



(51) International Patent Classification:
G06N 3/063 (2023.01)

(21) International Application Number:
PCT/CN2023/141220

(22) International Filing Date:
22 December 2023 (22.12.2023)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventor: YU, Chong; No. 5709 Shenjiang Road /No. 26
Qiu Yue Road, Floors 1, 2, 3, 5, 6, 7, 8, Pudong New Area,
Shanghai 201210 (CN).

(74) Agent: P. C. & ASSOCIATES; 9th Floor Scitech Tower,
No. 22 Jian Guo Men Wai Avenue, Chaoyang District, Bei-
jing 100022 (CN).

(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, CV, CZ, DE, DJ, DK, DM,
DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT,
HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG,
KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY,
MA, MD, MG, MK, MN, MU, 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, CV,
GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, 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, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE,
SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN,
GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

(54) Title: NEURAL NETWORK MODIFICATION

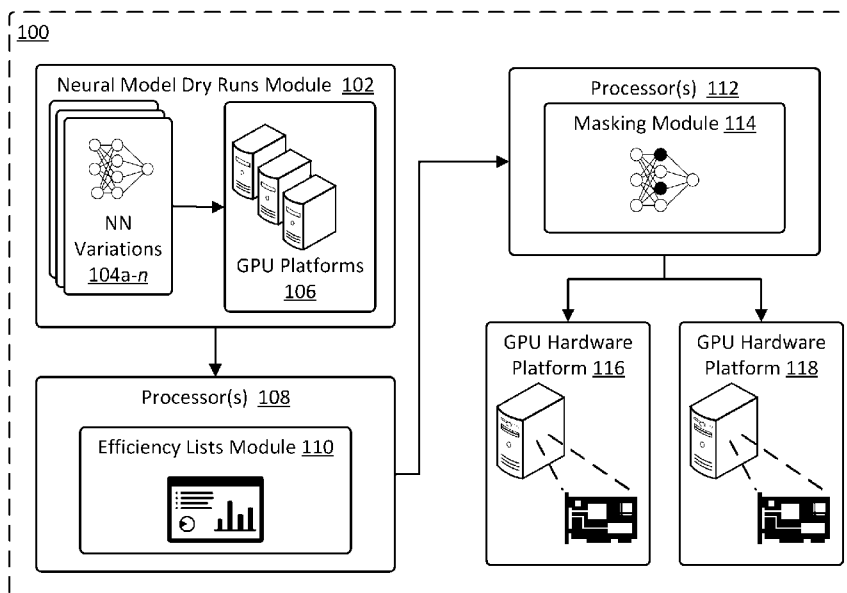


FIG. 1A

(57) Abstract: Apparatuses, systems, and techniques are to modify a neural network according to hardware type. In at least one embodiment, one or more masks are used with a neural network to be deployed on a specific GPU hardware platform.



NEURAL NETWORK MODIFICATION

TECHNICAL FIELD

[0001] At least one embodiment pertains to processing resources used to perform neural network models. For example, a processor comprising one or more circuits are to modify a neural network model to be performed on different hardware resources.

BACKGROUND

[0002] Neural network models can perform differently on different hardware resources. For example, a neural network model may exhibit suboptimal performance on one hardware resource but not on another hardware resource due to different hardware resource constraints. Therefore, improvements can be made to neural network model performance.

BRIEF DESCRIPTION OF DRAWINGS

[0003] FIG. 1A illustrates a block diagram of a system comprising one or more processors to modify a neural network model to be deployed on different hardware resources, according to at least one embodiment;

[0004] FIG. 1B illustrates a database that stores efficiency lists used in selecting masks to insert into a neural network model; according to at least one embodiment;

[0005] FIG. 2 illustrates a block diagram of modifications made to a residual block of a neural network model to be used, in part, to generate efficiency lists, according to at least one embodiment;

[0006] FIG. 3 illustrates a block diagram of modifications made to a self-attention block of a neural network model to be used, in part, to generate efficiency lists, according to at least one embodiment;

[0007] FIG. 4 illustrates a block diagram of a neural network model dry runs system, according to at least one embodiment;

[0008] FIG. 5 illustrates a block diagram of a neural network model masking system, according to at least one embodiment;

[0009] FIG. 6 illustrates a block diagram of an efficient neural network model deployment system, according to at least one embodiment;

[0010] FIG. 7 illustrates a process of modifying a neural network model to achieve cross-GPU-platform efficiency, according to at least one embodiment;

[0011] FIG. 8A illustrates a block diagram of a driver and/or runtime used to achieve cross-GPU-platform efficiency, according to at least one embodiment;

[0012] FIG. 8B illustrates a block diagram of a processor and various modules used to achieve cross-GPU-platform efficiency, according to at least one embodiment;

[0013] FIG. 9A illustrates logic, according to at least one embodiment;

[0014] FIG. 9B illustrates logic, according to at least one embodiment;

[0015] FIG. 10 illustrates training and deployment of a neural network model, according to at least one embodiment;

[0016] FIG. 11 illustrates an example data center system, according to at least one embodiment;

[0017] FIG. 12A illustrates an example of an autonomous vehicle, according to at least one embodiment;

[0018] FIG. 12B illustrates an example of camera locations and fields of view for the autonomous vehicle of FIG. 12A, according to at least one embodiment;

[0019] FIG. 12C is a block diagram illustrating an example system architecture for the autonomous vehicle of FIG. 12A, according to at least one embodiment;

[0020] FIG. 12D is a diagram illustrating a system for communication between cloud-based server(s) and the autonomous vehicle of FIG. 12A, according to at least one embodiment;

[0021] FIG. 13 is a block diagram illustrating a computer system, according to at least one embodiment;

[0022] FIG. 14 is a block diagram illustrating a computer system, according to at least one embodiment;

[0023] FIG. 15 illustrates a computer system, according to at least one embodiment;

- [0024] FIG. 16 illustrates a computer system, according to at least one embodiment;
- [0025] FIG. 17A illustrates a computer system, according to at least one embodiment;
- [0026] FIG. 17B illustrates a computer system, according to at least one embodiment;
- [0027] FIG. 17C illustrates a computer system, according to at least one embodiment;
- [0028] FIG. 17D illustrates a computer system, according to at least one embodiment;
- [0029] FIGS. 17E and 17F illustrate a shared programming model, according to at least one embodiment;
- [0030] FIG. 18 illustrates exemplary integrated circuits and associated graphics processors, according to at least one embodiment;
- [0031] FIGS. 19A-19B illustrate exemplary integrated circuits and associated graphics processors, according to at least one embodiment;
- [0032] FIGS. 20A-20B illustrate additional exemplary graphics processor logic according to at least one embodiment;
- [0033] FIG. 21 illustrates a computer system, according to at least one embodiment;
- [0034] FIG. 22A illustrates a parallel processor, according to at least one embodiment;
- [0035] FIG. 22B illustrates a partition unit, according to at least one embodiment;
- [0036] FIG. 22C illustrates a processing cluster, according to at least one embodiment;
- [0037] FIG. 22D illustrates a graphics multiprocessor, according to at least one embodiment;
- [0038] FIG. 23 illustrates a multi-graphics processing unit (GPU) system, according to at least one embodiment;
- [0039] FIG. 24 illustrates a graphics processor, according to at least one embodiment;
- [0040] FIG. 25 is a block diagram illustrating a processor micro-architecture for a processor, according to at least one embodiment;
- [0041] FIG. 26 illustrates a deep learning application processor, according to at least one embodiment;

[0042] FIG. 27 is a block diagram illustrating an example neuromorphic processor, according to at least one embodiment;

[0043] FIG. 28 illustrates at least portions of a graphics processor, according to one or more embodiments;

[0044] FIG. 29 illustrates at least portions of a graphics processor, according to one or more embodiments;

[0045] FIG. 30 illustrates at least portions of a graphics processor, according to one or more embodiments;

[0046] FIG. 31 is a block diagram of a graphics processing engine of a graphics processor in accordance with at least one embodiment;

[0047] FIG. 32 is a block diagram of at least portions of a graphics processor core, according to at least one embodiment;

[0048] FIGS. 33A-33B illustrate thread execution logic including an array of processing elements of a graphics processor core according to at least one embodiment;

[0049] FIG. 34 illustrates a parallel processing unit ("PPU"), according to at least one embodiment;

[0050] FIG. 35 illustrates a general processing cluster ("GPC"), according to at least one embodiment;

[0051] FIG. 36 illustrates a memory partition unit of a parallel processing unit ("PPU"), according to at least one embodiment;

[0052] FIG. 37 illustrates a streaming multi-processor, according to at least one embodiment;

[0053] FIG. 38 is an example data flow diagram for an advanced computing pipeline, in accordance with at least one embodiment;

[0054] FIG. 39 is a system diagram for an example system for training, adapting, instantiating and deploying machine learning models in an advanced computing pipeline, in accordance with at least one embodiment;

[0055] FIG. 40 includes an example illustration of an advanced computing pipeline 3910A for processing imaging data, in accordance with at least one embodiment;

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

[0057] FIG. 41B includes an example data flow diagram of a virtual instrument supporting an CT scanner, in accordance with at least one embodiment;

[0058] FIG. 42A illustrates a data flow diagram for a process to train a machine learning model, in accordance with at least one embodiment;

[0059] FIG. 42B is an example illustration of a client-server architecture to enhance annotation tools with pre-trained annotation models, in accordance with at least one embodiment; and

[0060] FIG. 43 illustrates components of a system to access a large language model, according to at least one embodiment.

DETAILED DESCRIPTION

[0061] In the following description, numerous specific details are set forth to provide a more thorough understanding of at least one embodiment. However, it will be apparent to one skilled in the art that the inventive concepts may be practiced without one or more of these specific details, and that two or more aspects of any one or more embodiments described herein may be combined.

[0062] In at least one embodiment, a processor comprising one or more circuits are to select one or more masks (described further herein at least in conjunction with FIG. 5) to be used with a neural network model based, at least in part, on one or more hardware resources that are to perform that neural network model. In at least one embodiment, a processor comprising one or more circuits are to modify a neural network model to be performed by a specific hardware resource based, at least in part, on a previous performance of that neural network model on a different hardware resource. In at least one embodiment, a processor comprising one or more circuits are to modify a neural network model such that different graphics processing unit (GPU) platforms can perform that modified neural network model to meet and/or exceed one or more threshold levels of performance, such as levels of inferencing accuracy and/or inferencing speed.

In at least one embodiment, a processor comprising one or more circuits are used to modify a neural network model to be performed on a specific GPU hardware platform reduces or eliminates training time of that neural network model normally required without that modification. In at least one embodiment, a processor comprising one or more circuits are used to modify a neural network model so that one or more specific (GPU) platforms perform that modified neural network model to meet and/or exceed one or more threshold levels of performance is referred to as a neural network model optimization method to achieve cross-GPU-platform efficiency. In at least one embodiment, a processor comprising one or more circuits are to collect performance metrics related to different variations of an efficient neural network model performed by different hardware platforms so that those performance metrics can be used to inform how to make a neural model efficient and perform well on a specific hardware platform, while minimizing and/or eliminating a need to train that efficient neural model prior to deploying it on that specific hardware platform.

[0063] In at least one embodiment, a processor comprising one or more circuits are to modify a neural network model includes generating different variations of a neural network model by quantizing (converting data formats) and/or pruning (removing components) different layers of that neural network model with each variation, which is described further herein. In at least one embodiment, dry runs (e.g., practice runs) of different variations of a neural network model are performed by different GPU hardware platforms and performance metrics are collected as a result of those dry runs. In at least one embodiment, performance metrics generated by dry runs of different variations of a neural network model are used to insert data conversion masks and sparse pruning masks into a trained, dense neural network model to create a masked super neural network model. In at least one embodiment, a masked super neural network model is further trained using input data to improve that masked super neural network model's performance in terms of inferencing accuracy. In at least one embodiment, inferencing accuracy is referred to as accuracy. In at least one embodiment, a masked super neural network model is referred to as a masked super neural model.

[0064] In at least one embodiment, a masked super neural network model is used as a base neural network model to be deployed on different GPU hardware platforms. In at least one embodiment, a masked super neural network model is further modified based on performance

metrics of dry runs of different variations of neural network models performed by a specific type of GPU hardware platform that is also to perform that masked super neural network model. In at least one embodiment, a masked super neural network model that has been modified based on a specific type of GPU hardware is referred to as an efficient neural network model for deployment. In at least one embodiment, an efficient neural network model for deployment is modified by using one or more techniques of knowledge distillation, which is described further herein.

[0065] FIG. 1A illustrates, as a block diagram, a system 100 comprising one or more processors used to modify a neural network model so that different hardware resources perform that modified neural network model to meet and/or exceed one or more threshold levels of performance. In at least one embodiment, a hardware resource is an artificial intelligence (AI) hardware platform and/or a graphics processing unit (GPU) hardware platform. In at least one embodiment, an AI hardware platform and/or a GPU hardware platform is a system of one or more GPUs, one or more central processing units (CPUs), one or more deep learning (DL) accelerators, one or more memory components, firmware, software, or some combination thereof, configured to perform neural network model training and/or inferencing. In at least one embodiment, a GPU hardware platform is an GPU hardware platform that uses one or more specific types of GPUs. In at least one embodiment, for example, a GPU hardware platform is an NVIDIA® Jetson AGX Orin™ Developer Kit AI Edge Computer or an AMD® Instinct™ MI300 accelerator. In at least one embodiment, a hardware platform is referred to as a system.

[0066] In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 1A are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 2-8B. In at least one embodiment, one or more processors perform one or more operations used as part of system 100. In at least one embodiment, one or more processors that perform one or more operations used as part of system 100 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 performs an operation used by masking module

114, such as inserting masks into a neural network model. In at least one embodiment, processor(s) 822 performs an operation used by efficiency lists module 110, such as fusing efficiency lists, which is described further herein.

[0067] In at least one embodiment, as used in any implementation described herein, unless otherwise clear from context or stated explicitly to contrary, terms such as “system” and “module,” and nominalized verbs (e.g., compiler, and/or other terms) each refers to any combination of software logic, firmware logic, hardware logic, and/or circuitry configured to provide functionality described herein. In at least one embodiment, any combination of software logic, firmware logic, hardware logic, and/or circuitry configured to provide functionality described herein is referred to as a component. In at least one embodiment, any component described herein are combined and/or communicatively connected with at least one other component, regardless of how such components are described to be combined and/or communicatively connected in other embodiments. In at least one embodiment, software may be embodied as a software package, code, and/or instruction set or instructions. In at least one embodiment, hardware includes, singly or in any combination, hardwired circuitry, programmable circuitry, state machine circuitry, fixed function circuitry, execution unit circuitry, and/or firmware that stores instructions executed by programmable circuitry. In at least one embodiment, modules may, collectively or individually, be embodied as circuitry that forms part of a larger system, for example, an integrated circuit (IC), system on-chip (SoC), and so forth.

[0068] In at least one embodiment, system 100 as depicted in FIG. 1A provides an overview of a system comprising one or more processors used to modify a neural network model so that different hardware resources perform that modified neural network model to meet and/or exceed one or more threshold levels of performance. In at least one embodiment, system 100 is referred to as a process, method, system, algorithm, series of operations, series of tasks, or some combination thereof.

[0069] In at least one embodiment, a neural network model is referred to as a neural network or a neural model. In at least one embodiment, a neural model includes one or more neural models. In at least one embodiment, a neural model is a deep learning network or deep neural network (DNN). In at least one embodiment, a neural model includes any neural model discussed herein, including those discussed in conjunction with FIGS. 9 and 10C. In at least one

embodiment, a portion of a neural model is a component of a neural model. In at least one embodiment, a portion of a neural model is a layer, a node, a function, a value, or some combination thereof. In at least one embodiment, a neural model includes processors, memory, functions, and neural model parameters used to make inferences based on input data. In at least one embodiment, neural model parameters include values that affect inferencing performance of a neural model, values such as weights, coefficients, biases, or some combination thereof. In at least one embodiment, values are referred to as data values. In at least one embodiment, neural model parameters are regularly modified based on a reward generated during a deep learning reinforcement neural model training process. In at least one embodiment, neural model parameters include parameters that indicate learning rates of a neural model, local iterations of a neural model, aggregation weights of a neural model, a number of neurons of neural model, or some combination thereof.

[0070] In at least one embodiment, a neural model is a recurrent neural model (RNN), a convolutional neural model (CNN), generative adversarial network (GAN), transformer, graph neural model (GNN), or some combination thereof. In at least one embodiment, a neural model is a CNN such as U-Net with a 5-level encoder-decoder network architecture with residual blocks at every level. In at least one embodiment, a residual block is a stack of neural model layers, where an output of a layer is added to another layer deeper in that residual block. In at least one embodiment, a layer is a structure or network topology that contains nodes corresponding to a feature extracted from a dataset. In at least one embodiment, an encoder-decoder network includes two recurrent neural models, one to encode input data and another to decode encoded data. In at least one embodiment, a recurrent network is a neural model that uses sequential data. In at least one embodiment, a training system uses training data to train an untrained neural model to generate a trained neural model, using systems and methods such as those described herein. In at least one embodiment, an untrained neural model is a neural model which has been partially trained, and for which additional training is to occur. In at least one embodiment, training data is a training dataset. In at least one embodiment, a trained neural model is a neural model that has undergone a process of learning from a dataset by, at least in part, adjusting weights of that neural model in order to cause that neural model to perform inferences on previously unseen input data at a level acceptable to a user and/or an application.

[0071] In at least one embodiment, a neural model, such as efficient neural model for deployment 660, is applied in one or more fields, such as deep learning super sampling (DLSS), autonomous vehicle navigation, medical segmentation, object classification, natural language processing (NLP), physics modeling, image processing, financial services, robotics, or some combination thereof.

[0072] In at least one embodiment, a dense neural model is a neural model where each neuron in a layer is connected to every neuron in a preceding layer. In at least one embodiment, a first neuron in one layer is considered to be connected to a second neuron in a preceding layer when an output of that second neuron is used as an input to that first neuron. In at least one embodiment, a neuron (node) is a portion of a layer in a neural model. In at least one embodiment, a neuron includes a function that outputs a weighted sum of input values multiplied by corresponding weights and sends that weighted sum as input to an activation function.

[0073] In at least one embodiment, an efficient neural model is any neural model that has been modified to reduce computational complexity and/or memory requirements of that neural model. In at least one embodiment, an efficient neural model is referred to as a compressed, optimized, sparse, or quantized neural model. In at least one embodiment, an efficient neural model that has been created based on a dense neural model is referred to as an efficient variation of a dense neural model. In at least one embodiment, a modification to a neural model refers to any change, removal, insertion, alteration, or some combination thereof, of one or more portions of a neural model. In at least one embodiment, an efficient neural model is a sparse neural model. In at least one embodiment, a sparse neural model is a neural model where at least a portion of neural model, such as a set of neurons of a layer, is only connected to a subset of neurons in a previous layer. In at least one embodiment, a sparse layer is a layer of a neural model where at least one or more values of that layer are zero. In at least one embodiment, a sparse neural model or sparse layer is referred to as exhibiting sparsity. In at least one embodiment, a sparse neural model is a neural model that includes at least one sparse layer, which is a layer in which at least one of its values has been changed to zero. In at least one embodiment, a sparse neural model is generated by changing one or more neural model values, such as weights, to zero. In at least one embodiment, a sparse neural model is generated by modifying a neural model's architecture, such as by removing a layer of a neural model. In at least one embodiment, a sparse neural model

is generated by using an activation function that increase chances that an output of that function is zero. In at least one embodiment, a sparse neural model is generated when a number of features in its feature maps, such as those in convolutional neural models (CNNs), is reduced. In at least one embodiment, changing one or more neural model weights to zero is referred to as pruning. In at least one embodiment, any modifications used to reduce computational complexity and/or memory requirements of that neural model is referred to pruning. In at least one embodiment, to prune a layer of a neural model is to change one or more neural model weights to zero. In at least one embodiment, a neural model that has been modified to reduce computational complexity and/or memory requirements of that neural model is referred to as a compressed neural model. In at least one embodiment, a neural model that has been modified to reduce computational complexity and/or memory requirements of that neural model is referred to as an efficient neural model. In at least one embodiment, any technique used to modify a neural model to reduce computational complexity and/or memory requirements of that neural model is referred to as a neural model compression technique, efficiency technique, or optimization technique.

[0074] In at least one embodiment, an efficient neural model is a quantized neural model. In at least one embodiment, a quantized neural model is a neural model that includes at least one quantized value. In at least one embodiment, quantization is a conversion of a value from one data format to another data format. In at least one embodiment, quantization includes operations to convert a value from a higher-precision data format, such as FP32, to a lower-precision data format, such as INT8. In at least one embodiment, a value that has been converted from a higher-precision data format to a lower-precision data format is referred to as a quantized value. In at least one embodiment, a process of converting a quantized value to a higher-precision and/or its original data format is called dequantization. In at least one embodiment, one or more portions of a neural model has quantized values and/or dequantized values. In at least one embodiment, a process of quantizing and/or dequantizing values causes a processor's performance of a neural model to slow due to operations required to quantize and/or dequantize. In at least one embodiment, quantizing and/or dequantizing values reduces accuracy of a neural model due to negative side effects such as rounding errors, underflow, overflow, poor mapping between quantized and original values, or some combination thereof. In at least one embodiment, quantizing and/or dequantizing values affects neural model performance due to factors such as

types of quantizing techniques used and types of data formats best suited for a particular processor.

