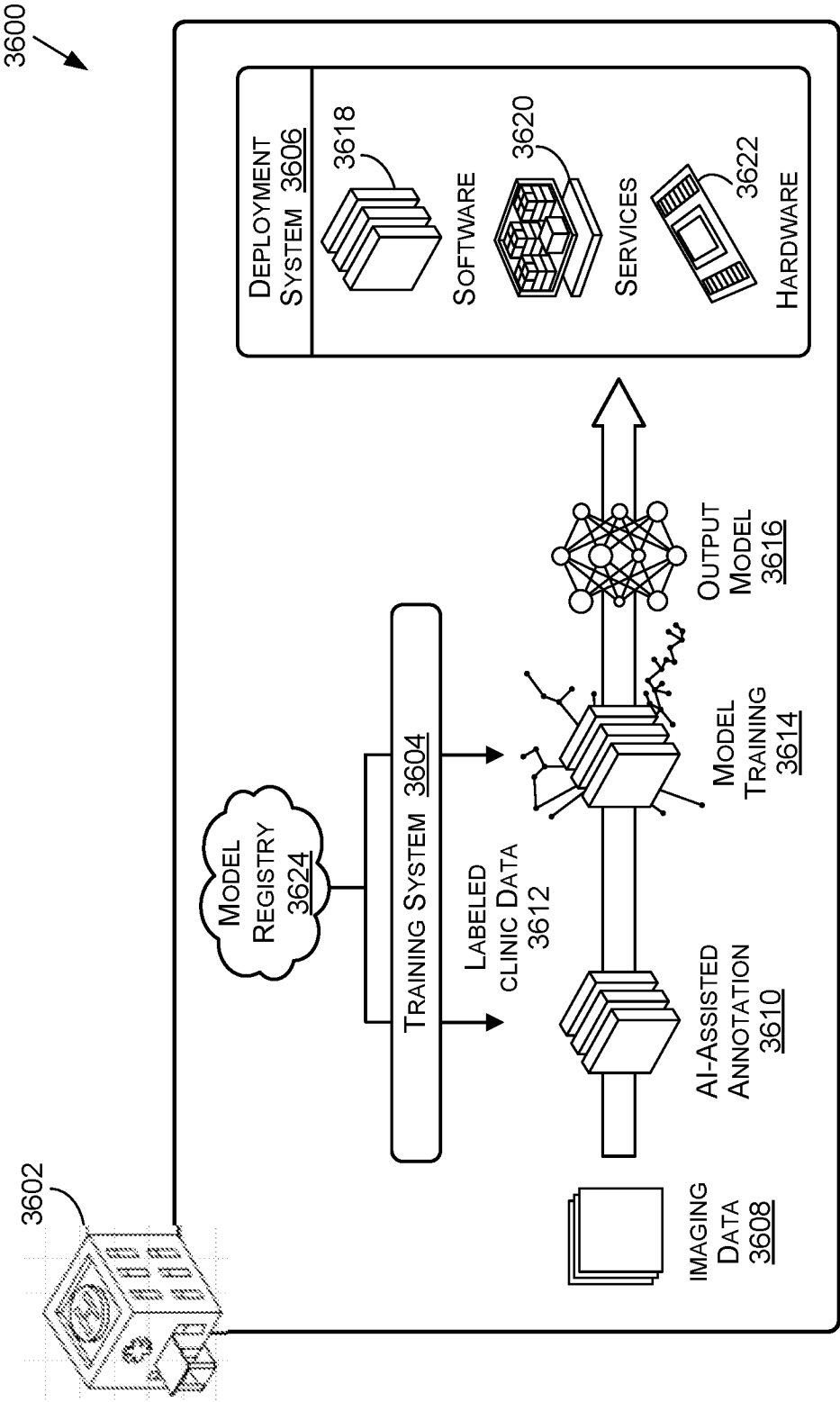


FIG. 36

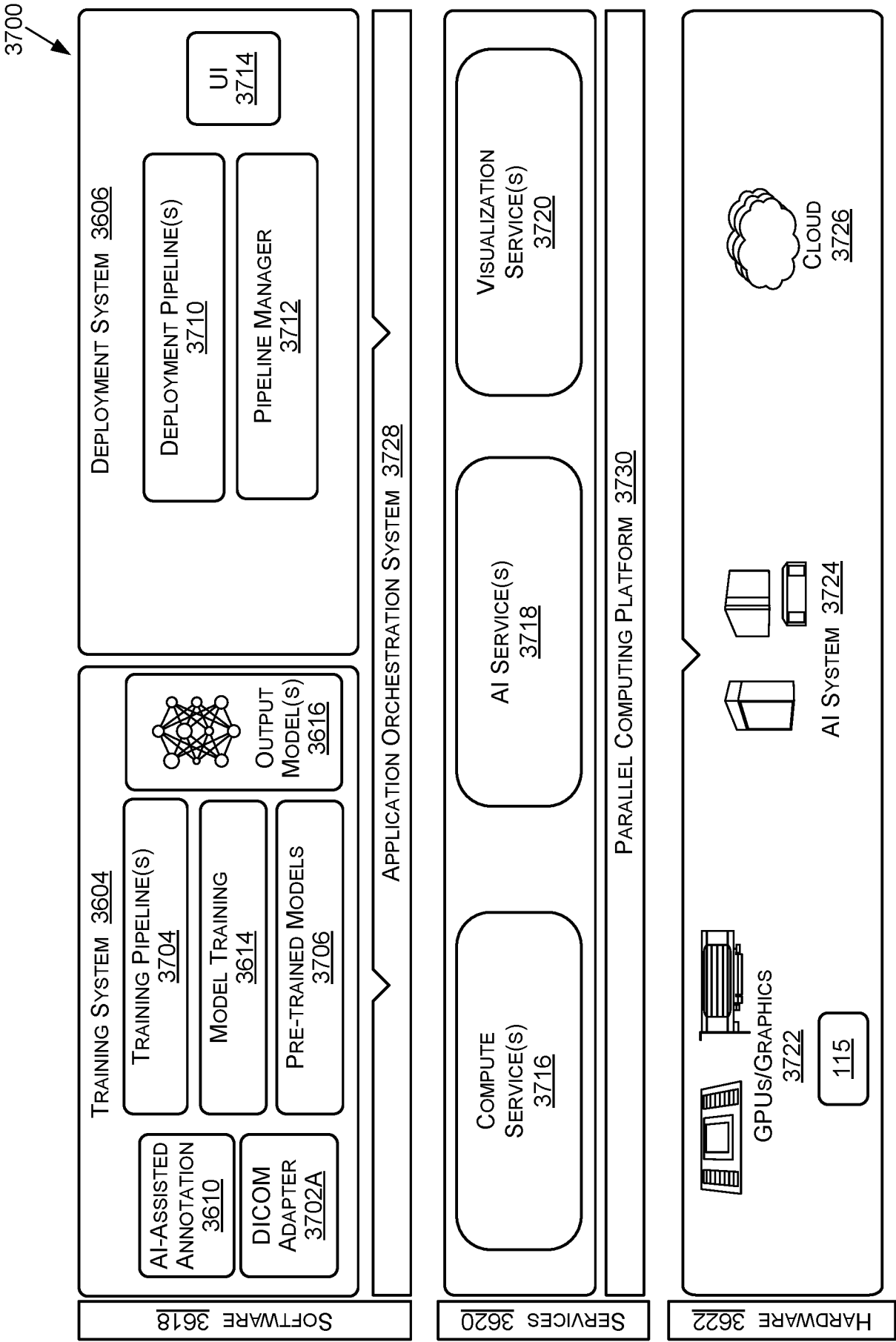


FIG. 37

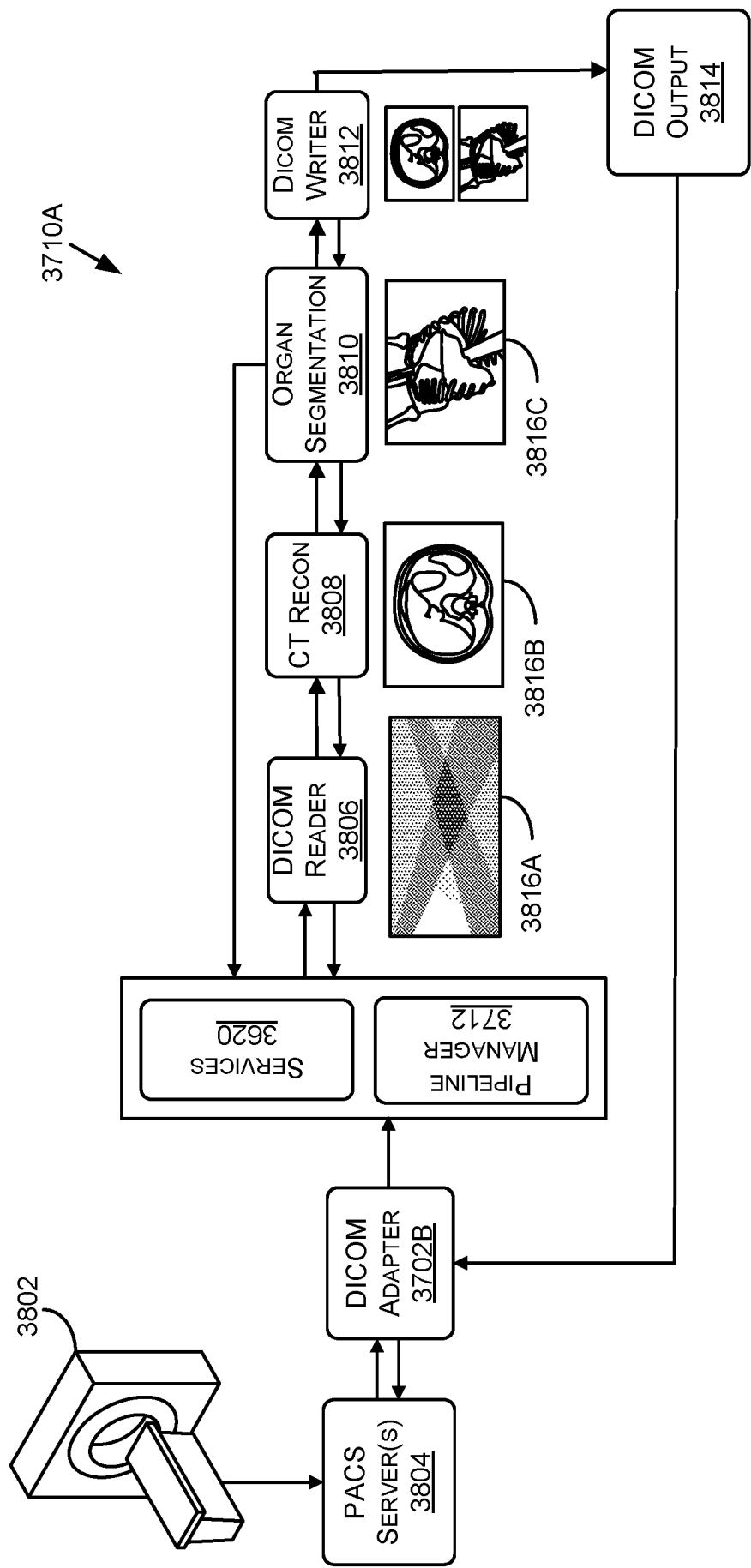


FIG. 38

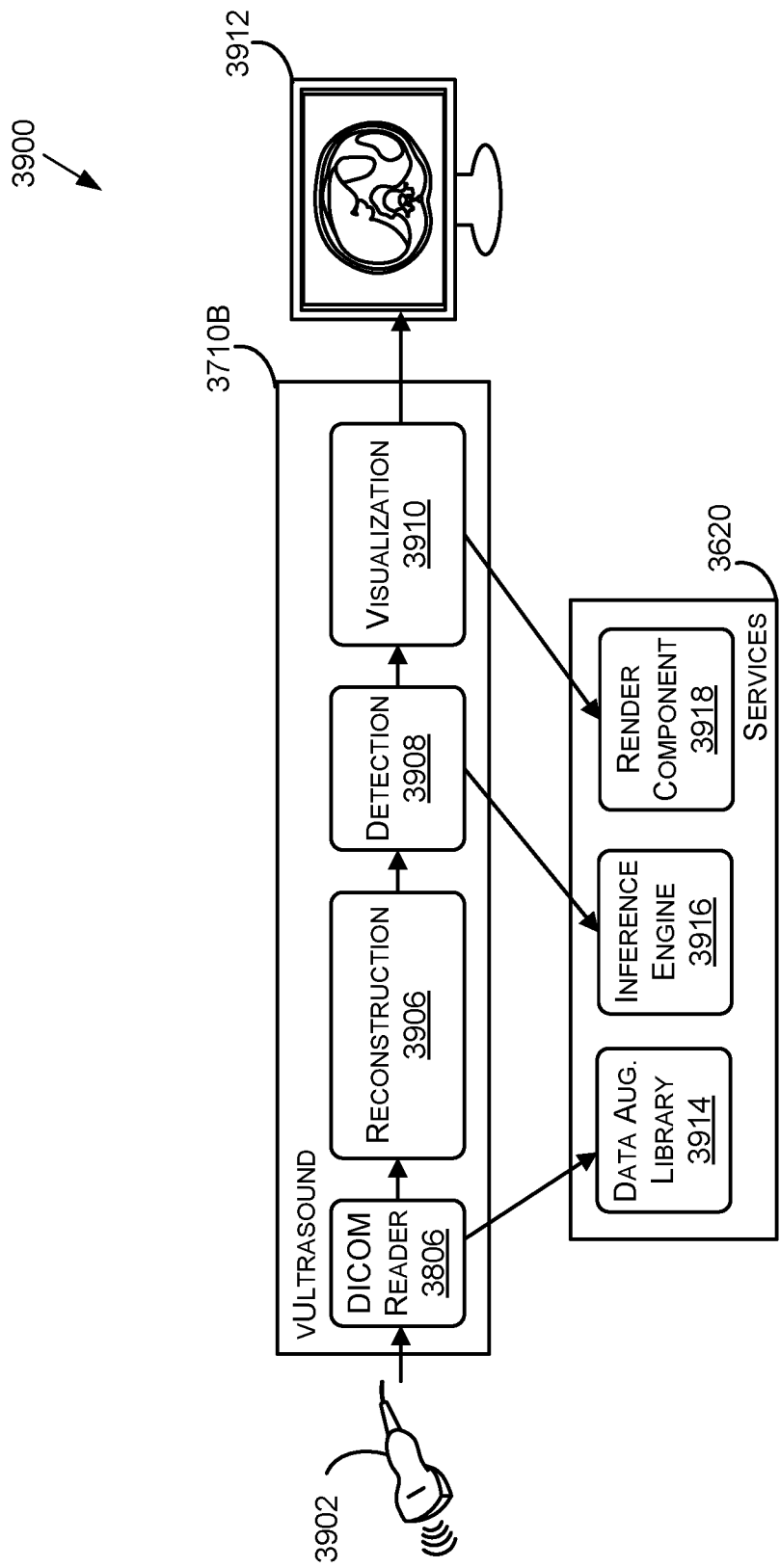


FIG. 39A

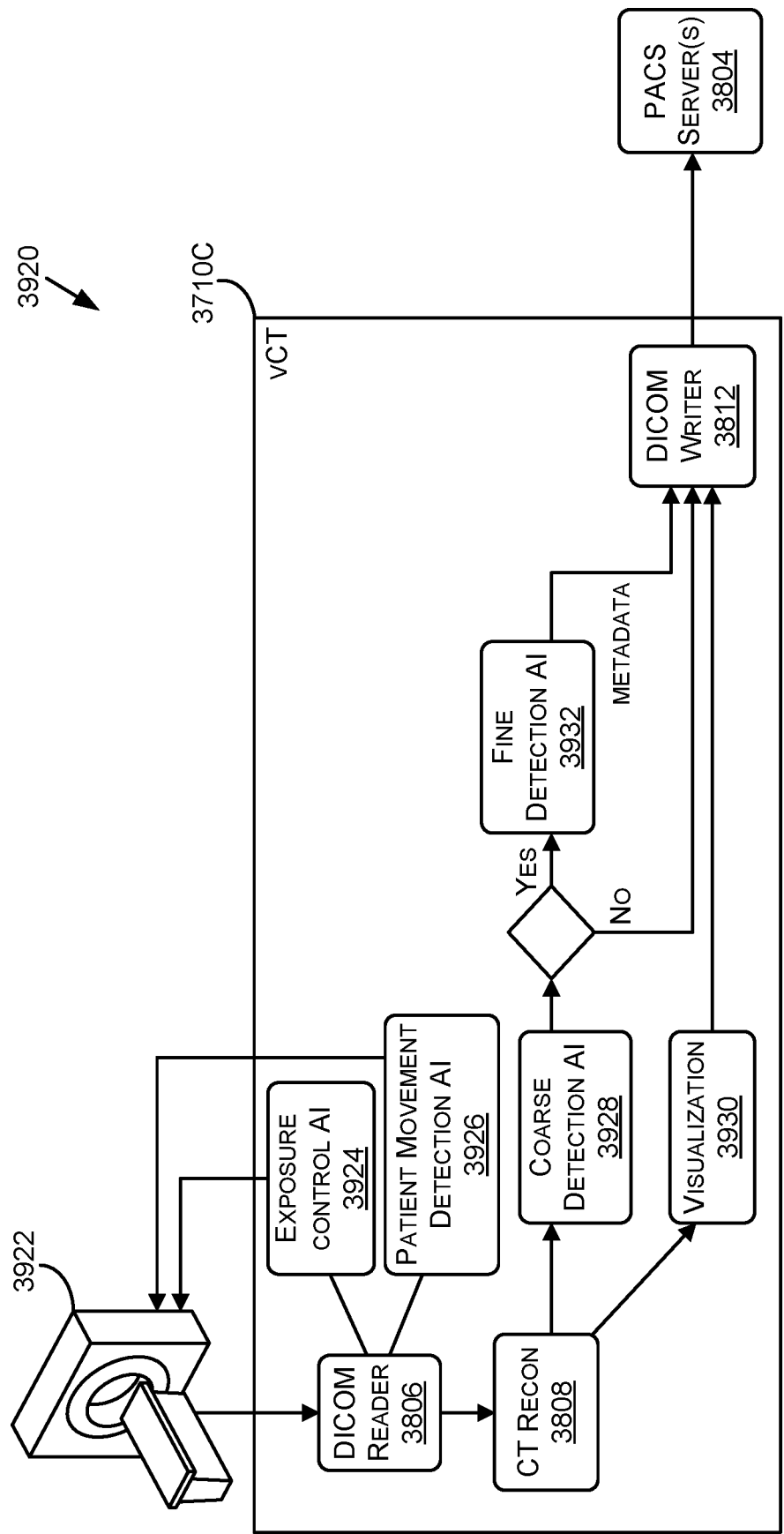


FIG. 39B

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2021/049714

A. CLASSIFICATION OF SUBJECT MATTER

INV. **G06K9/62 G06N3/04**
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)

G06T G06K