[0075] In at least one embodiment, an efficient neural model includes two or more operations fused together. In at least one embodiment, a process of merging two or more operations into a single software kernel is referred to operator fusion. In at least one embodiment, a neural model that includes at least two or more operations fused together is referred to as a fused neural model. In at least one embodiment, operator fusion allows an activation function to be performed as part of one or more operations in one or more previous layers, such as a convolution and/or batch normalization layer. In at least one embodiment, operator fusion causes increased computational overhead due to factors such as computing resources required to merge two operations, additional memory required by a fused operation, increased kernel launch times due to kernels containing more operations, or some combination thereof.

[0076] In at least one embodiment, an efficient neural model includes two or more fused layers. In at least one embodiment, layer fusion addresses redundant and/or similar layers by combining weights of those layers with corresponding layers. In at least one embodiment, a fused-layer neural model is a neural model with at least two of its layers fused. In at least one embodiment, layer fusion includes freezing (maintaining) values in one of two layers, freezing gradients in one of two layers; calculating a mean value between a pair of layers, using a mean value during backpropagation; interpolating data between two layers; or some combination thereof. In at least one embodiment, layers of a neural model are convolution layers, where, at least conceptually, convolutions are performed and include matrix multiplication of weights with input values. In at least one embodiment, weights of a neural model include hard and soft weights. In at least one embodiment, soft weights are capable of changing after a trained neural model performs a round of inferencing. In at least one embodiment, hard weights do not change after a trained neural model performs a round of inferencing. In at least one embodiment, layer fusion causes additional computational overhead due to factors such as increased computational complexity, increased memory usage, or some combination thereof. In at least one embodiment, computational overhead as a result of layer fusion is an increased amount of time required to perform a certain set of operations. In at least one embodiment, an efficient neural model

includes a some combination of sparsity, quantization, operator fusion, and layer fusion techniques.

[0077] In at least one embodiment, system 100 comprises one or more processors that trains a neural model that will be used, at least in part, to generate NN variations 104a-*n*, which are described further herein. In at least one embodiment, a neural model is referred to as a neural model. In at least one embodiment, system 100 comprises one or more processors to perform an operation of neural model dry runs module 102 by making modifications to a trained, dense neural model. In at least one embodiment, a trained, dense neural model is dense target neural model (M_{DT}) 412 of FIG. 4. In at least one embodiment, neural model dry runs module 102 generates different variations of a neural model, such as neural model (NN) variations 104a-*n*, where *n* represents a number of different variations. In at least one embodiment, NN variations 104a-*n* are based, at least in part, on a trained, dense neural model. In at least one embodiment, NN variations 104a-*n* are sparse and/or quantized versions of a trained, dense neural model. In at least one embodiment, neural model dry runs module 102 generates NN variations 104a-*n* by using a different combination of fused layers and quantized data in each NN variations 104a-*n*. In at least one embodiment, examples of NN variations 104a-*n* are described in conjunction with FIGS. 2 and 3.

[0078] In at least one embodiment, neural model dry runs module 102 causes GPU hardware platforms 106 to perform dry runs of NN variations 104a-*n*. In at least one embodiment, a GPU hardware platform is referred to as an AI hardware platform. In at least one embodiment, a GPU hardware platform is referred to as hardware resource. In at least one embodiment, GPU hardware platforms 106 include different GPU hardware platforms. In at least one embodiment, GPU hardware platforms include different GPUs. In at least one embodiment, GPU hardware platforms 106 include GPU hardware platform 116 and GPU hardware platform 118. In at least one embodiment, GPU hardware platforms 106 do not include GPU hardware platform 116 and GPU hardware platform 118.

[0079] In at least one embodiment, GPU hardware platforms perform a same neural model differently due to differing specifications and/or constraints related to hardware and/or software. As an example, in at least one embodiment, AMD® Radeon™ RX 6750 GPUs, which use ROCm® software, perform sparse matrix operations differently from NVIDIA® HGX A100™

GPUs, which use CUDA® software despite both ROCm® and CUDA® supporting 2:4 structured sparsity, which is described further herein, CUDA® is capable of using an extra column of data that allows skipping of some zero values during computations. As an example, in at least one embodiment, AMD® GPUs that use CDNA™ 3 GPU architecture improves inferencing performance for operations using FP16 by 3x and INT8 by 6.8x when compared with GPUs that use CDNA™ 2 GPU architecture.

[0080] In at least one embodiment, dry runs of NN variations 104a-*n* on different GPU hardware platforms are simulations of different GPUs performing NN variations 104a-*n*. In at least one embodiment, dry runs of NN variations 104a-*n* on different GPU hardware platforms are performed by actual GPUs performing NN variations 104a-*n*. In at least one embodiment, neural model dry runs module 102 performs dry runs of NN variations 104a-*n* on different GPU hardware platforms and outputs performance metrics that are received and/or otherwise obtained by efficiency lists module 110 implemented on processor(s) 108.

[0081] In at least one embodiment, processor(s) 108 perform one or more operations of efficiency lists module 110 to cause efficiency lists to be stored in a database. In at least one embodiment, an efficiency list is one or more sets of performance metrics generated by one or more dry runs, which are described further herein at least in conjunction with FIG. 4. In at least one embodiment, efficiency lists, which are described further herein at least in conjunction with FIGS. 1B and 4, include details about how specific neural model compression (efficiency) techniques implemented in a neural model affect performance metrics of that neural model being performed on a specific GPU hardware platform. In at least one embodiment, an efficiency list is a spreadsheet or similar, that includes a column that lists an indication of a layer of a neural model. In at least one embodiment, an efficiency list includes another column that lists a data type (e.g., FP32, INT8). In at least one embodiment, an efficient list includes another column that lists a hardware type (e.g., NVIDIA® A100™ GPU). In at least one embodiment, least one column an efficiency list includes another column that lists an inferencing performance speed up factor.

[0082] In at least one embodiment, efficiency lists module 110 performs operations that combine one or more portions of two or more efficiency lists, where that combination is referred to as a fused efficiency list and is described further herein at least in conjunction with FIG. 4. In

at least one embodiment, efficiency lists module 110 performs operations that analyze a fused efficiency list and outputs that fused efficiency list to masking module 114.

[0083] In at least one embodiment, processor(s) 112 performs one or more operations of masking module 114 to insert masks, described further herein at least in conjunction with FIG. 5, into a trained, dense neural model. In at least one embodiment, when a masking module 114 inserts masks into a trained, dense neural model, that model requires additional training due to a decrease in accuracy of that model. In at least one embodiment, masking module 114 inserts masks into a trained, dense neural model to change that neural model into an untrained, efficient neural model, due to a decrease in accuracy of that neural model below levels acceptable to a user and/or application, and as described further herein at least in conjunction with FIG. 5. Emb In at least one embodiment, an efficient neural model is a quantized and/or sparse neural model. In at least one embodiment, an untrained, efficient neural model is trained using input data. In at least one embodiment, input data used to train a dense neural model is input data also used to train an untrained, efficient neural model based on that dense neural model. In at least one embodiment, an untrained, an efficient neural model is trained to become a trained, efficient neural model, also referred to as a masked super neural model. In at least one embodiment, a masked super neural model is further modified to become a trained, efficient neural model for deployment, described further herein at least in conjunction with FIG. 6.

[0084] In at least one embodiment, a masked super neural model is output by masking module 114 and transferred to GPU hardware platform 116 and/or GPU hardware platform 118. In at least one embodiment, transferring any data, including data of a neural model or a efficiency list, uses any mode of transfer including hardwired connections, wireless connections, portable drives, memory cards, or some combination thereof. In at least one embodiment, a trained, efficient neural model output by masking module 114 is further modified based on efficiency lists associated with an GPU hardware platform identified to perform that modified trained, efficient neural model. In at least one embodiment, GPU hardware platform 116 and/or GPU hardware platform 118 are different GPU hardware platforms using different types of GPUs. In at least one embodiment, a modified trained, efficient neural model is referred to as an efficient neural model for deployment, described further herein at least in conjunction with FIG. 6.

[0085] FIG. 1B illustrates a database 130 that stores efficiency lists to be used by a processor comprising one or more circuits to select one or more masks to be used by one or more neural models, according to at least one embodiment. In at least one embodiment, FIG. 1B illustrates simplified examples of efficiency lists. In at least one embodiment, FIG. 1B illustrates efficiency list 132 and fused efficiency list 134. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 1B are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A and 3-8B. In at least one embodiment, one or more processors generate one or more efficiency lists and cause those efficiency lists to be stored in database 130. In at least one embodiment, one or more processors that perform one or more operations used to generate and store efficiency lists in database 130 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, database 130 is any one or combination of databases described herein. In at least one embodiment, database 130 is retrieval database 4314 of FIG. 43. In at least one embodiment, database 130 hardware, firmware, software, or a combination thereof, used to store and organize data to be accessed and/or retrieved by an application and/or processor. In at least one embodiment, database 130 includes memory used to store efficiency lists, memory such as random access memory (RAM), read only memory (ROM), a hard disk drive (HDD), a solid state drive (SDD), cache memory, flash memory, or some combination thereof. In at least one embodiment, database 130 includes memory 814 of FIG. 8.

[0086] In at least one embodiment, efficiency lists module 110 of FIG. 1 and/or efficiency lists module 410 of FIG. 4 generate efficiency list 132 and fused efficiency list 134. In at least one embodiment, efficiency list 132 and/or efficiency list 134 includes a list of one or more residual block modifications 202 of FIG. 2 applied to a neural model to be used during dry runs.. In at least one embodiment, efficiency list 132 and/or efficiency list 134 includes a list of one or more self-attention block modifications 302 applied to a neural model to be used during dry runs. In at least one embodiment, efficiency list 132 and/or fused efficiency list 134 are used by masking module 540 of FIG. 5 to select masks to insert into a neural model. In at least one embodiment, a processor modifies efficiency list 132 and/or fused efficiency list 134 to generate refreshed

efficiency list 622 of FIG. 6 based on parameters of a GPU hardware platform. In at least one embodiment, processor(s) 822 perform one or more aspects of operation 704 of FIG. 7 to generate efficiency list 132. In at least one embodiment, processor(s) 822 performs an API of API(s) 810 of FIG. 8A to cause generation of efficiency list 132 based on dry runs of neural models. In at least one embodiment, processor(s) 822 perform neural model optimization module 824 to generate efficiency list 132 after modifying a neural network using one or more neural model optimization techniques.

[0087] In at least one embodiment, efficiency list 132 is a table and/or spreadsheet. In at least one embodiment, data contained in efficiency list 132 is stored in any format, including charts, graphs, arrays, vectors, matrices, tensors, linked lists, trees, or some combination thereof. In at least one embodiment, efficiency list 132 is one of efficiency list(s) for specific platform 422a-f, which are described further herein at least in conjunction with FIG. 4. In at least one embodiment, efficiency list 132 includes a column that lists an indication of a layer of a neural model. In at least one embodiment, efficiency list 132 includes a column that lists a data format (data type) (e.g., FP32, INT8) used for values in a layer of a neural model. In at least one embodiment, efficiency list 132 includes a column that lists a hardware type (e.g., NVIDIA® A100™ GPU). In at least one embodiment, efficiency list 132 includes a column that lists an inferencing performance speed up factor (e.g., 1.1x). In at least one embodiment, efficiency list 132 indicates a change in inferencing performance of a layer based on modifications made to a layer's data type used, sparse ratio used, and hardware used. In at least one embodiment, each row of efficiency list 132 indicates modifications made to a specific layer of a neural model, hardware used to perform that layer, and performance results using those modifications and hardware for each dry run of a neural model.

[0088] In at least one embodiment, fused efficiency list 134 is a type of efficiency list, and is described further herein at least in conjunction with fused efficiency lists module 430 of FIG. 4. In at least one embodiment, fused efficiency list 134 is a modification and/or combination of two efficiency lists, such as efficiency list 132. In at least one embodiment, fused efficiency list 134 includes only data related to dry runs using a combination of optimization techniques and hardware that showed an inferencing speed improvement above a certain threshold level (e.g., above a speed up factor of 1.4x). In at least one embodiment, fused efficiency list 134 includes

only data related to dry runs using a combination of optimization techniques and hardware that showed inferencing speed improvement greater than any other combination of optimization techniques used with that hardware. In at least one embodiment, fused efficiency list 132 includes types of information identical to efficiency list 132, such as indication of layer, data type, sparse ratio, hardware, and inferencing performance speed up factor.

[0089] FIG. 2 illustrates, as a block diagram, a system 200 that makes modifications to a residual block of a neural model to be used in one or more dry runs when generating efficiency lists, according to at least one embodiment. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 2 are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-1B and 3-8B. In at least one embodiment, one or more processors perform one or more operations used to make residual block modifications 202. In at least one embodiment, one or more processors that perform one or more operations used to make residual block modifications 202 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 performs an operation to fuse two or more layers as shown in FIG. 2. In at least one embodiment, processor(s) 822 performs an operation to quantize values in a convolution layer as shown in FIG. 2. In at least one embodiment, neural model dry runs module 102 of FIG. 1A performs one or more operations to generate one or more modifications shown in residual block modifications 202 to create efficient variations of a trained, dense neural model. In at least one embodiment, one or more modifications of residual block modifications 202 are combined with self-attention block modifications 302 of FIG. 3. In at least one embodiment, dry runs on various hardware platforms 420 of FIG. 4 use residual block modifications 202 to perform dry runs. In at least one embodiment, efficiency lists of efficiency lists module 210 are used, at least in part, by masking module 540 to insert masks into masked super neural model (M_{MS}) 550. In at least one embodiment, efficiency lists of efficiency lists module 210 are included in refreshed efficiency list for specific platform 622 of FIG. 6. In at least one embodiment, one or more operations of used to generate residual block modifications 202 are used as part of operation 701 to generate efficient variations of neural models based on a trained,

dense neural model. In at least one embodiment, an API of API(s) 810 of FIG. 8A causes modifications to a neural model such as residual block modifications 202. In at least one embodiment, neural model optimization module 824 performs operations to cause modifications to a neural model such as residual block modifications 202.

[0090] In at least one embodiment, residual block 204 is a specific arrangement of layers in a neural model to perform one or more specific functions. In at least one embodiment, residual block 204 is part of a neural model architecture such as a residual network (ResNet). In at least one embodiment, residual block 204 is used in a convolutional neural model (CNN). In at least one embodiment, residual block 204 includes multiple layers including convolution layers, such as convolution layer 207. In at least one embodiment, residual block 204 includes a batch normalization layer, such as BatchNorm layer 208, which uses a batch normalization technique to normalize a distribution of values. In at least one embodiment, residual block 204 includes a scale layer, such as scale layer 209, which includes operations that adjust sizes (dimensions) of tensors and/or scales values in one or more dimensions of a tensor. In at least one embodiment, residual block 204 includes a rectified linear unit (ReLU) layer, such as ReLU layer 211, where ReLU is an activation function.

[0091] In at least one embodiment, residual block 204 includes a connection 206 that causes an output of rectified linear unit (ReLU), an activation function, to skip a set of layers 220 and be input a set of layers 222. In at least one embodiment, neural model dry runs module 102 performs optimization operations to modify different layers of residual block 204 using different combinations of neural model optimization techniques, which are described further herein at least in conjunction with FIGS. 1 and 3. In at least one embodiment, optimization operations are referred to as compression operations or efficiency operations. In at least one embodiment, residual block modifications 202 includes indications of types of optimization techniques that have been applied to a layer, such as overhead/fusion 222, data type 226, and sparse ratio 228. In at least one embodiment, an overhead/fusion symbol, such as overhead/fusion 222, and its corresponding arrows indicates an instance of operator fusion applied to operations of two or more layers. In at least one embodiment, an overhead/fusion symbol indicates that computational overhead as a result of operator fusion is being measured by one or more components such as neural model dry runs module 102 and/or efficiency lists module 110 of FIG. 1.

[0092] In at least one embodiment, a data type symbol, such as data type 226, indicates an instance of quantization applied to one or more values in a layer identified by that symbol's corresponding arrow. In at least one embodiment, quantization converts values from a floating point data format into an integer data format. In at least one embodiment, quantization includes any type of quantization techniques, including dynamic quantization, static quantization, vector quantization, low-bit quantization, ternary quantization, or some combination thereof.

[0093] In at least one embodiment, a sparse ratio symbol, such as sparse ratio 228, indicates an instance of sparsity applied, in accordance with a sparse ratio, to a layer identified by that symbol's corresponding arrow. In at least one embodiment, a sparse ratio is a number of zero values (elements) per given block in a tensor. In at least one embodiment, a 2:4 sparse ratio means that two out of each contiguous block of four values in a tensor must be zero. In at least one embodiment, a sparse ratio can be defined in terms of how many values in a given contiguous block are nonzero. In at least one embodiment, a sparse ratio that follows specific constraints, such as a 2:4 ratio, is referred to as a type of structured sparsity. In at least one embodiment, structured sparsity is a type of specification and/or parameter of a processor.

[0094] In at least one embodiment, residual block modifications 202 include modifications made through operator fusion. In at least one embodiment, multiple operations are fused together, such as those of batch normalization and those of ReLu, as shown by overhead/fusion 222 and dashed arrows pointing to data type symbols that correspond to layers Batch Norm and ReLu as depicted in set of layers 222. As another example, in at least one embodiment, multiple operations are fused together, such as those of batch normalization and those of convolution as shown by overhead/fusion 224 and dashed arrows pointing into data type symbols corresponding to layers Batch Norm and convolution and ReLu depicted in set of layers 222.

[0095] In at least one embodiment, multiple types of optimization techniques are applied to a single layer. In at least one embodiment, optimizations of a neural model is referred to as a modification of a neural model. In at least one embodiment, a layer is modified using quantization, sparsity, operator fusion, and layer fusion techniques. In at least one embodiment, a convolution layer in set of layers 222 is quantized, sparse, and operator fused as shown with data type 226, sparse ratio 228, overhead/fusion 224, and their corresponding arrows.

[0096] In at least one embodiment, residual block modifications 202 are sent to and received, or otherwise obtained, by efficiency lists module 210. In at least one embodiment, efficiency lists module 210 stores neural model optimizations made to one or more portions of a neural model and correlates those optimizations with performance metrics collected during dry runs as further described at least in conjunction with FIG. 4. In at least one embodiment, a database that includes correlations of one or more neural model optimizations with performance metrics and specific hardware platforms is referred to as an efficiency list, such as any of efficiency list(s) for specific platform 422a-f of FIG. 4.

[0097] FIG. 3 illustrates, as a block diagram, a system 300 that makes modifications to a multi-headed self-attention block of a neural model to be used as part of one or more dry runs when generating efficiency lists, according to at least one embodiment. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 3 are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-2, and 4-8B. In at least one embodiment, one or more processors perform one or more operations used to make self-attention block modifications 302. In at least one embodiment, one or more processors that perform one or more operations used to make self-attention block modifications 302 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 performs an operation to fuse two or more operations as shown in FIG. 3. In at least one embodiment, processor(s) 822 performs an operation to quantize values in a matrix-multiplication layer as shown in FIG. 3. In at least one embodiment, neural model dry runs module 102 of FIG. 1A performs one or more operations to generate one or more modifications shown in self-attention block modifications 302 to create efficient variations of a trained, dense neural model. In at least one embodiment, one or more modifications of self-attention block modifications 302 are combined with residual block modifications 202 of FIG. 2. In at least one embodiment, dry runs on various hardware platforms 420 of FIG. 4 use self-attention block modifications 302 to perform dry runs. In at least one embodiment, efficiency lists of efficiency lists module 310 are used, at least in part, by masking module 540 to insert masks into masked super neural model

550. In at least one embodiment, efficiency lists of efficiency lists module 310 are included in refreshed efficiency list for specific platform 622 of FIG. 6. In at least one embodiment, one or more operations used to generate self-attention block modifications 302 are used as part of operation 701 to generate efficient variations of a trained, dense neural model. In at least one embodiment, an API of API(s) 810 of FIG. 8A causes modifications to a neural model, such as self-attention block modifications 302. In at least one embodiment, neural model optimization module 824 performs operations to cause modifications to a neural model, such as self-attention block modifications 302.

[0098] In at least one embodiment, self-attention block 304 is a specific arrangement of neural model layers to perform a specific function. In at least one embodiment, self-attention block 304 is a multi-headed self-attention block. In at least one embodiment, self-attention block 304 is used as part of a neural model designed for natural language processing (NLP) tasks. In at least one embodiment, self-attention block 304 is part of a transformer type neural model architecture. In at least one embodiment, self-attention block 304 uses attention heads 306 to allow a neural model to learn different aspects of an text-based input sequence, such syntactic structure, semantic meaning, or some combination thereof. In at least one embodiment, each attention head of attention heads 306 uses its own parameters and weights to allow a neural model to learn different aspects of an input sequence. In at least one embodiment, attention heads improves a model's performance and accuracy on various NLP tasks, such as machine translation, text summarization, question answering, text generation, and more.

[0099] In at least one embodiment, self-attention block 304 includes multiple layers including matrix multiplication layers, such as mat-mul layer 307. In at least one embodiment, self-attention block 304 includes a softmax layer, such as softmax layer 308, which uses a softmax function to normalize a distribution of values. In at least one embodiment, self-attention block 304 includes an optional mask layer, such as mask layer 309, which selects which sets of values corresponding to specific positions and/or tokens to ignore. In at least one embodiment, self-attention block 304 includes a scale layer, such as scale layer 311, which includes operations that scales one or more dimensions of tensors and/or scales values of one or more dimensions of a tensor.

[0100] In at least one embodiment, neural model dry runs module 102 performs optimization operations to modify different layers of self-attention block 304 using different combinations of neural model optimization techniques, which are described further herein at least in conjunction with FIGS. 1 and 2. In at least one embodiment, optimization operations are referred to as compression operations or efficiency operations. In at least one embodiment, self-attention block modifications 302 includes indicators of which types of optimization techniques have been applied to a layer, such as overhead/fusion 322, data type 326, and sparse ratio 328. In at least one embodiment, an overhead/fusion symbol, such as overhead/fusion 322, and its corresponding arrows indicates an instance of operator fusion applied to a two or more layers. In at least one embodiment, an overhead/fusion symbol indicates a monitoring of how an operator fusion affects computational overhead, where such monitoring is performed by one or more components such as neural model dry runs module 102 and/or efficiency lists module 110 of FIG. 1.

[0101] In at least one embodiment, a data type symbol, such as data type 326, indicates an instance of quantization applied to one or more values in a layer identified by that symbol's corresponding arrow. In at least one embodiment, quantization converts values from a floating point data format into an integer data format. In at least one embodiment, quantization includes any type of quantization techniques, including dynamic quantization, static quantization, vector quantization, low-bit quantization, ternary quantization, or some combination thereof.

[0102] In at least one embodiment, a sparse ratio symbol, such as sparse ratio 328, indicates an instance of sparsity applied, in accordance with a sparse ratio, to a layer identified by that symbol's corresponding arrow. In at least one embodiment, a sparse ratio is a number of zero values (elements) per given block in a tensor. In at least one embodiment, a 2:4 sparse ratio means that two out of each contiguous block of four values in a tensor must be zero. In at least one embodiment, a sparse ratio can be defined in terms of how many values in a given contiguous block are nonzero.

[0103] In at least one embodiment, self-attention block modifications 302 include modifications made through operator fusion. In at least one embodiment, multiple operations are fused together, such as those of mat-mul layer 312 and linear attention heads K and V of attention heads 306, and as shown by dashed arrows pointing in between data type and sparse

ratio symbols correlating to mat-mul layer 312 and attention heads K and V of attention heads 206.

[0104] In at least one embodiment, multiple types of optimization techniques are applied to each of two layers and then fused. In at least one embodiment, fused operations are referred to as a fusion. In at least one embodiment, data type and sparse ratio optimization techniques are applied to mat-mul layer 312. In at least one embodiment, data type and sparse ratio optimization techniques are applied to data values of attention head K of attention heads 306. In at least one embodiment, two layers that have each been optimized using two different optimization techniques, have their operations fused together, such as a fusion shown with overhead fusion symbol 322 and mat-mul layer 312 and attention head K of attention heads 306.

[0105] In at least one embodiment, self-attention block modifications 302 are sent to, or obtained by, efficiency lists module 310. In at least one embodiment, efficiency lists module 310 stores neural model optimizations made to one or more portions of a neural model and correlates those optimizations with performance metrics collected during dry runs as further described at least in conjunction with FIGS. 1 and 4. In at least one embodiment, a database that correlates one or more optimizations made to a neural model with performance metrics is referred to as an efficiency list, such as any of efficiency list(s) for specific platform 422a-f of FIG. 4. In at least one embodiment, an efficiency list is not a list, but a table, spreadsheet, tensor, or some combination thereof.

[0106] FIG. 4 illustrates, as a block diagram, a system that, at least in part, performs dry runs of neural models, according to at least one embodiment. In at least one embodiment, neural model dry runs system 400 comprises one or more processors to perform operations that make modifications to a trained, dense neural model to be used as part of one or more dry runs when generating efficiency lists. In at least one embodiment, a trained, dense neural model used as part of neural model dry runs system 400 is referred to as a dense target neural model. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 4 are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-3, and 5-8B. In at least one embodiment, one or more processors perform one or more operations used to generate efficiency lists such as those generated by efficiency lists module 410. In at least one

embodiment, one or more processors that perform one or more operations used as part of neural model dry runs system 400 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 performs an operation to generate an efficiency list of efficiency lists module 410. In at least one embodiment, processor(s) 822 performs an operation of fused efficiency lists module 430. In at least one embodiment, neural model dry runs module 102 of FIG. 1A performs one or more operations to generate one or more modifications of a dense target neural model 412 to be used as part of dry runs on various hardware platforms 420. In at least one embodiment, one or more residual block modifications 202 of FIG. 2 are implemented in dense target neural model 412. In at least one embodiment, one or more of self-attention block modifications 302 of FIG. 3 are implemented in dense target neural model 412. In at least one embodiment, a fused efficiency list of fused efficiency lists module 430 are used, at least in part, by masking module 540 to insert masks into masked super neural model 550. In at least one embodiment, an efficiency list of efficiency lists module 410 is included in a refreshed efficiency list for specific platform 622 of FIG. 6. In at least one embodiment, one or more operations used by efficiency lists module 410 are used as part of operation 704 to generate one or more efficiency lists. In at least one embodiment, an API of API(s) 810 of FIG. 8A causes efficiency lists to be generated by efficiency lists module 410. In at least one embodiment, neural model optimization module 824 performs operations to cause modifications to a dense target neural model 412.

[0107] In at least one embodiment, efficient variations of a dense target neural model 412 are used in dry runs on different hardware platforms. In at least one embodiment, dense target neural model 412 is any one or combination of neural models described further herein. In at least one embodiment, dense target neural model 412 is a convolutional neural model (CNN), such as any one or combination of CNNs described in connection with FIG. 12C. In at least one embodiment, dense target neural model 412 is a large language model (LLM), such as LLM 4312 described in conjunction with FIG. 43. In at least one embodiment, dense target neural model 412 is a trained model.

[0108] In at least one embodiment, dense target neural model 412 includes input and position embedding block 414. In at least one embodiment, input and position embedding block 414 is part of a transformer type of neural model architecture. In at least one embodiment, input and position embedding block 414 includes an embedding layer that maps input information from a high-dimensional to a lower-dimensional tensor. In at least one embodiment, an embedding layer converts each element of a sequence, such as each word in a sentence, of input data into tensors comprising numerical values that capture semantic and syntactic features of those words. In at least one embodiment, input and position embedding block 414 includes an encoding layer that adds information about a position (location) of each element in a sequence, such as each word in a sentence, used as input data.

[0109] In at least one embodiment, dense target neural model 412 includes a transformer block 416. In at least one embodiment, transformer block is or includes a multi-headed self-attention block such as self-attention block 304 of FIG. 3. In at least one embodiment, transformer block 416 is used in natural language processing tasks, such as machine translation, text summarization, text generation, question answering, and sentiment analysis.

[0110] In at least one embodiment, dense target neural model 412 includes final projection block 418. In at least one embodiment, a final projection block includes a final output, such as predictions or inferences, of a transformer type of neural model. In at least one embodiment, a final projection block is used in tasks such as classification of objects identified in images.

[0111] In at least one embodiment, dry runs on various hardware platforms 420 is a set of dry runs of efficient variations of dense target neural model 412. In at least one embodiment, dry runs on various hardware platforms 420 includes dry runs on various GPU hardware platforms. In at least one embodiment, efficient variations of dense target neural model 412 are generated by modifying one or more layers of dense target neural model 412 using neural model dry runs module 102 of FIG. 1A and/or operation 701 of FIG. 7. In at least one embodiment, efficient variations of dense target neural model 412 include one or more modifications described in conjunction with at least FIGS. 2 and 3.

[0112] In at least one embodiment, various hardware platforms include a different processor and/or a different combination of processors between those various hardware platforms. In at least one embodiment, various hardware platform use different GPUs, such as GPUs that have

different sparsity requirements, different data format requirements, different modules that perform quantization and dequantization, or some combination thereof. In at least one embodiment, efficient variations of a dense target neural model are generated using different combinations of data formats, quantization, operator fusion, and layer fusion techniques. In at least one embodiment, an efficient variation of a dense target neural model is run on different hardware platforms. In at least one embodiment, efficient variations of a dense target neural model are run on a single hardware platform. In at least one embodiment, neural model dry runs module 102 of FIG. 1A records and organizes performance results of each dry run of dry runs on various hardware platforms 420. In at least one embodiment, performance results, which include performance metrics, of each dry run are output to efficiency lists module 410. In at least one embodiment, dry runs on various hardware platforms 420 are one or more inferencing operations performed by efficient variations of a dense target neural model that each use a single set of input data, such as a set of images or text, but on different hardware platforms.

[0113] In at least one embodiment, efficiency lists module 410 receives and/or otherwise obtains performance metrics that are recorded and output by neural model dry runs module 102 of FIG. 1. In at least one embodiment, efficiency lists module 410 organizes performance metrics and information related to neural model optimizations into efficient lists, such as efficiency list(s) for specific platform 422a-f. In at least one embodiment, efficiency lists module 410 stores efficiency list(s) for specific platform 422a-f in a type of datastore, database, memory, or some combination thereof. In at least one embodiment, each efficiency list of efficiency list(s) for specific platform 422a-f includes performance results of multiple variations of a dense target neural model.

[0114] In at least one embodiment, performance results are indicated by performance metrics that indicate how long a neural model takes to perform one or more inferencing tasks. In at least one embodiment, performance results are indicated by performance metrics that indicate how accurate a neural model is when performing one or more inferencing tasks. In at least one embodiment, performance results are indicated by performance metrics, which are units of measure such as floating point operations per second (FLOPS), trillion operations per second (TOPS), throughput, or some combination thereof. In at least one embodiment, a performance metric is inferencing speed, which is a number of elements of input data, such as a number of

image samples, for which a neural model generates an output in a given time. In at least one embodiment, inferencing speed is represented in units of samples per second or inputs per second. In at least one embodiment, throughput refers to a number of operations that can be processed per unit of time. In at least one embodiment, a performance metric is a value, such as a factor, fraction, integer, real number, or percentage, indicating how much faster or slower performance of an efficient neural model is when compared to a dense neural model on a specific hardware resource. In at least one embodiment, a performance metric is a value related to accuracy of a neural model, such as classification accuracy, logarithmic loss, true positives, true negatives, false positives, false negatives, precision, area under curve, mean absolute error, mean squared error, or some combination thereof. In at least one embodiment, how much an efficient neural model improves its inferencing speed and/or accuracy is based on performance metrics of a corresponding dense neural model performed on a specific hardware platform. In at least one embodiment, how much an efficient neural model improves its inferencing speed and/or accuracy over a corresponding dense neural model, is indicated as a times (multiplication) value, such as 1.5x, or 0.75x, which is sometimes referred to as a factor. In at least one embodiment, efficiency list(s) for specific platform 422a-f each include performance metrics for an efficient variation of a dense target neural model that correspond to a set of modifications made to that neural model and a hardware platform that performed that neural model.

[0115] In at least one embodiment, fused efficiency lists module 430 receives and/or otherwise obtains efficiency list(s) for specific platform 422a-f output by efficiency lists module 410. In at least one embodiment, fused efficiency lists module 430 performs an analysis of efficiency list(s) for specific platform 422a-f. In at least one embodiment, analysis of efficiency list(s) for specific platform 422a-f includes one or more identifications of which combination of optimizations applied to dense target neural model 412 cause performance improvements to be greatest, as shown among multiple dry runs, while using a particular hardware platform. In at least one embodiment, analysis of efficiency list(s) for specific platform 422a-f includes one or more identifications of which combination of optimizations applied to dense target neural model 412 cause performance metrics to meet and/or exceed a threshold performance value, as shown among multiple dry runs, while using a particular hardware platform. In at least one embodiment, a threshold performance value is referred to as a threshold performance metric. In at least one embodiment, a threshold performance metric is referred to as a threshold value. In at least one

embodiment, a threshold performance metric is set by a user and/or application. In at least one embodiment, a threshold value indicates a minimum level of acceptable inferencing speed and/or accuracy according to a user and/or application. In at least one embodiment, fused efficiency lists module 430 generates a fused efficiency list, which includes a set of combinations of optimizations and corresponding hardware platforms that cause one or more efficient variations of dense target neural models to perform at or above a threshold level with respect to time and/or accuracy. In at least one embodiment, a threshold level is referred to as a threshold performance value. In at least one embodiment, a threshold performance value is referred to as a threshold performance metric. In at least one embodiment, a metric is a value. In at least one embodiment, fused efficiency lists module 430 outputs a fused efficiency list to masking module 540 of FIG. 5.

[0116] FIG. 5 illustrates, as a block diagram, a system that, at least in part, performs operations that insert optimization masks into a dense target neural model, according to at least one embodiment. In at least one embodiment, super neural model masking system 500 comprises one or more processors to perform operations that insert data conversion masks and sparse pruning masks based on a fused efficiency list output by fused efficiency lists module 450 of FIG. 4. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 5 are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-4, and 6-8B. In at least one embodiment, one or more processors perform one or more operations used during super neural model masking system 500 such as training masked super neural model 550. In at least one embodiment, one or more processors that perform one or more operations used as part of super neural model masking system 500 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 performs an operation to insert one or more optimization masks to create a masked super neural model 550. In at least one embodiment, masking module 114 of FIG. 1A performs one or more operations of masking module 540. In at least one embodiment, one or more residual block modifications 202 of FIG. 2 are implemented in masked super neural model 550. In at least one embodiment, one or more of self-attention block modifications 302 of FIG. 3 are implemented in masked super neural model 550. In at least

one embodiment, a fused efficiency list of fused efficiency lists module 430 are used, at least in part, by masking module 540 to insert masks into masked super neural model 550. In at least one embodiment, masked super neural model 550 is used, at least in part, in generating efficient neural model for deployment 660 of FIG. 6. In at least one embodiment, one or more operations performed by masking module 540 are included as part of operation 708 of FIG. 7. In at least one embodiment, an API of API(s) 810 of FIG. 8A causes masks to be inserted into a neural model by masking module 540. In at least one embodiment, neural model optimization module 824 performs operations to insert masks into a neural model by using masking module 540.

[0117] In at least one embodiment, masking module 540 obtains and/or otherwise receives one or more fused efficiency lists that are output by fused efficiency lists module 430. In at least one embodiment, masking module 540 inserts and/or otherwise implements one or more optimization masks based, at least in part, on one or more fused efficiency lists. In at least one embodiment, masking module 540 inserts one or more data conversion mask(s) 542 and one or more sparse pruning mask(s) 544 into a trained, dense target neural model, such as dense target neural model 412 of FIG. 4. In at least one embodiment, masking module 540 inserts masks into an untrained, dense neural model, a trained sparse neural model, untrained sparse neural model, or some combination thereof. In at least one embodiment, masking module 540 uses fused efficiency lists to insert different combinations of neural model optimizations at different locations, such as before, after, or in specific layers, of a dense, target neural model. In at least one embodiment, FIG. 5 illustrates a simplified example of how and where two types of masks might be inserted into various blocks of a neural model by using arrows and representations of layers within each block.

[0118] In at least one embodiment, a mask is a tensor of values that indicate which parameters and/or values of a neural model should be modified when that neural model performs training and/or inferencing tasks. In at least one embodiment, a mask is a tensor that indicates which nonzero values of one or more neural networks should be set to zero based on performance metrics such as inferencing speed and/or inferencing accuracy. In at least one embodiment, a mask is a tensor comprising binary values such as 0 and 1. In at least one embodiment, a mask is a tensor multiplied with values of a neural model layer to convert one or more of those layers. In at least one embodiment, a mask includes values that indicate which values of a layer should

input into functions that convert those values from one data type into another data type as part of a quantization process. In at least one embodiment, a mask modifies data values of a layer to cause quantization of those data values and/or to convert those data values into zeros. In at least one embodiment, data conversion mask(s) 542 is used to select which values, such as weights, should be quantized and/or dequantized and with which data format(s). In at least one embodiment, a data conversion mask inserted into a neural model reduces that model's size and computational requirements. In at least one embodiment, data conversion mask(s) 542 are applied to any values and/or functions of a neural model. In at least one embodiment, data conversion mask(s) 542 inserted into a neural model causes one or more data type modifications as described in further herein and at least in conjunction with FIGS. 2 and 3.

[0119] In at least one embodiment, sparse pruning mask(s) 544 are used to indicate which nonzero parameters and/or values, such as weights, of neural model should be set and/or changed to zero. In at least one embodiment, using sparse pruning mask(s) 544 with weights effectively removes a connection to one or more nodes of a previous neural model layer. In at least one embodiment, a sparse pruning mask is applied to a layer to reduce that layer's size in terms of a number of nonzero values included in that layer, in terms of dimensions, in terms of data size, or some combination thereof. In at least one embodiment, a sparse pruning mask is referred to as a pruning mask or a sparse mask. In at least one embodiment, a sparse pruning mask(s) 544 are applied to any value and/or function of a neural model. In at least one embodiment, sparse pruning mask(s) 544 inserted into a neural model network causes one or more sparse ratio type modifications as described further herein and at least in conjunction with FIGS. 2 and 3. In at least one embodiment, a single mask is used to implement two or more optimization techniques, such as quantization and sparsity, in a layer of a neural model.

[0120] In at least one embodiment, masked super neural model 550 is created by masking module 540 inserting one or more masks into trained, dense neural model such as dense target neural model 412 of FIG. 4. In at least one embodiment, masked super neural model 550 is a neural model used to generate efficient neural models for deployment, such as efficient neural model for deployment 660 of FIG. 6. In at least one embodiment, after masking module 540 inserts masks into a trained, dense target neural model, that model is considered to be untrained since accuracy of that model is likely to have decreased below a level acceptable to a user and/or

application. In at least one embodiment, a masked neural model is an efficient neural model. In at least one embodiment, a masked, untrained target neural model becomes masked super neural model 550 after being trained. In at least one embodiment, at least conceptually, masked super neural model 550 is considered to be a re-trained, efficient version of trained, dense target neural model 412 of FIG. 4.

[0121] In at least one embodiment, a data conversion mask is inserted at one layer of input and position embedding block 514. In at least one embodiment, a sparse pruning mask is inserted at an output layer of input and position embedding block 514. In at least one embodiment, input and position embedding block 514 shows an arrow pointing towards transformer block 516 to indicate an output of data being used as input data to transformer block 516.

[0122] In at least one embodiment, one or more types of data conversion masks are inserted at three layers of transformer block 516. In at least one embodiment, different types of data conversion masks use different quantization methods, dequantization methods, data formats, or some combination thereof. In at least one embodiment, one or more types of sparse pruning masks are inserted at three layers of transformer block 516. In at least one embodiment, different types of sparse pruning masks use different sparse ratios, cause sparsity in different types of layers, or some combination thereof.

[0123] In at least one embodiment, a data conversion mask is inserted at one layer of final projection block 518. In at least one embodiment, a sparse pruning mask is inserted at one layer final projection block 518. In at least one embodiment, masks inserted in final projection block 518 prepare data output by transformer block 516 to undergo further operations such as classification, fine-tuning, regression, or some combination thereof.

[0124] In at least one embodiment, a masked but untrained neural model is trained with input data 560. In at least one embodiment, input data 560 comprises text. In at least one embodiment, input data 560 comprises image samples (images). In at least one embodiment, input data 560 comprises input data also used to train a dense, trained neural model, such as dense target neural model 412 of FIG. 4. In at least one embodiment, input data 560 comprises data different from input data used to train a dense, trained neural model, such as dense target neural model 412 of FIG. 4. In at least one embodiment, input data 560 comprises a mix of data used and not used to train a dense, trained neural model, such as dense target neural model 412 of FIG. 4. In at least

one embodiment, masked super neural model 550 are output to eventually become one or more neural models for deployment on specific hardware platforms, such as efficient neural model for deployment (M_{ED}) 660 and efficient neural model for deployment (M_{ED}) 662 of FIG. 6. In at least one embodiment, input data 560 are used in knowledge distillation, described further herein at least in conjunction with FIG. 6, by one or more neural models for deployment on specific hardware platforms, such as efficient neural model for deployment (M_{ED}) 660 and efficient neural model for deployment (M_{ED}) 662 of FIG. 6.