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	CHERIAN ANOOP ET AL: "Sem-GAN: Semantically-Consistent Image-to-Image Translation", 2019 IEEE WINTER CONFERENCE ON APPLICATIONS OF COMPUTER VISION (WACV), IEEE, 7 January 2019 (2019-01-07), pages 1797-1806, XP033525786, DOI: 10.1109/WACV.2019.00196	1, 2, 8, 14, 20, 21, 27, 31
Y	Section 1, 3.4 and 3.5; figure 1	3-7, 9-13, 15-19, 22-26, 28-30
	----- -/--	

☒ 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

6 December 2021

Date of mailing of the international search report

04/01/2022

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

Meurisse, Wim

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2021/049714

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	KARRAS TERO ET AL: "A Style-Based Generator Architecture for Generative Adversarial Networks", 2019 IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), IEEE, 15 June 2019 (2019-06-15), pages 4396-4405, XP033686853, DOI: 10.1109/CVPR.2019.00453	3-7, 9-13, 15-19, 22-26, 28-30
A	the whole document	1, 2, 8, 14, 20, 21, 27, 31
A	----- WANG TING-CHUN ET AL: "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs", 2018 IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, IEEE, 18 June 2018 (2018-06-18), pages 8798-8807, XP033473804, DOI: 10.1109/CVPR.2018.00917 Section 1, 3; figure 5	1-31
A	----- LIU SHUANGTING ET AL: "Pixel Level Data Augmentation for Semantic Image Segmentation Using Generative Adversarial Networks", ICASSP 2019 - 2019 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), IEEE, 12 May 2019 (2019-05-12), pages 1902-1906, XP033566165, DOI: 10.1109/ICASSP.2019.8683590 figure 1	1-31
A	----- WO 2018/184187 A1 (INTEL CORP [US]; YAO ANBANG [CN] ET AL.) 11 October 2018 (2018-10-11) paragraphs [0210] - [0221]; figures 15, 15 -----	1-31

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2021/049714

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2018184187 A1	11-10-2018	CN 110352430 A	18-10-2019
		EP 3607492 A1	12-02-2020
		US 2021201078 A1	01-07-2021
		WO 2018184187 A1	11-10-2018