[0125] FIG. 6 illustrates, as a block diagram, a system 600 that, at least in part, comprises one or more processors to perform operations that modify a neural model such that a specific GPU hardware platform will perform that neural model to meet and/or exceed a threshold performance metric, according to at least one embodiment. In at least one embodiment, efficient neural model deployment system 600 includes operations that perform knowledge distillation of an efficient neural model to be deployed on a specific GPU hardware platform. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 6 are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-5, and 7-8B. In at least one embodiment, one or more processors perform one or more operations used during efficient neural model deployment system 600 such as calculating loss functions such as loss functions 670 used in knowledge distillation of a neural model. In at least one embodiment, one or more processors that perform one or more operations used as part of efficient neural model deployment system 600 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 performs an operation insert one or more optimization masks to create a masked super neural model 550. In at least one embodiment, GPU hardware platform 116 of FIG. 1A performs efficient neural model for deployment 660. In at least one embodiment, one or more residual block modifications 202 of FIG. 2 are implemented in efficient neural model for deployment 660. In at least one embodiment, one or more of self-attention block modifications 302 of FIG. 3 are implemented in efficient neural model for deployment 660. In at least one embodiment, an efficiency list of efficiency lists module 410 are used, at least in part, as

refreshed efficiency list for specific platform 622. In at least one embodiment, masked super neural model 550 of FIG. 5 is used, at least in part, in generating efficient neural model for deployment 660. In at least one embodiment, one or more operations used to perform knowledge distillation as part of efficient neural model for deployment system 600 are included as part of operation 716 of FIG. 7. In at least one embodiment, an API of API(s) 810 of FIG. 8A causes masks to be removed from efficient neural model for deployment 660. In at least one embodiment, neural model optimization module 824 performs operations to remove masks efficient neural model for deployment 660.

[0126] In at least one embodiment, efficient neural model for deployment system 600 includes an identification of one or more GPU hardware platforms that will perform one or more efficient neural models based on masked super neural model 550 of FIG. 5. In at least one embodiment, identified GPU hardware platforms include GPU hardware platform 680 and GPU hardware platform 682. In at least one embodiment, once GPU hardware platforms have been identified to perform efficient neural models, parameters (specifications) of those identified GPU hardware platforms are obtained. In at least one embodiment, obtained parameters of identified GPU hardware platforms are GPU hardware platform parameters 616 and GPU hardware platform parameters 618.

[0127] In at least one embodiment, GPU hardware platform parameters 616 and GPU hardware platform parameters 618 are calibrated by calibration module 620. In at least one embodiment, calibrating GPU hardware platform parameters includes adjusting parameters of a GPU to optimize its performance and efficiency for a type of neural model, such as a CNN or LLM, that will be performed by that GPU. In at least one embodiment, calibration includes adjusting settings of a GPU driver, selecting operations of an application programming interface (API) used to perform neural model operations, or some combination thereof. In at least one embodiment, calibration of GPU hardware platform parameters is based, at least in part, on each GPU's different capabilities and limitations, such as memory size, power consumption, temperature, utilization, or some combination thereof.

[0128] In at least one embodiment, after calibration module 620 calibrates GPU hardware platform parameters, calibration module 620 outputs a refreshed efficiency list for a specific platform, such as refreshed efficiency list for a specific platform 622 and a refreshed efficiency

list for a specific platform 624. In at least one embodiment, a refreshed efficiency list for a specific platform is one or a combination of efficiency list(s) for specific platform 422a-f of FIG. 4. In at least one embodiment, a refreshed efficiency list for a specific platform is based on performance metrics collected during previous dry runs, as described further herein at least in conjunction with FIGS. 1 and 4. In at least one embodiment, a refreshed efficiency list for a specific platform is based on a comparison of calibrated GPU hardware platform parameters with GPU hardware platform parameters used during dry runs and recorded in efficiency lists.

[0129] In at least one embodiment, a refreshed efficiency list for a specific platform is used to select which masks to remove, retain, modify, or some combination thereof, in an efficient neural model to be deployed on a specific GPU hardware platform. In at least one embodiment, efficient neural models to be deployed on specific GPU hardware platforms include efficient neural model for deployment (M_{ED}) 660 and efficient neural model for deployment (M_{ED}) 662. In at least one embodiment, efficient neural model for deployment (M_{ED}) 660 and efficient neural model for deployment (M_{ED}) 662 are modified versions of masked super neural model 550 of FIG. 5, where such modifications include masks that have been retained, removed, modified, or some combination thereof, based on refreshed efficiency lists for specific platforms.

[0130] In at least one embodiment, once masks have been retained, removed, modified, or some combination thereof, in efficient neural models for deployment, knowledge distillation is performed on those efficient neural models for deployment by using, at least in part, dense target neural model 612. In at least one embodiment, knowledge distillation further optimizes efficient neural model for deployment (M_{ED}) 660 and efficient neural model for deployment (M_{ED}) 662. In at least one embodiment, knowledge distillation uses input data 560 of FIG. 5. In at least one embodiment, further training of efficient neural models for deployment is performed prior to knowledge distillation.

[0131] In at least one embodiment, knowledge distillation includes calculations of loss functions 670, such as hard label distillation loss, soft logits distillation loss, and feature maps distillation loss. In at least one embodiment, hard label distillation loss uses a loss function that measures a difference between predictions of an efficient neural model for deployment and ground-truth labels of a dense neural model on which that efficient neural model is based, such as dense target neural model 612. In at least one embodiment, soft logits distillation loss uses a

loss function that measures a difference between probabilities of correct inferences of an efficient neural model and probabilities of correct inferences of a dense target neural model. In at least one embodiment, logits refer to raw, unnormalized output scores of a last layer of a classification model. In at least one embodiment, logits are input into an activation function to convert them into probabilities. In at least one embodiment, feature maps distillation loss uses a loss function that measures a difference between feature maps of an efficient neural model and feature maps of a dense target neural model. In at least one embodiment, feature maps distillation loss is based on mean-squared error, cross-entropy, focal loss, or some combination thereof. In at least one embodiment, once knowledge distillation has been performed on efficient neural model for deployment (M_{ED}) 660 and efficient neural model for deployment (M_{ED}) 662, those neural models are ready to be respectively deployed on GPU hardware platform 680 and GPU hardware platform 682.

[0132] FIG. 7 illustrates process 700 to performs operations that modify a neural model such that a specific GPU hardware platform will perform that neural model to meet and/or exceed a threshold performance metric, according to at least one embodiment. In at least one embodiment, one or more operations of process 700 are performed in a different order, combined, omitted, or some combination thereof. In at least one embodiment, one or more aspects of one or more embodiments described herein in conjunction with FIG. 7 are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-6, and 8B. In at least one embodiment, one or more processors perform one or more operations used in process 700, such inserting masks into a target neural model with operation 708. In at least one embodiment, one or more processors that perform one or more operations used as part of process 700 are any one processor, or combination of processors, described herein, including processor(s) 822 described in conjunction with FIG. 8B, accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, neural model dry runs module 102 of FIG. 1A performs one or more operations of process 700, including performing efficient variations of a target neural model with operation 702. In at least one embodiment, one or more residual block modifications 202 of FIG. 2 are implemented with operation 701. In at least one embodiment, one or more of self-attention block modifications 302 of FIG. 3 are implemented with operation

701. In at least one embodiment, fused efficiency lists module 430 performs one or more operations of process 700, including fusing efficiency lists with operation 706. In at least one embodiment, masking module 540 of FIG. 5 performs one or more operations of process 700, including inserting masks with operation 708. In at least one embodiment, an API of API(s) 810 of FIG. 8A causes operations of process 700 to be performed, including identifying masks to be removed from efficient neural models with operation 714. In at least one embodiment, neural model optimization module 824 performs one or more operations of process 700, including operations to remove masks in an efficient neural model for deployment with operation 714.

[0133] In at least one embodiment, a processor begins process 700 by generating efficient variations of a target neural model with operation 701. In at least one embodiment, an efficient variation of a neural model is a neural model that has been modified using optimization techniques, which are described further herein at least in conjunction with FIGS. 1A-4. In at least one embodiment, generating efficient variations of a target neural model is described further herein at least in conjunction with FIGS. 1 and 4. In at least one embodiment, variations of a target neural model are based on a dense target neural model, such as dense target neural model 412 of FIG. 4.

[0134] In at least one embodiment, a processor continues process 700 by performing efficient variations of a target neural model on various GPU hardware platforms, which is described further herein at least in conjunction with FIGS. 1 and 4. In at least one embodiment, operation 701 includes a processor that performs a dry run on a dense target neural model on various GPU hardware platforms. In at least one embodiment, a processor collects deployment efficiency indicators (metrics), including per-layer performance, acceleration ratios with a low-precision data format, measures that indicate improvements caused by sparse compression, cross-layer overheads, and indications of performance improvements as a result of operator and/or layer fusion.

[0135] In at least one embodiment, a processor continues process 700 by obtaining an efficiency list for each GPU hardware platform with operation 704. In at least one embodiment, efficiency lists for each GPU hardware platform is used to identify appropriate positions within a neural model to insert masks. In at least one embodiment, efficiency lists for each GPU hardware platform are generated by efficiency lists module 410 of FIG. 4.

[0136] In at least one embodiment, a processor continues process 700 by fusing efficiency lists for each GPU hardware platform with operation 706. In at least one embodiment, fused efficiency lists module 430 of FIG. 4 performs one or more aspects of operation 706. In at least one embodiment, items in efficiency lists, such as performance metrics and optimization techniques, are fused to determine which optimization operations and combinations of those operations will produce performance benefits when an efficient neural model is deployed on a specific GPU hardware platform. In at least one embodiment, common items across efficiency lists for different GPU hardware platforms are combined. In at least one embodiment, combining common items occurs when a specific quantization method works equally as well on a neural model on two different GPU hardware platforms, and those two GPU hardware platforms are included in a single fused efficiency list entry that indicates that quantization method and a performance metric.

[0137] In at least one embodiment, a processor continues process 700 with operation 708 by inserting data conversion and sparse pruning masks into a dense target neural model to generate a masked super neural model such as masked super neural model 550 of FIG. 5. In at least one embodiment, masking module 540 of FIG. 5 performs one or more aspects of operation 708. In at least one embodiment, identifying locations in which to insert masks with operation 708 is based on a fused efficiency list generated with operation 706, where inserting masks is further described herein at least in conjunction with FIG. 5.

[0138] In at least one embodiment, a processor continues process 700 with operation 710 by training a masked super neural model such as masked super neural model 550 of FIG. 5. In at least one embodiment, masked super neural model 550 of FIG. 5 is trained to converge by simultaneously using data conversion and sparse pruning masks inserted with operation 708. In at least one embodiment, training masked super neural model 550 of FIG. 5 to converge improves its accuracy, which was likely negatively impacted by insertion of masks.

[0139] In at least one embodiment, a processor continues process 700 with operation 712 by calibrating a GPU hardware platform and generating a refreshed efficiency list of a specific deployment platform, which is described further herein at least in conjunction with FIG. 6. In at least one embodiment, calibration module 620 of FIG. 6 performs one or more aspects of operation 712.

[0140] In at least one embodiment, a processor continues process 700 by identifying masks to retain, remove, modify, or some combination thereof, with operation 714. In at least one embodiment, a processor uses a refreshed efficiency list of operation 712 to identify masks with operation 714. In at least one embodiment, masked super neural model 550 of FIG. 5, at least conceptually, initializes weight parameters of an efficient neural model for deployment, such as efficient neural model for deployment 660 and efficient neural model for deployment 662 of FIG. 6.

[0141] In at least one embodiment, a processor continues process 700 by performing knowledge distillation on an efficient neural model for deployment with operation 716. In at least one embodiment, operation 716 causes knowledge distillation using hard labels, soft logits, and feature maps. In at least one embodiment, knowledge distillation causes, at least conceptually, a transfer of knowledge of a dense target neural model to improve accuracy of an efficient neural model for deployment, where knowledge distillation is described further herein at least in conjunction with FIG. 6. In at least one embodiment, process 700 ends after knowledge distillation of an efficient neural model for deployment because that neural model has been tailored for, and is ready to be performed by, a specific GPU hardware platform.

[0142] FIG. 8A illustrates a block diagram of a driver and/or runtime comprising one or more libraries to provide one or more application programming interfaces (APIs), according to at least one embodiment. In at least one embodiment, any one processor, or combination of processors, perform API(s) 810, including processor(s) 822 of FIG. 8B, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, API(s) 810 are described further herein. In at least one embodiment, an invocation of API(s) 810 cause any one or more operations of any one or more modules of FIGS. 1 and 4-6 to be performed. In at least one embodiment, API(s) 810 receives as input, a dense target neural model 412 of FIG. 4, or an indication thereof, and causes neural model dry runs module 102 of FIG. 1A to perform operations that generate different, efficient variations of a target neural model as described further herein. In at least one embodiment, API(s) 810 take two or more efficiency lists as input, or indications thereof, and outputs a fused efficiency list as described further herein at least in conjunction with FIG. 4. In at least one embodiment, API(s) 810 is implemented as part of masking module 540 of FIG. 5 and takes, as

inputs, a fused efficiency list and a target neural model, or indications thereof, which causes masks to be inserted into that target neural model as described further herein at least in conjunction with FIG. 5. In at least one embodiment, API(s) 810 takes masked super neural model 550 of FIG. 5, or indications thereof, as input and causes modifications to that masked super neural model to create an efficient neural model for deployment.

[0143] In at least one embodiment, a software program 802 is a software module. In at least one embodiment, a software program 802 comprises one or more software modules. In at least one embodiment, one or more APIs 810 are sets of software instructions that, if executed, cause one or more processors to perform one or more computational operations. In at least one embodiment, one or more APIs 810 are distributed or otherwise provided as a part of one or more libraries 806, runtimes 804, drivers 804, and/or any other grouping of software and/or executable code further described herein. In at least one embodiment, one or more APIs 810 perform one or more computational operations in response to invocation by software programs 802. In at least one embodiment, a software program 802 is a collection of software code, commands, instructions, or other sequences of text to instruct a computing device to perform one or more computational operations and/or invoke one or more other sets of instructions, such as APIs 810 or API function(s) to cause modification of neural network(s) using masks 812, to be executed. In at least one embodiment, API function(s) to cause modification of neural network(s) using masks 312 include one or more operations and/or functions used to modify a neural network using masks, such as operations and/or functions described in conjunction with masking module 540 of FIG. 5 and operations and/or functions as otherwise described herein at least in conjunction with FIGS. 1A-7 In at least one embodiment, functionality provided by one or more APIs 810 include software functions 812, such as those usable to accelerate one or more portions of software programs 802 using one or more parallel processing units (PPUs), such as graphics processing units (GPUs). In at least one embodiment, a software program is a compiler.

[0144] In at least one embodiment, APIs 810 are hardware interfaces to one or more circuits to perform one or more computational operations. In at least one embodiment, one or more software APIs 810 described herein are implemented as one or more circuits to perform one or more techniques described herein. In at least one embodiment, one or more software programs

802 comprise instructions that, if executed, cause one or more hardware devices and/or circuits to perform one or more techniques further described herein.

[0145] In at least one embodiment, software programs 802, such as user-implemented software programs, utilize one or more application programming interfaces (APIs) 810 to perform various computing operations, such as memory reservation, matrix multiplication, arithmetic operations, or any computing operation performed by parallel processing units (PPUs), such as graphics processing units (GPUs), as further described herein. In at least one embodiment, one or more APIs 810 provide a set of callable functions 812, referred to herein as APIs, API functions, and/or functions, that individually perform one or more computing operations, such as computing operations related to parallel computing. For example, in an embodiment, one or more APIs 810 provide functions 812 to cause a scheduler to schedule instructions to be performed by processors based on latency of interconnects coupled to these processors.

[0146] In at least one embodiment, one or more software programs 802 interact or otherwise communicate with one or more APIs 810 to perform one or more computing operations using one or more PPU, such as GPUs. In at least one embodiment, one or more computing operations using one or more PPU comprise at least one or more groups of computing operations to be accelerated by execution at least in part by said one or more PPU. In at least one embodiment, one or more software programs 802 interact with one or more APIs 810 to facilitate parallel computing using a remote or local interface.

[0147] In at least one embodiment, an interface is software instructions that, if executed, provide access to one or more functions 812 provided by one or more APIs 810. In at least one embodiment, a software program 802 uses a local interface when a software developer compiles one or more software programs 802 in conjunction with one or more libraries 806 comprising or otherwise providing access to one or more APIs 810. In at least one embodiment, one or more software programs 802 are compiled statically in conjunction with pre-compiled libraries 806 or uncompiled source code comprising instructions to perform one or more APIs 810. In at least one embodiment, one or more software programs 802 are compiled dynamically and said one or more software programs utilize a linker to link to one or more pre-compiled libraries 806 comprising one or more APIs 810.

[0148] In at least one embodiment, a software program 802 uses a remote interface when a software developer executes a software program that utilizes or otherwise communicates with a library 806 comprising one or more APIs 810 over a network or other remote communication medium. In at least one embodiment, one or more libraries 806 comprising one or more APIs 810 are to be performed by a remote computing service, such as a computing resource services provider. In another embodiment, one or more libraries 806 comprising one or more APIs 810 are to be performed by any other computing host providing said one or more APIs 810 to one or more software programs 802.

[0149] In at least one embodiment, a processor performing or using one or more software programs 802 calls, uses, performs, or otherwise implements one or more APIs 810 to allocate and otherwise manage memory to be used by said software programs 802. In at least one embodiment, one or more software programs 802 utilize one or more APIs 810 to allocate and otherwise manage memory to be used by one or more portions of said software programs 802 to be accelerated using one or more PPUs, such as GPUs or any other accelerator or processor further described herein. Those software programs 802 may be performed by one or more processors based, at least in part, on latency of interconnects coupled to one or more processors using functions 812 provided, in an embodiment, by one or more APIs 810.

[0150] In at least one embodiment, an API 810 is an API to facilitate parallel computing. In at least one embodiment, an API 810 is any other API further described herein. In at least one embodiment, an API 810 is provided by a driver and/or runtime 804. In at least one embodiment, an API 810 is provided by a CUDA user-mode driver. In at least one embodiment, an API 810 is provided by a CUDA runtime. In at least one embodiment, a driver 804 is data values and software instructions that, if executed, perform or otherwise facilitate operation of one or more functions 812 of an API 810 during load and execution of one or more portions of a software program 802. In at least one embodiment, a runtime 804 is data values and software instructions that, if executed, perform or otherwise facilitate operation of one or more functions 812 of an API 810 during execution of a software program 802. In at least one embodiment, one or more software programs 802 utilize one or more APIs 810 implemented or otherwise provided by a driver and/or runtime 804 to perform combined arithmetic operations by said one or more software programs 802 during execution by one or more PPUs, such as GPUs.

[0151] In at least one embodiment, one or more software programs 802 utilize one or more APIs 810 provided by a driver and/or runtime 804 to perform combined arithmetic operations of one or more PPUs, such as GPUs. In at least one embodiment, one or more APIs 810 provide combined arithmetic operations through a driver and/or runtime 804, as described above. In at least one embodiment, one or more software programs 802 utilize one or more APIs 810 provided by a driver and/or runtime 804 to allocate or otherwise reserve one or more blocks of memory 814 of one or more PPUs, such as GPUs. In at least one embodiment, one or more software programs 802 utilize one or more APIs 810 provided by a driver and/or runtime 804 to allocate or otherwise reserve blocks of memory. In at least one embodiment, one or more APIs 810 are to perform combined mathematical functions as described herein.

[0152] In at least one embodiment, to improve software programs 802 usability and/or optimization of one or more portions of said software programs 802 to be accelerated by one or more PPUs, such as GPUs, one or more APIs 810 provide one or more API functions 812 to perform a scheduling system usable or used by one or more computing devices as described herein. In at least one embodiment, a processor performs one or more software programs to combine two or more application programming interfaces (APIs) into a single API. In at least one embodiment, a processor uses an API to cause a scheduler to select a thread selection mechanism and/or otherwise perform operations described herein. In at least one embodiment, an API invokes a scheduler to cause a resource allocation. In at least one embodiment, a processor uses an exemplary API to schedule one or more instructions to be performed by one or more processors based, at least in part, on latency of one or more interconnects coupled to these one or more processors.

[0153] In at least one embodiment, memory 814 is system memory 1904 of computing system 1900. In at least one embodiment, memory 814 stores data parameters, such as baseband and RF weights of one or more signals being transmitted. In at least one embodiment, memory 814 stores data parameters used by various modules described herein, modules such as neural model dry runs module 102 of FIG. 1, efficiency lists module 210 of FIG. 2, efficiency lists module 310 of FIG. 3, efficiency lists module 410 of FIG. 4, and masking module 540 of FIG. 5, and calibration module 620 of FIG. 6. In at least one embodiment, memory 814 stores SINR values computed using operation 608.

[0154] In at least one embodiment, memory 814 is a computer readable storage medium and/or code stored on said computer readable storage medium in a form of a computer program including a plurality of computer readable instructions executable by one or more processors. In at least one embodiment, a computer readable storage medium is a non-transitory computer readable medium. In at least one embodiment, at least some computer readable instructions usable to perform operations described in relation to FIG. 1A are not stored solely using transitory signals (e.g., a propagating transient electric or electromagnetic transmission). In at least one embodiment, a non-transitory computer readable medium does not necessarily include non-transitory data storage circuitry (e.g., buffers, caches, and queues) within transceivers of transitory signals. In at least one embodiment, memory 814 is implemented as a non-transitory computer readable storage medium storing executable instructions that, if executed by one or more processors of a computer system, cause said computer system to insert masks into a neural model to be performed on a specific GPU hardware platform as described further herein at least in conjunction with FIGS. 1A-8A.

[0155] FIG. 8B illustrates block diagram 800 that includes processor(s) 822 used to perform any one or more operations of one or more modules and/or processes described herein at least in conjunction with FIGS. 1A-7, according to at least one embodiment. In at least one embodiment, processor(s) 822 are used to perform modification of a dense target neural model as described further herein. In at least one embodiment, one or more aspects of one or more embodiments described in conjunction with FIG. 8B are combined with one or more aspects of one or more embodiments described herein, including those described at least in conjunction with FIGS. 1A-8A. In at least one embodiment, processor(s) 822 is any one processor, or combination of processors, described herein, including accelerator(s) 1214 described in conjunction with FIG. 12C, graphics processor 1910 described in conjunction with FIG. 19A, and parallel processing unit (“PPU”) 3400 described in conjunction with FIG. 34. In at least one embodiment, processor(s) 822 is processor(s) 1902 of computing system 1900.

[0156] In at least one embodiment, one or more modules described herein, including those described at least in conjunction with FIGS. 1A-6, are installed on processor(s) 822. In at least one embodiment, an exemplary module on processor(s) 822 is neural model optimization module 824. In at least one embodiment, neural model optimization module 824 performs any one or

more operations that optimize a dense target neural model as described further herein at least in conjunction with FIG. 1-7. In at least one embodiment, neural model optimization module 824 includes one or more of API(s) 810 of FIG. 8A to perform one or more operations of related to modifying a dense target neural model, such as generating different variations of that neural model as described further herein at least in conjunction with FIG. 1.

[0157] In at least one embodiment, an exemplary module on processor(s) 822 is efficient neural network modification module 826. In at least one embodiment, efficient neural network modification module 826 performs any operation used to retain, remove, modify, or some combination thereof, layers of an efficient neural model and as described further herein at least in conjunction with FIG. 6. In at least one embodiment, efficient neural network modification module 826 performs any operation used in knowledge distillation as described further herein at least in conjunction with FIG. 6. In at least one embodiment, efficient neural network modification module 824 includes one or more of API(s) 810 of FIG. 8A to perform one or more operations related to modifying an efficient neural model such as retaining, removing, modifying, or some combination thereof, layers based on a refreshed efficiency list for a specific GPU hardware platform as described herein at least in conjunction with FIG. 6.

LOGIC

[0158] FIG. 9A illustrates logic 915 which, as described elsewhere herein, can be used in one or more devices to perform operations such as those discussed herein in accordance with at least one embodiment. In at least one embodiment, logic 915 is used to perform inferencing and/or training operations associated with one or more embodiments. In at least one embodiment, logic 915 is inference and/or training logic. Details regarding logic 915 are provided below in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic refers to any combination of software logic, hardware logic, and/or firmware logic to provide functionality or operations described herein, wherein logic may be, collectively or individually, embodied as circuitry that forms part of a larger system, for example, an integrated circuit (IC), system-on-chip (SoC), or one or processors (e.g., CPU, GPU).

[0159] In at least one embodiment, logic 915 may include, without limitation, code and/or data storage 901 to store forward and/or output weight and/or input/output data, and/or other parameters to configure neurons or layers of a neural network trained and/or used for inferencing

in aspects of one or more embodiments. In at least one embodiment, logic 915 may include, or be coupled to code and/or data storage 901 to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs)). In at least one embodiment, code, such as graph code, loads weight or other parameter information into processor ALUs based on an architecture of a neural network to which such code corresponds. In at least one embodiment, code and/or data storage 901 stores weight parameters and/or input/output data of each layer of a neural network trained or used in conjunction with one or more embodiments during forward propagation of input/output data and/or weight parameters during training and/or inferencing using aspects of one or more embodiments. In at least one embodiment, any portion of code and/or data storage 901 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0160] In at least one embodiment, any portion of code and/or data storage 901 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or code and/or data storage 901 may be cache memory, dynamic randomly addressable memory ("DRAM"), static randomly addressable memory ("SRAM"), non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, a choice of whether code and/or code and/or data storage 901 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

[0161] In at least one embodiment, logic 915 may include, without limitation, a code and/or data storage 905 to store backward and/or output weight and/or input/output data corresponding to neurons or layers of a neural network trained and/or used for inferencing in aspects of one or more embodiments. In at least one embodiment, code and/or data storage 905 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, logic 915 may include, or be coupled to code and/or data storage 905 to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs)).

[0162] In at least one embodiment, code, such as graph code, causes the loading of weight or other parameter information into processor ALUs based on an architecture of a neural network to which such code corresponds. In at least one embodiment, any portion of code and/or data storage 905 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory. In at least one embodiment, any portion of code and/or data storage 905 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or data storage 905 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, a choice of whether code and/or data storage 905 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash memory or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

[0163] In at least one embodiment, code and/or data storage 901 and code and/or data storage 905 may be separate storage structures. In at least one embodiment, code and/or data storage 901 and code and/or data storage 905 may be a combined storage structure. In at least one embodiment, code and/or data storage 901 and code and/or data storage 905 may be partially combined and partially separate. In at least one embodiment, any portion of code and/or data storage 901 and code and/or data storage 905 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0164] In at least one embodiment, logic 915 may include, without limitation, one or more arithmetic logic unit(s) ("ALU(s)") 910, including integer and/or floating point units, to perform logical and/or mathematical operations based, at least 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 920 that are functions of input/output and/or weight parameter data stored in code and/or data storage 901

and/or code and/or data storage 905. In at least one embodiment, activations stored in activation storage 920 are generated according to linear algebraic and or matrix-based mathematics performed by ALU(s) 910 in response to performing instructions or other code, wherein weight values stored in code and/or data storage 905 and/or data storage 901 are used as operands along with other values, such as bias values, gradient information, momentum values, or other parameters or hyperparameters, any or all of which may be stored in code and/or data storage 905 or code and/or data storage 901 or another storage on or off-chip.

[0165] In at least one embodiment, ALU(s) 910 are included within one or more processors or other hardware logic devices or circuits, whereas in another embodiment, ALU(s) 910 may be external to a processor or other hardware logic device or circuit that uses them (e.g., a co-processor). In at least one embodiment, ALUs 910 may be included within a processor's execution units or otherwise within a bank of ALUs accessible by a processor's execution units either within same processor or distributed between different processors of different types (e.g., central processing units, graphics processing units, fixed function units, etc.). In at least one embodiment, code and/or data storage 901, code and/or data storage 905, and activation storage 920 may share a processor or other hardware logic device or circuit, whereas in another embodiment, they may be in different processors or other hardware logic devices or circuits, or some combination of same and different processors or other hardware logic devices or circuits. In at least one embodiment, any portion of activation storage 920 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, 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.

[0166] In at least one embodiment, activation storage 920 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, activation storage 920 may be completely or partially within or external to one or more processors or other logical circuits. In at least one embodiment, a choice of whether activation storage 920 is internal or external to a processor, for example, or comprising 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.

[0167] In at least one embodiment, logic 915 illustrated in FIG. 9A may be used in conjunction with an application-specific integrated circuit (“ASIC”), such as a TensorFlow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, logic 915 illustrated in FIG. 9A 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”).

[0168] FIG. 9B illustrates logic 915, according to at least one embodiment. In at least one embodiment, logic 915 is inference and/or training logic. In at least one embodiment, logic 915 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, logic 915 illustrated in FIG. 9B may be used in conjunction with an application-specific integrated circuit (ASIC), such as TensorFlow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, logic 915 illustrated in FIG. 9B may be used in conjunction with central processing unit (CPU) hardware, graphics processing unit (GPU) hardware or other hardware, such as field programmable gate arrays (FPGAs). In at least one embodiment, logic 915 includes, without limitation, code and/or data storage 901 and code and/or data storage 905, which may be used to store code (e.g., graph code), weight values and/or other information, including bias values, gradient information, momentum values, and/or other parameter or hyperparameter information. In at least one embodiment illustrated in FIG. 9B, each of code and/or data storage 901 and code and/or data storage 905 is associated with a dedicated computational resource, such as computational hardware 902 and computational hardware 906, respectively. In at least one embodiment, each of computational hardware 902 and computational hardware 906 comprises one or more ALUs that perform mathematical functions, such as linear algebraic functions, only on information stored in code and/or data storage 901 and code and/or data storage 905, respectively, result of which is stored in activation storage 920.

[0169] In at least one embodiment, each of code and/or data storage 901 and 905 and corresponding computational hardware 902 and 906, respectively, correspond to different layers of a neural network, such that resulting activation from one storage/computational pair 901/902 of code and/or data storage 901 and computational hardware 902 is provided as an input to a next storage/computational pair 905/906 of code and/or data storage 905 and computational hardware 906, in order to mirror a conceptual organization of a neural network. In at least one embodiment, each of storage/computational pairs 901/902 and 905/906 may correspond to more than one neural network layer. In at least one embodiment, additional storage/computation pairs (not shown) subsequent to or in parallel with storage/computation pairs 901/902 and 905/906 may be included in logic 915.

NEURAL NETWORK TRAINING AND DEPLOYMENT

[0170] FIG. 10 illustrates training and deployment of a deep neural network, according to at least one embodiment. In at least one embodiment, untrained neural network 1006 is trained using a training dataset 1002. In at least one embodiment, training framework 1004 is a PyTorch framework, whereas in other embodiments, training framework 1004 is a TensorFlow, Boost, Caffe, Microsoft Cognitive Toolkit/CNTK, MXNet, Chainer, Keras, Deeplearning4j, or other training framework. In at least one embodiment, training framework 1004 trains an untrained neural network 1006 and enables it to be trained using processing resources described herein to generate a trained neural network 1008. In at least one embodiment, weights may be chosen randomly or by pre-training using a deep belief network. In at least one embodiment, training may be performed in either a supervised, partially supervised, or unsupervised manner.

[0171] In at least one embodiment, untrained neural network 1006 is trained using supervised learning, wherein training dataset 1002 includes an input paired with a desired output for an input, or where training dataset 1002 includes input having a known output and an output of neural network 1006 is manually graded. In at least one embodiment, untrained neural network 1006 is trained in a supervised manner and processes inputs from training dataset 1002 and compares resulting outputs against a set of expected or desired outputs. In at least one embodiment, errors are then propagated back through untrained neural network 1006. In at least one embodiment, training framework 1004 adjusts weights that control untrained neural network 1006. In at least one embodiment, training framework 1004 includes tools to monitor how well

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

[0172] In at least one embodiment, untrained neural network 1006 is trained using unsupervised learning, wherein untrained neural network 1006 attempts to train itself using unlabeled data. In at least one embodiment, unsupervised learning training dataset 1002 will include input data without any associated output data or “ground truth” data. In at least one embodiment, untrained neural network 1006 can learn groupings within training dataset 1002 and can determine how individual inputs are related to untrained dataset 1002. In at least one embodiment, unsupervised training can be used to generate a self-organizing map in trained neural network 1008 capable of performing operations useful in reducing dimensionality of new dataset 1012. In at least one embodiment, unsupervised training can also be used to perform anomaly detection, which allows identification of data points in new dataset 1012 that deviate from normal patterns of new dataset 1012.

[0173] In at least one embodiment, semi-supervised learning may be used, which is a technique in which in training dataset 1002 includes a mix of labeled and unlabeled data. In at least one embodiment, training framework 1004 may be used to perform incremental learning, such as through transferred learning techniques. In at least one embodiment, incremental learning enables trained neural network 1008 to adapt to new dataset 1012 without forgetting knowledge instilled within trained neural network 1008 during initial training.

[0174] In at least one embodiment, training framework 1004 is a framework processed in connection with a software development toolkit such as an OpenVINO (Open Visual Inference and Neural network Optimization) toolkit. In at least one embodiment, an OpenVINO toolkit is a toolkit such as those developed by Intel Corporation of Santa Clara, CA. In at least one

embodiment, OpenVINO comprises logic 915 or uses logic 915 to perform operations described herein. In at least one embodiment, an SoC, integrated circuit, or processor uses OpenVINO to perform operations described herein.

[0175] In at least one embodiment, OpenVINO is a toolkit for facilitating development of applications, specifically neural network applications, for various tasks and operations, such as human vision emulation, speech recognition, natural language processing, recommendation systems, and/or variations thereof. In at least one embodiment, OpenVINO supports neural networks such as convolutional neural networks (CNNs), recurrent and/or attention-based neural networks, and/or various other neural network models. In at least one embodiment, OpenVINO supports various software libraries such as OpenCV, OpenCL, and/or variations thereof.

[0176] In at least one embodiment, OpenVINO supports neural network models for various tasks and operations, such as classification, segmentation, object detection, face recognition, speech recognition, pose estimation (e.g., humans and/or objects), monocular depth estimation, image inpainting, style transfer, action recognition, colorization, and/or variations thereof.

[0177] In at least one embodiment, OpenVINO comprises one or more software tools and/or modules for model optimization, also referred to as a model optimizer. In at least one embodiment, a model optimizer is a command line tool that facilitates transitions between training and deployment of neural network models. In at least one embodiment, a model optimizer optimizes neural network models for execution on various devices and/or processing units, such as a GPU, CPU, PPU, GPGPU, and/or variations thereof. In at least one embodiment, a model optimizer generates an internal representation of a model, and optimizes said model to generate an intermediate representation. In at least one embodiment, a model optimizer reduces a number of layers of a model. In at least one embodiment, a model optimizer removes layers of a model that are utilized for training. In at least one embodiment, a model optimizer performs various neural network operations, such as modifying inputs to a model (e.g., resizing inputs to a model), modifying a size of inputs of a model (e.g., modifying a batch size of a model), modifying a model structure (e.g., modifying layers of a model), normalization, standardization, quantization (e.g., converting weights of a model from a first representation, such as floating point, to a second representation, such as integer), and/or variations thereof.

[0178] In at least one embodiment, OpenVINO comprises one or more software libraries for inferencing, also referred to as an inference engine. In at least one embodiment, an inference engine is a C++ library, or any suitable programming language library. In at least one embodiment, an inference engine is utilized to infer input data. In at least one embodiment, an inference engine implements various classes to infer input data and generate one or more results. In at least one embodiment, an inference engine implements one or more API functions to process an intermediate representation, set input and/or output formats, and/or execute a model on one or more devices.

[0179] In at least one embodiment, OpenVINO provides various abilities for heterogeneous execution of one or more neural network models. In at least one embodiment, heterogeneous execution, or heterogeneous computing, refers to one or more computing processes and/or systems that utilize one or more types of processors and/or cores. In at least one embodiment, OpenVINO provides various software functions to execute a program on one or more devices. In at least one embodiment, OpenVINO provides various software functions to execute a program and/or portions of a program on different devices. In at least one embodiment, OpenVINO provides various software functions to, for example, run a first portion of code on a CPU and a second portion of code on a GPU and/or FPGA. In at least one embodiment, OpenVINO provides various software functions to execute one or more layers of a neural network on one or more devices (e.g., a first set of layers on a first device, such as a GPU, and a second set of layers on a second device, such as a CPU).

[0180] In at least one embodiment, OpenVINO includes various functionality similar to functionalities associated with a CUDA programming model, such as various neural network model operations associated with frameworks such as TensorFlow, PyTorch, and/or variations thereof. In at least one embodiment, one or more CUDA programming model operations are performed using OpenVINO. In at least one embodiment, various systems, methods, and/or techniques described herein are implemented using OpenVINO.

DATA CENTER

[0181] FIG. 11 illustrates an example data center 1100, in which at least one embodiment may be used. In at least one embodiment, data center 1100 includes a data center infrastructure layer 1110, a framework layer 1120, a software layer 1130 and an application layer 1140.

[0182] In at least one embodiment, as shown in FIG. 11, data center infrastructure layer 1110 may include a resource orchestrator 1112, grouped computing resources 1114, and node computing resources (“node C.R.s”) 1116(1)-1116(N), where “N” represents a positive integer (which may be a different integer “N” than used in other figures). In at least one embodiment, node C.R.s 1116(1)-1116(N) may include, but are not limited to, any number of central processing units (“CPUs”) or other processors (including accelerators, field programmable gate arrays (FPGAs), graphics processors, etc.), memory storage devices 1118(1)-1118(N) (e.g., dynamic read-only memory, solid state storage or disk drives), network input/output (“NW I/O”) devices, network switches, virtual machines (“VMs”), power modules, and cooling modules, etc. In at least one embodiment, one or more node C.R.s from among node C.R.s 1116(1)-1116(N) may be a server having one or more of above-mentioned computing resources.

[0183] In at least one embodiment, grouped computing resources 1114 may include separate groupings of node C.R.s housed within one or more racks (not shown), or many racks housed in data centers at various geographical locations (also not shown). In at least one embodiment, separate groupings of node C.R.s within grouped computing resources 1114 may include grouped compute, network, memory or storage resources that may be configured or allocated to support one or more workloads. In at least one embodiment, several node C.R.s including CPUs or processors may grouped within one or more racks to provide compute resources to support one or more workloads. In at least one embodiment, one or more racks may also include any number of power modules, cooling modules, and network switches, in any combination.

[0184] In at least one embodiment, resource orchestrator 1112 may configure or otherwise control one or more node C.R.s 1116(1)-1116(N) and/or grouped computing resources 1114. In at least one embodiment, resource orchestrator 1112 may include a software design infrastructure (“SDI”) management entity for data center 1100. In at least one embodiment, resource orchestrator 912 may include hardware, software or some combination thereof.

[0185] In at least one embodiment, as shown in FIG. 11, framework layer 1120 includes a job scheduler 1122, a configuration manager 1124, a resource manager 1126 and a distributed file system 1128. In at least one embodiment, framework layer 1120 may include a framework to support software 1132 of software layer 1130 and/or one or more application(s) 1142 of application layer 1140. In at least one embodiment, software 1132 or application(s) 1142 may

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

[0186] In at least one embodiment, software 1132 included in software layer 1130 may include software used by at least portions of node C.R.s 1116(1)-1116(N), grouped computing resources 1114, and/or distributed file system 1128 of framework layer 1120. In at least one embodiment, one or more types of software may include, but are not limited to, Internet web page search software, e-mail virus scan software, database software, and streaming video content software.

[0187] In at least one embodiment, application(s) 1142 included in application layer 1140 may include one or more types of applications used by at least portions of node C.R.s 1116(1)-1116(N), grouped computing resources 1114, and/or distributed file system 1128 of framework layer 1120. In at least one embodiment, one or more types of applications may include, but are not limited to, any number of a genomics application, a cognitive compute, 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.

[0188] In at least one embodiment, any of configuration manager 1124, resource manager 1126, and resource orchestrator 1112 may implement any number and type of self-modifying

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

[0189] In at least one embodiment, data center 1100 may include tools, services, software or other resources to train one or more machine learning models or predict or infer information using one or more machine learning models according to one or more embodiments described herein. For example, in at least one embodiment, a machine learning model may be trained by calculating weight parameters according to a neural network architecture using software and computing resources described above with respect to data center 1100. In at least one embodiment, trained machine learning models corresponding to one or more neural networks may be used to infer or predict information using resources described above with respect to data center 1100 by using weight parameters calculated through one or more training techniques described herein.

[0190] 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.

[0191] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in data center 1100 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.

[0192] In at least one embodiment, at least one component shown or described with respect to FIGS. 9A-11 is used to implement techniques and/or functions described in connection with FIGS. 1A-6. In at least one embodiment, logic 915 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient

neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

AUTONOMOUS VEHICLE

[0193] FIG. 12A illustrates an example of an autonomous vehicle 1200, according to at least one embodiment. In at least one embodiment, autonomous vehicle 1200 (alternatively referred to herein as “vehicle 1200”) 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 1200 may be a semi-tractor-trailer truck used for hauling cargo. In at least one embodiment, vehicle 1200 may be an airplane, robotic vehicle, or other kind of vehicle.

[0194] Autonomous vehicles may be described in terms of automation levels, defined by National Highway Traffic Safety Administration (“NHTSA”), a division of US Department of Transportation, and Society of Automotive Engineers (“SAE”) “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles” (e.g., Standard No. J3016-201806, published on June 15, 2018, Standard No. J3016-201609, published on September 30, 2016, and previous and future versions of this standard). In at least one embodiment, vehicle 1200 may be capable of functionality in accordance with one or more of Level 1 through Level 5 of autonomous driving levels. For example, in at least one embodiment, vehicle 1200 may be capable of conditional automation (Level 3), high automation (Level 4), and/or full automation (Level 5), depending on embodiment.

[0195] In at least one embodiment, vehicle 1200 may include, without limitation, components such as a chassis, a vehicle body, wheels (e.g., 2, 4, 6, 8, 18, etc.), tires, axles, and other components of a vehicle. In at least one embodiment, vehicle 1200 may include, without limitation, a propulsion system 1250, such as an internal combustion engine, hybrid electric power plant, an all-electric engine, and/or another propulsion system type. In at least one embodiment, propulsion system 1250 may be connected to a drive train of vehicle 1200, which may include, without limitation, a transmission, to enable propulsion of vehicle 1200. In at least one embodiment, propulsion system 1250 may be controlled in response to receiving signals from a throttle/accelerator(s) 1252.

[0196] In at least one embodiment, a steering system 1254, which may include, without limitation, a steering wheel, is used to steer vehicle 1200 (e.g., along a desired path or route) when propulsion system 1250 is operating (e.g., when vehicle 1200 is in motion). In at least one embodiment, steering system 1254 may receive signals from steering actuator(s) 1256. In at least one embodiment, a steering wheel may be optional for full automation (Level 5) functionality. In at least one embodiment, a brake sensor system 1246 may be used to operate vehicle brakes in response to receiving signals from brake actuator(s) 1248 and/or brake sensors.

[0197] In at least one embodiment, controller(s) 1236, which may include, without limitation, one or more system on chips (“SoCs”) (not shown in FIG. 12A) and/or graphics processing unit(s) (“GPU(s)”), provide signals (e.g., representative of commands) to one or more components and/or systems of vehicle 1200. For instance, in at least one embodiment, controller(s) 1236 may send signals to operate vehicle brakes via brake actuator(s) 1248, to operate steering system 1254 via steering actuator(s) 1256, to operate propulsion system 1250 via throttle/accelerator(s) 1252. In at least one embodiment, controller(s) 1236 may include one or more onboard (e.g., integrated) computing devices that process sensor signals, and output operation commands (e.g., signals representing commands) to enable autonomous driving and/or to assist a human driver in driving vehicle 1200. In at least one embodiment, controller(s) 1236 may include a first controller for autonomous driving functions, a second controller for functional safety functions, a third controller for artificial intelligence functionality (e.g., computer vision), a fourth controller for infotainment functionality, a fifth controller for redundancy in emergency conditions, and/or other controllers. In at least one embodiment, a single controller may handle two or more of above functionalities, two or more controllers may handle a single functionality, and/or any combination thereof.

[0198] In at least one embodiment, controller(s) 1236 provide signals for controlling one or more components and/or systems of vehicle 1200 in response to sensor data received from one or more sensors (e.g., sensor inputs). In at least one embodiment, sensor data may be received from, for example and without limitation, global navigation satellite systems (“GNSS”) sensor(s) 1258 (e.g., Global Positioning System sensor(s)), RADAR sensor(s) 1260, ultrasonic sensor(s) 1262, LIDAR sensor(s) 1264, inertial measurement unit (“IMU”) sensor(s) 1266 (e.g., accelerometer(s), gyroscope(s), a magnetic compass or magnetic compasses, magnetometer(s), etc.), microphone(s)

1296, stereo camera(s) 1268, wide-view camera(s) 1270 (e.g., fisheye cameras), infrared camera(s) 1272, surround camera(s) 1274 (e.g., 360 degree cameras), long-range cameras (not shown in FIG. 12A), mid-range camera(s) (not shown in FIG. 12A), speed sensor(s) 1244 (e.g., for measuring speed of vehicle 1200), vibration sensor(s) 1242, steering sensor(s) 1240, brake sensor(s) (e.g., as part of brake sensor system 1246), and/or other sensor types.

[0199] In at least one embodiment, one or more of controller(s) 1236 may receive inputs (e.g., represented by input data) from an instrument cluster 1232 of vehicle 1200 and provide outputs (e.g., represented by output data, display data, etc.) via a human-machine interface (“HMI”) display 1234, an audible annunciator, a loudspeaker, and/or via other components of vehicle 1200. 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. 12A)), location data (e.g., vehicle’s 1200 location, such as on a map), direction, location of other vehicles (e.g., an occupancy grid), information about objects and status of objects as perceived by controller(s) 1236, etc. For example, in at least one embodiment, HMI display 1234 may display information about presence of one or more objects (e.g., a street sign, caution sign, traffic light changing, etc.), and/or information about driving maneuvers vehicle has made, is making, or will make (e.g., changing lanes now, taking exit 34B in two miles, etc.).

[0200] In at least one embodiment, vehicle 1200 further includes a network interface 1224 which may use wireless antenna(s) 1226 and/or modem(s) to communicate over one or more networks. For example, in at least one embodiment, network interface 1224 may be capable of communication over Long-Term Evolution (“LTE”), Wideband Code Division Multiple Access (“WCDMA”), Universal Mobile Telecommunications System (“UMTS”), Global System for Mobile communication (“GSM”), IMT-CDMA Multi-Carrier (“CDMA2000”) networks, etc. In at least one embodiment, wireless antenna(s) 1226 may also enable communication between objects in environment (e.g., vehicles, mobile devices, etc.), using local area network(s), such as Bluetooth, Bluetooth Low Energy (“LE”), Z-Wave, ZigBee, etc., and/or low power wide-area network(s) (“LPWANs”), such as LoRaWAN, SigFox, etc. protocols.

[0201] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in vehicle 1200 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.

[0202] In at least one embodiment, at least one component shown or described with respect to FIG. 12A is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, wireless antenna(s) 1226 is used to transmit and/or receive one or more wireless signals to be used as inputs into an efficient neural model for deployment that assists with autonomous driving and was generated by inserting and/or removing one or more neural network masks as described in conjunction with FIG. 5, and as otherwise described herein.

[0203] FIG. 12B illustrates an example of camera locations and fields of view for autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, cameras and respective fields of view are one example embodiment and are not intended to be limiting. For instance, in at least one embodiment, additional and/or alternative cameras may be included and/or cameras may be located at different locations on vehicle 1200.

[0204] In at least one embodiment, camera types for cameras may include, but are not limited to, digital cameras that may be adapted for use with components and/or systems of vehicle 1200. In at least one embodiment, camera(s) may operate at automotive safety integrity level (“ASIL”) B and/or at another ASIL. In at least one embodiment, camera types may be capable of any image capture rate, such as 60 frames per second (fps), 1220 fps, 240 fps, etc., depending on embodiment. In at least one embodiment, cameras may be capable of using rolling shutters, global shutters, another type of shutter, or a combination thereof. In at least one embodiment, color filter array may include a red clear clear clear (“RCCC”) color filter array, a red clear clear blue (“RCCB”) color filter array, a red blue green clear (“RBGC”) color filter array, a Foveon X3 color filter array, a Bayer sensors (“RGGB”) color filter array, a monochrome sensor color filter array, and/or another type of color filter array. In at least one embodiment, clear pixel cameras, such as cameras with an RCCC, an RCCB, and/or an RBGC color filter array, may be used in an effort to increase light sensitivity.

[0205] 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.

[0206] In at least one embodiment, one or more cameras may be mounted in a mounting assembly, such as a custom designed (three-dimensional (“3D”) printed) assembly, in order to cut out stray light and reflections from within vehicle 1200 (e.g., reflections from dashboard reflected in windshield mirrors) which may interfere with camera image data capture abilities. With reference to wing-mirror mounting assemblies, in at least one embodiment, wing-mirror assemblies may be custom 3D printed so that a camera mounting plate matches a shape of a wing-mirror. In at least one embodiment, camera(s) may be integrated into wing-mirrors. In at least one embodiment, for side-view cameras, camera(s) may also be integrated within four pillars at each corner of a cabin.

[0207] In at least one embodiment, cameras with a field of view that include portions of an environment in front of vehicle 1200 (e.g., front-facing cameras) may be used for surround view, to help identify forward facing paths and obstacles, as well as aid in, with help of one or more of controller(s) 1236 and/or control SoCs, providing information critical to generating an occupancy grid and/or determining preferred vehicle paths. In at least one embodiment, front-facing cameras may be used to perform many similar ADAS functions as LIDAR, including, without limitation, emergency braking, pedestrian detection, and collision avoidance. In at least one embodiment, front-facing cameras may also be used for ADAS functions and systems including, without limitation, Lane Departure Warnings (“LDW”), Autonomous Cruise Control (“ACC”), and/or other functions such as traffic sign recognition.

[0208] In at least one embodiment, a variety of cameras may be used in a front-facing configuration, including, for example, a monocular camera platform that includes a CMOS (“complementary metal oxide semiconductor”) color imager. In at least one embodiment, a wide-view camera 1270 may be used to perceive objects coming into view from a periphery (e.g., pedestrians, crossing traffic or bicycles). Although only one wide-view camera 1270 is illustrated in FIG. 12B, in other embodiments, there may be any number (including zero) wide-view cameras on vehicle 1200. In at least one embodiment, any number of long-range camera(s)

1298 (e.g., a long-view stereo camera pair) may be used for depth-based object detection, especially for objects for which a neural network has not yet been trained. In at least one embodiment, long-range camera(s) 1298 may also be used for object detection and classification, as well as basic object tracking.

[0209] In at least one embodiment, any number of stereo camera(s) 1268 may also be included in a front-facing configuration. In at least one embodiment, one or more of stereo camera(s) 1268 may include an integrated control unit comprising a scalable processing unit, which may provide a programmable logic (“FPGA”) and a multi-core micro-processor with an integrated Controller Area Network (“CAN”) or Ethernet interface on a single chip. In at least one embodiment, such a unit may be used to generate a 3D map of an environment of vehicle 1200, including a distance estimate for all points in an image. In at least one embodiment, one or more of stereo camera(s) 1268 may include, without limitation, compact stereo vision sensor(s) that may include, without limitation, two camera lenses (one each on left and right) and an image processing chip that may measure distance from vehicle 1200 to target object and use generated information (e.g., metadata) to activate autonomous emergency braking and lane departure warning functions. In at least one embodiment, other types of stereo camera(s) 1268 may be used in addition to, or alternatively from, those described herein.

[0210] In at least one embodiment, cameras with a field of view that include portions of environment to sides of vehicle 1200 (e.g., side-view cameras) may be used for surround view, providing information used to create and update an occupancy grid, as well as to generate side impact collision warnings. For example, in at least one embodiment, surround camera(s) 1274 (e.g., four surround cameras as illustrated in FIG. 12B) could be positioned on vehicle 1200. In at least one embodiment, surround camera(s) 1274 may include, without limitation, any number and combination of wide-view cameras, fisheye camera(s), 360 degree camera(s), and/or similar cameras. For instance, in at least one embodiment, four fisheye cameras may be positioned on a front, a rear, and sides of vehicle 1200. In at least one embodiment, vehicle 1200 may use three surround camera(s) 1274 (e.g., left, right, and rear), and may leverage one or more other camera(s) (e.g., a forward-facing camera) as a fourth surround-view camera.

[0211] In at least one embodiment, cameras with a field of view that include portions of an environment behind vehicle 1200 (e.g., rear-view cameras) may be used for parking assistance,

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

[0212] In at least one embodiment, at least one component shown or described with respect to FIG. 12B is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, wireless antenna(s) 1226 are used to transmit and/or receive one or more wireless signals to be used as inputs into an efficient neural model for deployment that assists with autonomous driving and was generated by inserting and/or removing one or more neural network masks as described in conjunction with FIG. 5, and as otherwise described herein.

[0213] FIG. 12C is a block diagram illustrating an example system architecture for autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, each of components, features, and systems of vehicle 1200 in FIG. 12C is illustrated as being connected via a bus 1202. In at least one embodiment, bus 1202 may include, without limitation, a CAN data interface (alternatively referred to herein as a “CAN bus”). In at least one embodiment, a CAN may be a network inside vehicle 1200 used to aid in control of various features and functionality of vehicle 1200, such as actuation of brakes, acceleration, braking, steering, windshield wipers, etc. In at least one embodiment, bus 1202 may be configured to have dozens or even hundreds of nodes, each with its own unique identifier (e.g., a CAN ID). In at least one embodiment, bus 1202 may be read to find steering wheel angle, ground speed, engine revolutions per minute (“RPMs”), button positions, and/or other vehicle status indicators. In at least one embodiment, bus 1202 may be a CAN bus that is ASIL B compliant.

[0214] In at least one embodiment, in addition to, or alternatively from CAN, FlexRay and/or Ethernet protocols may be used. In at least one embodiment, there may be any number of busses forming bus 1202, which may include, without limitation, zero or more CAN busses, zero or more FlexRay busses, zero or more Ethernet busses, and/or zero or more other types of busses using different protocols. In at least one embodiment, two or more busses may be used to perform different functions, and/or may be used for redundancy. For example, a first bus may be

used for collision avoidance functionality and a second bus may be used for actuation control. In at least one embodiment, each bus of bus 1202 may communicate with any of components of vehicle 1200, and two or more busses of bus 1202 may communicate with corresponding components. In at least one embodiment, each of any number of system(s) on chip(s) (“SoC(s)”) 1204 (such as SoC 1204(A) and SoC 1204(B)), each of controller(s) 1236, and/or each computer within vehicle may have access to same input data (e.g., inputs from sensors of vehicle 1200), and may be connected to a common bus, such CAN bus.

[0215] In at least one embodiment, vehicle 1200 may include one or more controller(s) 1236, such as those described herein with respect to FIG. 12A. In at least one embodiment, controller(s) 1236 may be used for a variety of functions. In at least one embodiment, controller(s) 1236 may be coupled to any of various other components and systems of vehicle 1200, and may be used for control of vehicle 1200, artificial intelligence of vehicle 1200, infotainment for vehicle 1200, and/or other functions.

[0216] In at least one embodiment, vehicle 1200 may include any number of SoCs 1204. In at least one embodiment, each of SoCs 1204 may include, without limitation, central processing units (“CPU(s)”) 1206, graphics processing units (“GPU(s)”) 1208, processor(s) 1210, cache(s) 1212, accelerator(s) 1214, data store(s) 1216, and/or other components and features not illustrated. In at least one embodiment, SoC(s) 1204 may be used to control vehicle 1200 in a variety of platforms and systems. For example, in at least one embodiment, SoC(s) 1204 may be combined in a system (e.g., system of vehicle 1200) with a High Definition (“HD”) map 1222 which may obtain map refreshes and/or updates via network interface 1224 from one or more servers (not shown in FIG. 12C).

[0217] In at least one embodiment, CPU(s) 1206 may include a CPU cluster or CPU complex (alternatively referred to herein as a “CCPLEX”). In at least one embodiment, CPU(s) 1206 may include multiple cores and/or level two (“L2”) caches. For instance, in at least one embodiment, CPU(s) 1206 may include eight cores in a coherent multi-processor configuration. In at least one embodiment, CPU(s) 1206 may include four dual-core clusters where each cluster has a dedicated L2 cache (e.g., a 2 megabyte (MB) L2 cache). In at least one embodiment, CPU(s) 1206 (e.g., CPLEX) may be configured to support simultaneous cluster operations enabling any combination of clusters of CPU(s) 1206 to be active at any given time.

[0218] In at least one embodiment, one or more of CPU(s) 1206 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) 1206 may further implement an enhanced algorithm for managing power states, where allowed power states and expected wakeup times are specified, and hardware/microcode determines which best power state to enter for core, cluster, and CCPLEX. In at least one embodiment, processing cores may support simplified power state entry sequences in software with work offloaded to microcode.

[0219] In at least one embodiment, GPU(s) 1208 may include an integrated GPU (alternatively referred to herein as an “iGPU”). In at least one embodiment, GPU(s) 1208 may be programmable and may be efficient for parallel workloads. In at least one embodiment, GPU(s) 1208 may use an enhanced tensor instruction set. In at least one embodiment, GPU(s) 1208 may include one or more streaming microprocessors, where each streaming microprocessor may include a level one (“L1”) cache (e.g., an L1 cache with at 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) 1208 may include at least eight streaming microprocessors. In at least one embodiment, GPU(s) 1208 may use compute application programming interface(s) (API(s)). In at least one embodiment, GPU(s) 1208 may use one or more parallel computing platforms and/or programming models (e.g., NVIDIA’s CUDA model).

[0220] In at least one embodiment, one or more of GPU(s) 1208 may be power-optimized for best performance in automotive and embedded use cases. For example, in at least one embodiment, GPU(s) 1208 could be fabricated on Fin field-effect transistor (“FinFET”) 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 FP64 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 scheduler (e.g., warp scheduler) or sequencer, 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.

[0221] In at least one embodiment, one or more of GPU(s) 1208 may include a high bandwidth memory (“HBM”) and/or a 16 GB HBM2 memory subsystem to provide, in some examples, about 900 GB/second peak memory bandwidth. In at least one embodiment, in addition to, or alternatively from, HBM memory, a synchronous graphics random-access memory (“SGRAM”) may be used, such as a graphics double data rate type five synchronous random-access memory (“GDDR5”).

[0222] In at least one embodiment, GPU(s) 1208 may include unified memory technology. In at least one embodiment, address translation services (“ATS”) support may be used to allow GPU(s) 1208 to access CPU(s) 1206 page tables directly. In at least one embodiment, embodiment, when a GPU of GPU(s) 1208 memory management unit (“MMU”) experiences a miss, an address translation request may be transmitted to CPU(s) 1206. In response, 2 CPU of CPU(s) 1206 may look in its page tables for a virtual-to-physical mapping for an address and transmit translation back to GPU(s) 1208, in at least one embodiment. In at least one embodiment, unified memory technology may allow a single unified virtual address space for memory of both CPU(s) 1206 and GPU(s) 1208, thereby simplifying GPU(s) 1208 programming and porting of applications to GPU(s) 1208.

[0223] In at least one embodiment, GPU(s) 1208 may include any number of access counters that may keep track of frequency of access of GPU(s) 1208 to memory of other 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.

[0224] In at least one embodiment, one or more of SoC(s) 1204 may include any number of cache(s) 1212, including those described herein. For example, in at least one embodiment, cache(s) 1212 could include a level three (“L3”) cache that is available to both CPU(s) 1206 and GPU(s) 1208 (e.g., that is connected to CPU(s) 1206 and GPU(s) 1208). In at least one embodiment, cache(s) 1212 may include a write-back cache that may keep track of states of lines, such as by using a cache coherence protocol (e.g., MEI, MESI, MSI, etc.). In at least one embodiment, a L3 cache may include 4 MB of memory or more, depending on embodiment, although smaller cache sizes may be used.

[0225] In at least one embodiment, one or more of SoC(s) 1204 may include one or more accelerator(s) 1214 (e.g., hardware accelerators, software accelerators, or a combination thereof). In at least one embodiment, SoC(s) 1204 may include a hardware acceleration cluster that may include optimized hardware accelerators and/or large on-chip memory. In at least one embodiment, large on-chip memory (e.g., 4 MB of SRAM), may enable a hardware acceleration cluster to accelerate neural networks and other calculations. In at least one embodiment, a hardware acceleration cluster may be used to complement GPU(s) 1208 and to off-load some of tasks of GPU(s) 1208 (e.g., to free up more cycles of GPU(s) 1208 for performing other tasks). In at least one embodiment, accelerator(s) 1214 could be used for 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.

[0226] In at least one embodiment, accelerator(s) 1214 (e.g., hardware acceleration cluster) may include one or more deep learning accelerator (“DLA”). In at least one embodiment, DLA(s) may include, without limitation, one or more Tensor processing units (“TPUs”) that may be configured to provide an additional ten trillion operations per second for deep learning applications and inferencing. In at least one embodiment, TPUs may be accelerators configured to, and optimized for, performing image processing functions (e.g., for CNNs, RCNNs, etc.). In at least one embodiment, DLA(s) may further be optimized for a specific set of neural network types and floating point operations, as well as inferencing. In at least one embodiment, design of DLA(s) may provide more performance per millimeter than a typical general-purpose GPU, and

typically vastly exceeds performance of a CPU. In at least one embodiment, TPU(s) may perform several functions, including a single-instance convolution function, supporting, for example, INT8, INT16, and FP16 data types for both features and weights, as well as post-processor functions. In at least one embodiment, DLA(s) may quickly and efficiently execute neural networks, especially CNNs, on processed or unprocessed data for any of a variety of functions, including, for example and without limitation: a CNN for object identification and detection using data from camera sensors; a CNN for distance estimation using data from camera sensors; a CNN for emergency vehicle detection and identification and detection using data from microphones; a CNN for facial recognition and vehicle owner identification using data from camera sensors; and/or a CNN for security and/or safety related events.

[0227] In at least one embodiment, DLA(s) may perform any function of GPU(s) 1208, and by using an inference accelerator, for example, a designer may target either DLA(s) or GPU(s) 1208 for any function. For example, in at least one embodiment, a designer may focus processing of CNNs and floating point operations on DLA(s) and leave other functions to GPU(s) 1208 and/or accelerator(s) 1214.

[0228] In at least one embodiment, accelerator(s) 1214 may include programmable vision accelerator (“PVA”), which may alternatively be referred to herein as a computer vision accelerator. In at least one embodiment, PVA may be designed and configured to accelerate computer vision algorithms for advanced driver assistance system (“ADAS”) 1238, autonomous driving, augmented reality (“AR”) applications, and/or virtual reality (“VR”) 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.

[0229] 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.

[0230] In at least one embodiment, DMA may enable components of PVA to access system memory independently of CPU(s) 1206. In at least one embodiment, DMA may support any number of features used to provide optimization to a PVA including, but not limited to, supporting multi-dimensional addressing and/or circular addressing. In at least one embodiment, DMA may support up to six or more dimensions of addressing, which may include, without limitation, block width, block height, block depth, horizontal block stepping, vertical block stepping, and/or depth stepping.

[0231] 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.

[0232] 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.

[0233] In at least one embodiment, accelerator(s) 1214 may include a computer vision network on-chip and static random-access memory (“SRAM”), for providing a high-bandwidth, low latency SRAM for accelerator(s) 1214. In at least one embodiment, on-chip memory may include at least 4 MB SRAM, comprising, for example and without limitation, eight field-configurable memory blocks, that may be accessible by both a PVA and a DLA. In at least one embodiment, each pair of memory blocks may include an advanced peripheral bus (“APB”) interface, configuration circuitry, a controller, and a multiplexer. In at least one embodiment, any type of memory may be used. In at least one embodiment, a PVA and a DLA may access memory via a backbone that provides a PVA and a DLA with high-speed access to memory. In at least one embodiment, a backbone may include a computer vision network on-chip that interconnects a PVA and a DLA to memory (e.g., using APB).

[0234] 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.

[0235] In at least one embodiment, one or more of SoC(s) 1204 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.

[0236] In at least one embodiment, accelerator(s) 1214 can have a wide array of uses for autonomous driving. In at least one embodiment, a PVA may be used for key processing stages in ADAS and autonomous vehicles. In at least one embodiment, a PVA's capabilities are a good match for algorithmic domains needing predictable processing, at low power and low latency. In other words, a PVA performs well on semi-dense or dense regular computation, even on small data sets, which might require predictable run-times with low latency and low power. In at least one embodiment, such as in vehicle 1200, PVAs might be designed to run classic computer vision algorithms, as they can be efficient at object detection and operating on integer math.

[0237] 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.

[0238] 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.

[0239] 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) 1266 that correlates with vehicle 1200 orientation, distance, 3D location estimates of object obtained from neural network and/or other sensors (e.g., LIDAR sensor(s) 1264 or RADAR sensor(s) 1260), among others.

[0240] In at least one embodiment, one or more of SoC(s) 1204 may include data store(s) 1216 (e.g., memory). In at least one embodiment, data store(s) 1216 may be on-chip memory of SoC(s) 1204, which may store neural networks to be executed on GPU(s) 1208 and/or a DLA. In at least one embodiment, data store(s) 1216 may be large enough in capacity to store multiple instances of neural networks for redundancy and safety. In at least one embodiment, data store(s) 1216 may comprise L2 or L3 cache(s).

[0241] In at least one embodiment, one or more of SoC(s) 1204 may include any number of processor(s) 1210 (e.g., embedded processors). In at least one embodiment, processor(s) 1210 may include a boot and power management processor that may be a dedicated processor and subsystem to handle boot power and management functions and related security enforcement. In at least one embodiment, a boot and power management processor may be a part of a boot sequence of SoC(s) 1204 and may provide runtime power management services. In at least one embodiment, a boot power and management processor may provide clock and voltage programming, assistance in system low power state transitions, management of SoC(s) 1204 thermals and temperature sensors, and/or management of SoC(s) 1204 power states. In at least one embodiment, each temperature sensor may be implemented as a ring-oscillator whose output frequency is proportional to temperature, and SoC(s) 1204 may use ring-oscillators to detect temperatures of CPU(s) 1206, GPU(s) 1208, and/or accelerator(s) 1214. In at least one embodiment, if temperatures are determined to exceed a threshold, then a boot and power management processor may enter a temperature fault routine and put SoC(s) 1204 into a lower power state and/or put vehicle 1200 into a chauffeur to safe stop mode (e.g., bring vehicle 1200 to a safe stop).

[0242] In at least one embodiment, processor(s) 1210 may further include a set of embedded processors that may serve as an audio processing engine which may be an audio subsystem that enables full hardware support for multi-channel audio over multiple interfaces, and a broad and flexible range of audio I/O interfaces. In at least one embodiment, an audio processing engine is a dedicated processor core with a digital signal processor with dedicated RAM.

[0243] In at least one embodiment, processor(s) 1210 may further include an always-on processor engine that may provide necessary hardware features to support low power sensor management and wake use cases. In at least one embodiment, an always-on processor engine may include, without limitation, a processor core, a tightly coupled RAM, supporting peripherals (e.g., timers and interrupt controllers), various I/O controller peripherals, and routing logic.

[0244] In at least one embodiment, processor(s) 1210 may further include a safety cluster engine that includes, without limitation, a dedicated processor subsystem to handle safety management for automotive applications. In at least one embodiment, a safety cluster engine may include, without limitation, two or more processor cores, a tightly coupled RAM, support peripherals (e.g., timers, an interrupt controller, etc.), and/or routing logic. In a safety mode, two or more cores may operate, in at least one embodiment, in a lockstep mode and function as a single core with comparison logic to detect any differences between their operations. In at least one embodiment, processor(s) 1210 may further include a real-time camera engine that may include, without limitation, a dedicated processor subsystem for handling real-time camera management. In at least one embodiment, processor(s) 1210 may further include a high-dynamic range signal processor that may include, without limitation, an image signal processor that is a hardware engine that is part of a camera processing pipeline.

[0245] In at least one embodiment, processor(s) 1210 may include a video image compositor that may be a processing block (e.g., implemented on a microprocessor) that implements video post-processing functions needed by a video playback application to produce a final image for a player window. In at least one embodiment, a video image compositor may perform lens distortion correction on wide-view camera(s) 1270, surround camera(s) 1274, and/or on in-cabin monitoring camera sensor(s). In at least one embodiment, in-cabin monitoring camera sensor(s) are preferably monitored by a neural network running on another instance of SoC 1204, configured to identify in cabin events and respond accordingly. In at least one embodiment, an

in-cabin system may perform, without limitation, lip reading to activate cellular service and place a phone call, dictate emails, change a vehicle's destination, activate or change a vehicle's infotainment system and settings, or provide voice-activated web surfing. In at least one embodiment, certain functions are available to a driver when a vehicle is operating in an autonomous mode and are disabled otherwise.

[0246] 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.

[0247] In at least one embodiment, a video image compositor may also be configured to perform stereo rectification on input stereo lens frames. In at least one embodiment, a video image compositor may further be used for user interface composition when an operating system desktop is in use, and GPU(s) 1208 are not required to continuously render new surfaces. In at least one embodiment, when GPU(s) 1208 are powered on and active doing 3D rendering, a video image compositor may be used to offload GPU(s) 1208 to improve performance and responsiveness.

[0248] In at least one embodiment, one or more SoC of SoC(s) 1204 may further include a mobile industry processor interface ("MIPI") camera serial interface for receiving video and input from cameras, a high-speed interface, and/or a video input block that may be used for a camera and related pixel input functions. In at least one embodiment, one or more of SoC(s) 1204 may further include an input/output controller(s) that may be controlled by software and may be used for receiving I/O signals that are uncommitted to a specific role.

[0249] In at least one embodiment, one or more SoC of SoC(s) 1204 may further include a broad range of peripheral interfaces to enable communication with peripherals, audio encoders/decoders ("codecs"), power management, and/or other devices. In at least one embodiment, SoC(s) 1204 may be used to process data from cameras (e.g., connected over Gigabit Multimedia Serial Link and Ethernet channels), sensors (e.g., LIDAR sensor(s) 1264,

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

[0250] In at least one embodiment, SoC(s) 1204 may be an end-to-end platform with a flexible architecture that spans automation Levels 3-5, thereby providing a comprehensive functional safety architecture that leverages and makes efficient use of computer vision and ADAS techniques for diversity and redundancy, and provides a platform for a flexible, reliable driving software stack, along with deep learning tools. In at least one embodiment, SoC(s) 1204 may be faster, more reliable, and even more energy-efficient and space-efficient than conventional systems. For example, in at least one embodiment, accelerator(s) 1214, when combined with CPU(s) 1206, GPU(s) 1208, and data store(s) 1216, may provide for a fast, efficient platform for Level 3-5 autonomous vehicles.

[0251] 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.

[0252] Embodiments described herein allow for multiple neural networks to be performed simultaneously and/or sequentially, and for results to be combined together to enable Level 3-5 autonomous driving functionality. For example, in at least one embodiment, a CNN executing on a DLA or a discrete GPU (e.g., GPU(s) 1220) may include text and word recognition, allowing reading and understanding of traffic signs, including signs for which a neural network has not been specifically trained. In at least one embodiment, a DLA may 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.

[0253] In at least one embodiment, multiple neural networks may be run simultaneously, as for Level 3, 4, or 5 driving. For example, in at least one embodiment, a warning sign stating “Caution: flashing lights indicate icy conditions,” along with an electric light, may be independently or collectively interpreted by several neural networks. In at least one embodiment, such warning sign itself may be identified as a traffic sign by a first deployed neural network (e.g., a neural network that has been trained), text “flashing lights indicate icy conditions” may be interpreted by a second deployed neural network, which informs a vehicle’s path planning software (preferably executing on a CPU Complex) that when flashing lights are detected, icy conditions exist. In at least one embodiment, a flashing light may be identified by operating a third deployed neural network over multiple frames, informing a vehicle’s path-planning software of a presence (or an absence) of flashing lights. In at least one embodiment, all three neural networks may run simultaneously, such as within a DLA and/or on GPU(s) 1208.

[0254] In at least one embodiment, a CNN for facial recognition and vehicle owner identification may use data from camera sensors to identify presence of an authorized driver and/or owner of vehicle 1200. In at least one embodiment, an always-on sensor processing engine may be used to unlock a vehicle when an owner approaches a driver door and turns on lights, and, in a security mode, to disable such vehicle when an owner leaves such vehicle. In this way, SoC(s) 1204 provide for security against theft and/or carjacking.

[0255] In at least one embodiment, a CNN for emergency vehicle detection and identification may use data from microphones 1296 to detect and identify emergency vehicle sirens. In at least one embodiment, SoC(s) 1204 use a CNN for classifying environmental and urban sounds, as well as classifying visual data. In at least one embodiment, a CNN running on a DLA is trained to identify a relative closing speed of an emergency vehicle (e.g., by using a Doppler effect). In at least one embodiment, a CNN may also be trained to identify emergency vehicles specific to a local area in which a vehicle is operating, as identified by GNSS sensor(s) 1258. In at least one embodiment, when operating in Europe, a CNN will seek to detect European sirens, and when in North America, a CNN will seek to identify only North American sirens. In at least one embodiment, once an emergency vehicle 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) 1262, until emergency vehicles pass.

[0256] In at least one embodiment, vehicle 1200 may include CPU(s) 1218 (e.g., discrete CPU(s), or dCPU(s)), that may be coupled to SoC(s) 1204 via a high-speed interconnect (e.g., PCIe). In at least one embodiment, CPU(s) 1218 may include an X86 processor, for example. CPU(s) 1218 may be used to perform any of a variety of functions, including arbitrating potentially inconsistent results between ADAS sensors and SoC(s) 1204, and/or monitoring status and health of controller(s) 1236 and/or an infotainment system on a chip (“infotainment SoC”) 1230, for example. In at least one embodiment, SoC(s) 1204 includes one or more interconnects, and an interconnect can include a peripheral component interconnect express (PCIe).

[0257] In at least one embodiment, vehicle 1200 may include GPU(s) 1220 (e.g., discrete GPU(s), or dGPU(s)), that may be coupled to SoC(s) 1204 via a high-speed interconnect (e.g., NVIDIA’s NVLINK channel). In at least one embodiment, GPU(s) 1220 may provide additional artificial intelligence functionality, such as by executing redundant and/or different neural networks, and may be used to train and/or update neural networks based at least in part on input (e.g., sensor data) from sensors of a vehicle 1200.

[0258] In at least one embodiment, vehicle 1200 may further include network interface 1224 which may include, without limitation, wireless antenna(s) 1226 (e.g., one or more wireless antennas for different communication protocols, such as a cellular antenna, a Bluetooth antenna, etc.). In at least one embodiment, network interface 1224 may be used to enable wireless connectivity to Internet cloud services (e.g., with server(s) and/or other network devices), with other vehicles, and/or with computing devices (e.g., client devices of passengers). In at least one embodiment, to communicate with other vehicles, a direct link may be established between vehicle 1200 and another vehicle and/or an indirect link may be established (e.g., across networks and over the Internet). In at least one embodiment, direct links may be provided using a vehicle-to-vehicle communication link. In at least one embodiment, a vehicle-to-vehicle communication link may provide vehicle 1200 information about vehicles in proximity to vehicle 1200 (e.g., vehicles in front of, on a side of, and/or behind vehicle 1200). In at least one

embodiment, such aforementioned functionality may be part of a cooperative adaptive cruise control functionality of vehicle 1200.

[0259] In at least one embodiment, network interface 1224 may include an SoC that provides modulation and demodulation functionality and enables controller(s) 1236 to communicate over wireless networks. In at least one embodiment, network interface 1224 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.

[0260] In at least one embodiment, vehicle 1200 may further include data store(s) 1228 which may include, without limitation, off-chip (e.g., off SoC(s) 1204) storage. In at least one embodiment, data store(s) 1228 may include, without limitation, one or more storage elements including RAM, SRAM, dynamic random-access memory (“DRAM”), video random-access memory (“VRAM”), flash memory, hard disks, and/or other components and/or devices that may store at least one bit of data.

[0261] In at least one embodiment, vehicle 1200 may further include GNSS sensor(s) 1258 (e.g., GPS and/or assisted GPS sensors), to assist in mapping, perception, occupancy grid generation, and/or path planning functions. In at least one embodiment, any number of GNSS sensor(s) 1258 may be used, including, for example and without limitation, a GPS using a USB connector with an Ethernet-to-Serial (e.g., RS-232) bridge.

[0262] In at least one embodiment, vehicle 1200 may further include RADAR sensor(s) 1260. In at least one embodiment, RADAR sensor(s) 1260 may be used by vehicle 1200 for long-range vehicle detection, even in darkness and/or severe weather conditions. In at least one embodiment, RADAR functional safety levels may be ASIL B. In at least one embodiment, RADAR sensor(s) 1260 may use a CAN bus and/or bus 1202 (e.g., to transmit data generated by RADAR sensor(s) 1260) for control and to access object tracking data, with access to Ethernet channels to access

raw data in some examples. In at least one embodiment, a wide variety of RADAR sensor types may be used. For example, and without limitation, RADAR sensor(s) 1260 may be suitable for front, rear, and side RADAR use. In at least one embodiment, one or more sensor of RADAR sensors(s) 1260 is a Pulse Doppler RADAR sensor.

[0263] In at least one embodiment, RADAR sensor(s) 1260 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) 1260 may help in distinguishing between static and moving objects, and may be used by ADAS system 1238 for emergency brake assist and forward collision warning. In at least one embodiment, sensors 1260(s) included in a long-range RADAR system may include, without limitation, monostatic multimodal RADAR with multiple (e.g., six or more) fixed RADAR antennae and a high-speed CAN and FlexRay interface. In at least one embodiment, with six antennae, a central four antennae may create a focused beam pattern, designed to record vehicle's 1200 surroundings at higher speeds with minimal interference from traffic in adjacent lanes. In at least one embodiment, another two antennae may expand field of view, making it possible to quickly detect vehicles entering or leaving a lane of vehicle 1200.

[0264] In at least one embodiment, mid-range RADAR systems may include, as an example, a range of up to 160 m (front) or 80 m (rear), and a field of view of up to 42 degrees (front) or 150 degrees (rear). In at least one embodiment, short-range RADAR systems may include, without limitation, any number of RADAR sensor(s) 1260 designed to be installed at both ends of a rear bumper. When installed at both ends of a rear bumper, in at least one embodiment, a RADAR sensor system may create two beams that constantly monitor blind spots in a rear direction and next to a vehicle. In at least one embodiment, short-range RADAR systems may be used in ADAS system 1238 for blind spot detection and/or lane change assist.

[0265] In at least one embodiment, vehicle 1200 may further include ultrasonic sensor(s) 1262. In at least one embodiment, ultrasonic sensor(s) 1262, which may be positioned at a front, a back, and/or side location of vehicle 1200, may be used for parking assist and/or to create and update

an occupancy grid. In at least one embodiment, a wide variety of ultrasonic sensor(s) 1262 may be used, and different ultrasonic sensor(s) 1262 may be used for different ranges of detection (e.g., 2.5 m, 4 m). In at least one embodiment, ultrasonic sensor(s) 1262 may operate at functional safety levels of ASIL B.

[0266] In at least one embodiment, vehicle 1200 may include LIDAR sensor(s) 1264. In at least one embodiment, LIDAR sensor(s) 1264 may be used for object and pedestrian detection, emergency braking, collision avoidance, and/or other functions. In at least one embodiment, LIDAR sensor(s) 1264 may operate at functional safety level ASIL B. In at least one embodiment, vehicle 1200 may include multiple LIDAR sensors 1264 (e.g., two, four, six, etc.) that may use an Ethernet channel (e.g., to provide data to a Gigabit Ethernet switch).

[0267] In at least one embodiment, LIDAR sensor(s) 1264 may be capable of providing a list of objects and their distances for a 360-degree field of view. In at least one embodiment, commercially available LIDAR sensor(s) 1264 may have an advertised range of approximately 100 m, with an accuracy of 2 cm to 3 cm, and with support for a 100 Mbps Ethernet connection, for example. In at least one embodiment, one or more non-protruding LIDAR sensors may be used. In such an embodiment, LIDAR sensor(s) 1264 may include a small device that may be embedded into a front, a rear, a side, and/or a corner location of vehicle 1200. In at least one embodiment, LIDAR sensor(s) 1264, in such an embodiment, may provide up to a 120-degree horizontal and 35-degree vertical field-of-view, with a 200 m range even for low-reflectivity objects. In at least one embodiment, front-mounted LIDAR sensor(s) 1264 may be configured for a horizontal field of view between 45 degrees and 135 degrees.

[0268] In at least one embodiment, LIDAR technologies, such as 3D flash LIDAR, may also be used. In at least one embodiment, 3D flash LIDAR uses a flash of a laser as a transmission source, to illuminate surroundings of vehicle 1200 up to approximately 200 m. In at least one embodiment, a flash LIDAR unit includes, without limitation, a receptor, which records laser pulse transit time and reflected light on each pixel, which in turn corresponds to a range from vehicle 1200 to objects. In at least one embodiment, flash LIDAR may allow for highly accurate and distortion-free images of surroundings to be generated with every laser flash. In at least one embodiment, four flash LIDAR sensors may be deployed, one at each side of vehicle 1200. In at least one embodiment, 3D flash LIDAR systems include, without limitation, a solid-state 3D

staring array LIDAR camera with no moving parts other than a fan (e.g., a non-scanning LIDAR device). In at least one embodiment, flash LIDAR device may use a 5 nanosecond class I (eye-safe) laser pulse per frame and may capture reflected laser light as a 3D range point cloud and co-registered intensity data.

[0269] In at least one embodiment, vehicle 1200 may further include IMU sensor(s) 1266. In at least one embodiment, IMU sensor(s) 1266 may be located at a center of a rear axle of vehicle 1200. In at least one embodiment, IMU sensor(s) 1266 may include, for example and without limitation, accelerometer(s), magnetometer(s), gyroscope(s), a magnetic compass, magnetic compasses, and/or other sensor types. In at least one embodiment, such as in six-axis applications, IMU sensor(s) 1266 may include, without limitation, accelerometers and gyroscopes. In at least one embodiment, such as in nine-axis applications, IMU sensor(s) 1266 may include, without limitation, accelerometers, gyroscopes, and magnetometers.

[0270] In at least one embodiment, IMU sensor(s) 1266 may be implemented as a miniature, high performance GPS-Aided Inertial Navigation System (“GPS/INS”) that combines micro-electro-mechanical systems (“MEMS”) inertial sensors, a high-sensitivity GPS receiver, and advanced Kalman filtering algorithms to provide estimates of position, velocity, and attitude. In at least one embodiment, IMU sensor(s) 1266 may enable vehicle 1200 to estimate its heading without requiring input from a magnetic sensor by directly observing and correlating changes in velocity from a GPS to IMU sensor(s) 1266. In at least one embodiment, IMU sensor(s) 1266 and GNSS sensor(s) 1258 may be combined in a single integrated unit.

[0271] In at least one embodiment, vehicle 1200 may include microphone(s) 1296 placed in and/or around vehicle 1200. In at least one embodiment, microphone(s) 1296 may be used for emergency vehicle detection and identification, among other things.

[0272] In at least one embodiment, vehicle 1200 may further include any number of camera types, including stereo camera(s) 1268, wide-view camera(s) 1270, infrared camera(s) 1272, surround camera(s) 1274, long-range camera(s) 1298, mid-range camera(s) 1276, and/or other camera types. In at least one embodiment, cameras may be used to capture image data around an entire periphery of vehicle 1200. In at least one embodiment, which types of cameras used depends on vehicle 1200. In at least one embodiment, any combination of camera types may be used to provide necessary coverage around vehicle 1200. In at least one embodiment, a number

of cameras deployed may differ depending on embodiment. For example, in at least one embodiment, vehicle 1200 could include six cameras, seven cameras, ten cameras, twelve cameras, or another number of cameras. In at least one embodiment, cameras may support, as an example and without limitation, Gigabit Multimedia Serial Link (“GMSL”) and/or Gigabit Ethernet communications. In at least one embodiment, each camera might be as described with more detail previously herein with respect to FIG. 12A and FIG. 12B.

[0273] In at least one embodiment, vehicle 1200 may further include vibration sensor(s) 1242. In at least one embodiment, vibration sensor(s) 1242 may measure vibrations of components of vehicle 1200, such as axle(s). For example, in at least one embodiment, changes in vibrations may indicate a change in road surfaces. In at least one embodiment, when two or more vibration sensors 1242 are used, differences between vibrations may be used to determine friction or slippage of road surface (e.g., when a difference in vibration is between a power-driven axle and a freely rotating axle).

[0274] In at least one embodiment, vehicle 1200 may include ADAS system 1238. In at least one embodiment, ADAS system 1238 may include, without limitation, an SoC, in some examples. In at least one embodiment, ADAS system 1238 may include, without limitation, any number and combination of an autonomous/adaptive/automatic cruise control (“ACC”) system, a cooperative adaptive cruise control (“CACC”) system, a forward crash warning (“FCW”) system, an automatic emergency braking (“AEB”) system, a lane departure warning (“LDW”) system, a lane keep assist (“LKA”) system, a blind spot warning (“BSW”) system, a rear cross-traffic warning (“RCTW”) system, a collision warning (“CW”) system, a lane centering (“LC”) system, and/or other systems, features, and/or functionality.

[0275] In at least one embodiment, ACC system may use RADAR sensor(s) 1260, LIDAR sensor(s) 1264, and/or any number of camera(s). In at least one embodiment, ACC system may include a longitudinal ACC system and/or a lateral ACC system. In at least one embodiment, a longitudinal ACC system monitors and controls distance to another vehicle immediately ahead of vehicle 1200 and automatically adjusts speed of vehicle 1200 to maintain a safe distance from vehicles ahead. In at least one embodiment, a lateral ACC system performs distance keeping, and advises vehicle 1200 to change lanes when necessary. In at least one embodiment, a lateral ACC is related to other ADAS applications, such as LC and CW.

[0276] In at least one embodiment, a CACC system uses information from other vehicles that may be received via network interface 1224 and/or wireless antenna(s) 1226 from other vehicles via a wireless link, or indirectly, over a network connection (e.g., over the Internet). In at least one embodiment, direct links may be provided by a vehicle-to-vehicle (“V2V”) communication link, while indirect links may be provided by an infrastructure-to-vehicle (“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 1200), while I2V communication provides information about traffic further ahead. In at least one embodiment, a CACC system may include either or both I2V and V2V information sources. In at least one embodiment, given information of vehicles ahead of vehicle 1200, a CACC system may be more reliable and it has potential to improve traffic flow smoothness and reduce congestion on road.

[0277] 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) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, an FCW system may provide a warning, such as in form of a sound, visual warning, vibration and/or a quick brake pulse.

[0278] In at least one embodiment, an AEB system detects an impending forward collision with another vehicle or other object, and may automatically apply brakes if a driver does not take corrective action within a specified time or distance parameter. In at least one embodiment, AEB system may use front-facing camera(s) and/or RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC. In at least one embodiment, 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.

[0279] In at least one embodiment, an LDW system provides visual, audible, and/or tactile warnings, such as steering wheel or seat vibrations, to alert driver when vehicle 1200 crosses lane markings. In at least one embodiment, an LDW system does not activate when a driver

indicates an intentional lane departure, such as by activating a turn signal. In at least one embodiment, an LDW system may use front-side facing cameras, coupled to a dedicated 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 1200 if vehicle 1200 starts to exit its lane.

[0280] In at least one embodiment, a BSW system detects and warns a driver of vehicles in an automobile's blind spot. In at least one embodiment, a BSW system may provide a visual, audible, and/or tactile alert to indicate that merging or changing lanes is unsafe. In at least one embodiment, a BSW system may provide an additional warning when a driver uses a turn signal. In at least one embodiment, a BSW system may use rear-side facing camera(s) and/or RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component.

[0281] In at least one embodiment, an RCTW system may provide visual, audible, and/or tactile notification when an object is detected outside a rear-camera range when vehicle 1200 is backing up. In at least one embodiment, an RCTW system includes an AEB system to ensure that vehicle brakes are applied to avoid a crash. In at least one embodiment, an RCTW system may use one or more rear-facing RADAR sensor(s) 1260, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component.

[0282] In at least one embodiment, conventional ADAS systems may be prone to false positive results which may be annoying and distracting to a driver, but typically are not catastrophic, because conventional ADAS systems alert a driver and allow that driver to decide whether a safety condition truly exists and act accordingly. In at least one embodiment, vehicle 1200 itself decides, in case of conflicting results, whether to heed result from a primary computer or a secondary computer (e.g., a first controller or a second controller of controllers 1236). For example, in at least one embodiment, ADAS system 1238 may be a backup and/or secondary computer for providing perception information to a backup computer rationality module. In at least one embodiment, a backup computer rationality monitor may run redundant diverse software on hardware components to detect faults in perception and dynamic driving tasks. In at

least one embodiment, outputs from ADAS system 1238 may be provided to a supervisory MCU. In at least one embodiment, if outputs from a primary computer and outputs from a secondary computer conflict, a supervisory MCU determines how to reconcile conflict to ensure safe operation.

[0283] 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.

[0284] 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) 1204.

[0285] In at least one embodiment, ADAS system 1238 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.

[0286] In at least one embodiment, an output of ADAS system 1238 may be fed into a primary computer's perception block and/or a primary computer's dynamic driving task block. For example, in at least one embodiment, if ADAS system 1238 indicates a forward crash warning due to an object immediately ahead, a perception block may use this information when identifying objects. In at least one embodiment, a secondary computer may have its own neural network that is trained and thus reduces a risk of false positives, as described herein.

[0287] In at least one embodiment, vehicle 1200 may further include infotainment SoC 1230 (e.g., an in-vehicle infotainment system (IVI)). Although illustrated and described as an SoC, infotainment system SoC 1230, in at least one embodiment, may not be an SoC, and may include, without limitation, two or more discrete components. In at least one embodiment, infotainment SoC 1230 may include, without limitation, a combination of hardware and software that may be used to provide audio (e.g., music, a personal digital assistant, navigational instructions, news, radio, etc.), video (e.g., TV, movies, streaming, etc.), phone (e.g., hands-free calling), network connectivity (e.g., LTE, WiFi, etc.), and/or information services (e.g., navigation systems, rear-parking assistance, a radio data system, vehicle related information such as fuel level, total distance covered, brake fuel level, oil level, door open/close, air filter information, etc.) to vehicle 1200. For example, infotainment SoC 1230 could include radios, disk players, navigation systems, video players, USB and Bluetooth connectivity, carputers, in-car entertainment, WiFi, steering wheel audio controls, hands free voice control, a heads-up display ("HUD"), HMI display 1234, a telematics device, a control panel (e.g., for controlling and/or interacting with various components, features, and/or systems), and/or other components. In at least one embodiment, infotainment SoC 1230 may further be used to provide information (e.g.,

visual and/or audible) to user(s) of vehicle 1200, such as information from ADAS system 1238, autonomous driving information such as planned vehicle maneuvers, trajectories, surrounding environment information (e.g., intersection information, vehicle information, road information, etc.), and/or other information.

[0288] In at least one embodiment, infotainment SoC 1230 may include any amount and type of GPU functionality. In at least one embodiment, infotainment SoC 1230 may communicate over bus 1202 with other devices, systems, and/or components of vehicle 1200. In at least one embodiment, infotainment SoC 1230 may be coupled to a supervisory MCU such that a GPU of an infotainment system may perform some self-driving functions in event that primary controller(s) 1236 (e.g., primary and/or backup computers of vehicle 1200) fail. In at least one embodiment, infotainment SoC 1230 may put vehicle 1200 into a chauffeur to safe stop mode, as described herein.

[0289] In at least one embodiment, vehicle 1200 may further include instrument cluster 1232 (e.g., a digital dash, an electronic instrument cluster, a digital instrument panel, etc.). In at least one embodiment, instrument cluster 1232 may include, without limitation, a controller and/or supercomputer (e.g., a discrete controller or supercomputer). In at least one embodiment, instrument cluster 1232 may include, without limitation, any number and combination of a set of instrumentation such as a speedometer, fuel level, oil pressure, tachometer, odometer, turn indicators, gearshift position indicator, seat belt warning light(s), parking-brake warning light(s), engine-malfunction light(s), supplemental restraint system (e.g., airbag) information, lighting controls, safety system controls, navigation information, etc. In some examples, information may be displayed and/or shared among infotainment SoC 1230 and instrument cluster 1232. In at least one embodiment, instrument cluster 1232 may be included as part of infotainment SoC 1230, or vice versa.

[0290] In at least one embodiment, at least one component shown or described with respect to FIG. 12C is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, data store(s) 1216 stores one or more efficiency lists used, at least in part, to select which neural network masks to remove from and/or insert into an efficient neural model for deployment on a specific GPU hardware platform as described in conjunction with FIGS. 4 and 6, and as otherwise described herein.

[0291] FIG. 12D is a diagram of a system for communication between cloud-based server(s) and autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, system may include, without limitation, server(s) 1278, network(s) 1290, and any number and type of vehicles, including vehicle 1200. In at least one embodiment, server(s) 1278 may include, without limitation, a plurality of GPUs 1284(A)-1284(H) (collectively referred to herein as GPUs 1284), PCIe switches 1282(A)-1282(D) (collectively referred to herein as PCIe switches 1282), and/or CPUs 1280(A)-1280(B) (collectively referred to herein as CPUs 1280). In at least one embodiment, GPUs 1284, CPUs 1280, and PCIe switches 1282 may be interconnected with high-speed interconnects such as, for example and without limitation, NVLink interfaces 1288 developed by NVIDIA and/or PCIe connections 1286. In at least one embodiment, GPUs 1284 are connected via an NVLink and/or NVSwitch SoC and GPUs 1284 and PCIe switches 1282 are connected via PCIe interconnects. Although eight GPUs 1284, two CPUs 1280, and four PCIe switches 1282 are illustrated, this is not intended to be limiting. In at least one embodiment, each of server(s) 1278 may include, without limitation, any number of GPUs 1284, CPUs 1280, and/or PCIe switches 1282, in any combination. For example, in at least one embodiment, server(s) 1278 could each include eight, sixteen, thirty-two, and/or more GPUs 1284.

[0292] In at least one embodiment, server(s) 1278 may receive, over network(s) 1290 and from vehicles, image data representative of images showing unexpected or changed road conditions, such as recently commenced road-work. In at least one embodiment, server(s) 1278 may transmit, over network(s) 1290 and to vehicles, neural networks 1292, updated or otherwise, and/or map information 1294, including, without limitation, information regarding traffic and road conditions. In at least one embodiment, updates to map information 1294 may include, without limitation, updates for HD map 1222, such as information regarding construction sites, potholes, detours, flooding, and/or other obstructions. In at least one embodiment, neural networks 1292, and/or map information 1294 may have resulted from new training and/or experiences represented in data received from any number of vehicles in an environment, and/or based at least in part on training performed at a data center (e.g., using server(s) 1278 and/or other servers).

[0293] In at least one embodiment, server(s) 1278 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) 1290), and/or machine learning models may be used by server(s) 1278 to remotely monitor vehicles.

[0294] In at least one embodiment, server(s) 1278 may receive data from vehicles and apply data to up-to-date real-time neural networks for real-time intelligent inferencing. In at least one embodiment, server(s) 1278 may include deep-learning supercomputers and/or dedicated AI computers powered by GPU(s) 1284, such as a DGX and DGX Station machines developed by NVIDIA. However, in at least one embodiment, server(s) 1278 may include deep learning infrastructure that uses CPU-powered data centers.

[0295] In at least one embodiment, deep-learning infrastructure of server(s) 1278 may be capable of fast, real-time inferencing, and may use that capability to evaluate and verify health of processors, software, and/or associated hardware in vehicle 1200. For example, in at least one embodiment, deep-learning infrastructure may receive periodic updates from vehicle 1200, such as a sequence of images and/or objects that vehicle 1200 has located in that sequence of images (e.g., via computer vision and/or other machine learning object classification techniques). In at least one embodiment, deep-learning infrastructure may run its own neural network to identify objects and compare them with objects identified by vehicle 1200 and, if results do not match and deep-learning infrastructure concludes that AI in vehicle 1200 is malfunctioning, then server(s) 1278 may transmit a signal to vehicle 1200 instructing a fail-safe computer of vehicle 1200 to assume control, notify passengers, and complete a safe parking maneuver.

[0296] In at least one embodiment, server(s) 1278 may include GPU(s) 1284 and one or more programmable inference accelerators (e.g., NVIDIA's TensorRT 3 devices). In at least one embodiment, a combination of GPU-powered servers and inference acceleration may make real-

time responsiveness possible. In at least one embodiment, such as where performance is less critical, servers powered by CPUs, FPGAs, and other processors may be used for inferencing. In at least one embodiment, hardware structure(s) 915 are used to perform one or more embodiments. Details regarding hardware structure(s) 915 are provided herein in conjunction with FIGS. 9A and/or 9B.

COMPUTER SYSTEMS

[0297] FIG. 13 is a block diagram illustrating an exemplary computer system, which may be a system with interconnected devices and components, a system-on-a-chip (SOC) or some combination thereof formed with a processor that may include execution units to execute an instruction, according to at least one embodiment. In at least one embodiment, a computer system 1300 may include, without limitation, a component, such as a processor 1302 to employ execution units including logic to perform algorithms for process data, in accordance with present disclosure, such as in embodiment described herein. In at least one embodiment, computer system 1300 may include processors, such as PENTIUM® Processor family, Xeon™, Itanium®, XScale™ and/or StrongARM™, Intel® Core™, or Intel® Nervana™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and like) may also be used. In at least one embodiment, computer system 1300 may execute a version of WINDOWS operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems (UNIX and Linux, for example), embedded software, and/or graphical user interfaces, may also be used.

[0298] Embodiments may be used in other devices such as handheld devices and embedded applications. Some examples of handheld devices include cellular phones, Internet Protocol devices, digital cameras, personal digital assistants (“PDAs”), and handheld PCs. In at least one embodiment, embedded applications may include a microcontroller, a digital signal processor (“DSP”), system on a chip, network computers (“NetPCs”), set-top boxes, network hubs, wide area network (“WAN”) switches, or any other system that may perform one or more instructions in accordance with at least one embodiment.

[0299] In at least one embodiment, computer system 1300 may include, without limitation, processor 1302 that may include, without limitation, one or more execution units 1308 to

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

[0300] In at least one embodiment, processor 1302 may include, without limitation, a Level 1 (“L1”) internal cache memory (“cache”) 1304. In at least one embodiment, processor 1302 may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor 1302. Other embodiments may also include a combination of both internal and external caches depending on particular implementation and needs. In at least one embodiment, a register file 1306 may store different types of data in various registers including, without limitation, integer registers, floating point registers, status registers, and an instruction pointer register.

[0301] In at least one embodiment, execution unit 1308, including, without limitation, logic to perform integer and floating point operations, also resides in processor 1302. In at least one embodiment, processor 1302 may also include a microcode (“ucode”) read only memory (“ROM”) that stores microcode for certain macro instructions. In at least one embodiment, execution unit 1308 may include logic to handle a packed instruction set 1309. In at least one embodiment, by including packed instruction set 1309 in an instruction set of a general-purpose processor, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using packed data in processor 1302. In at least one embodiment, many multimedia applications may be accelerated and executed more efficiently by using a full width of a processor’s data bus for performing operations on packed data, which may eliminate a need to transfer smaller units of data across that processor’s data bus to perform one or more operations one data element at a time.

[0302] In at least one embodiment, execution unit 1308 may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. In at least one embodiment, computer system 1300 may include, without limitation, a memory 1320. In at least one embodiment, memory 1320 may be a Dynamic Random Access Memory (“DRAM”) device, a Static Random Access Memory (“SRAM”) device, a flash memory device, or another memory device. In at least one embodiment, memory 1320 may store instruction(s) 1319 and/or data 1321 represented by data signals that may be executed by processor 1302.

[0303] In at least one embodiment, a system logic chip may be coupled to processor bus 1310 and memory 1320. In at least one embodiment, a system logic chip may include, without limitation, a memory controller hub (“MCH”) 1316, and processor 1302 may communicate with MCH 1316 via processor bus 1310. In at least one embodiment, MCH 1316 may provide a high bandwidth memory path 1318 to memory 1320 for instruction and data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH 1316 may direct data signals between processor 1302, memory 1320, and other components in computer system 1300 and to bridge data signals between processor bus 1310, memory 1320, and a system I/O interface 1322. In at least one embodiment, a system logic chip may provide a graphics port for coupling to a graphics controller. In at least one embodiment, MCH 1316 may be coupled to memory 1320 through high bandwidth memory path 1318 and a graphics/video card 1312 may be coupled to MCH 1316 through an Accelerated Graphics Port (“AGP”) interconnect 1314.

[0304] In at least one embodiment, computer system 1300 may use system I/O interface 1322 as a proprietary hub interface bus to couple MCH 1316 to an I/O controller hub (“ICH”) 1330. In at least one embodiment, ICH 1330 may provide direct connections to some I/O devices via a local I/O bus. In at least one embodiment, a local I/O bus may include, without limitation, a high-speed I/O bus for connecting peripherals to memory 1320, a chipset, and processor 1302. Examples may include, without limitation, an audio controller 1329, a firmware hub (“flash BIOS”) 1328, a wireless transceiver 1326, a data storage 1324, a legacy I/O controller 1323 containing user input and keyboard interfaces 1325, a serial expansion port 1327, such as a Universal Serial Bus (“USB”) port, and a network controller 1334. In at least one embodiment, data storage 1324 may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

[0305] In at least one embodiment, FIG. 13 illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. 13 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 13 may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of computer system 1300 are interconnected using compute express link (CXL) interconnects.

[0306] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in computer system 1300 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.

[0307] In at least one embodiment, at least one component shown or described with respect to FIGS. 12D-13 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, processor 1302 performs one or more operations related to inserting neural network masks into a trained, dense target neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0308] FIG. 14 is a block diagram illustrating an electronic device 1400 for utilizing a processor 1410, according to at least one embodiment. In at least one embodiment, electronic device 1400 may be, for example and without limitation, a notebook, a tower server, a rack server, a blade server, a laptop, a desktop, a tablet, a mobile device, a phone, an embedded computer, or any other suitable electronic device.

[0309] In at least one embodiment, electronic device 1400 may include, without limitation, processor 1410 communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor 1410 is coupled using a bus or interface, such as a I2C bus, a System Management Bus (“SMBus”), a Low Pin Count (LPC) bus, a Serial Peripheral Interface (“SPI”), a High Definition Audio (“HDA”) bus, a Serial Advance Technology Attachment (“SATA”) bus, a Universal Serial Bus (“USB”) (versions 1, 2, 3, etc.), or a Universal Asynchronous Receiver/Transmitter (“UART”) bus. In at least one

embodiment, FIG. 14 illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. 14 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 14 may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of FIG. 14 are interconnected using compute express link (CXL) interconnects.

[0310] In at least one embodiment, FIG. 14 may include a display 1424, a touch screen 1425, a touch pad 1430, a Near Field Communications unit (“NFC”) 1445, a sensor hub 1440, a thermal sensor 1446, an Express Chipset (“EC”) 1435, a Trusted Platform Module (“TPM”) 1438, BIOS/firmware/flash memory (“BIOS, FW Flash”) 1422, a DSP 1460, a drive 1420 such as a Solid State Disk (“SSD”) or a Hard Disk Drive (“HDD”), a wireless local area network unit (“WLAN”) 1450, a Bluetooth unit 1452, a Wireless Wide Area Network unit (“WWAN”) 1456, a Global Positioning System (GPS) unit 1455, a camera (“USB 3.0 camera”) 1454 such as a USB 3.0 camera, and/or a Low Power Double Data Rate (“LPDDR”) memory unit (“LPDDR3”) 1415 implemented in, for example, an LPDDR3 standard. These components may each be implemented in any suitable manner.

[0311] In at least one embodiment, other components may be communicatively coupled to processor 1410 through components described herein. In at least one embodiment, an accelerometer 1441, an ambient light sensor (“ALS”) 1442, a compass 1443, and a gyroscope 1444 may be communicatively coupled to sensor hub 1440. In at least one embodiment, a thermal sensor 1439, a fan 1437, a keyboard 1436, and touch pad 1430 may be communicatively coupled to EC 1435. In at least one embodiment, speakers 1463, headphones 1464, and a microphone (“mic”) 1465 may be communicatively coupled to an audio unit (“audio codec and class D amp”) 1462, which may in turn be communicatively coupled to DSP 1460. In at least one embodiment, audio unit 1462 may include, for example and without limitation, an audio coder/decoder (“codec”) and a class D amplifier. In at least one embodiment, a SIM card (“SIM”) 1457 may be communicatively coupled to WWAN unit 1456. In at least one embodiment, components such as WLAN unit 1450 and Bluetooth unit 1452, as well as WWAN unit 1456 may be implemented in a Next Generation Form Factor (“NGFF”).

[0312] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in electronic device 1400 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0313] In at least one embodiment, at least one component shown or described with respect to FIG. 14 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, processor 1415 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0314] FIG. 15 illustrates a computer system 1500, according to at least one embodiment. In at least one embodiment, computer system 1500 is configured to implement various processes and methods described throughout this disclosure.

[0315] In at least one embodiment, computer system 1500 comprises, without limitation, at least one central processing unit (“CPU”) 1502 that is connected to a communication bus 1510 implemented using any suitable protocol, such as PCI (“Peripheral Component Interconnect”), peripheral component interconnect express (“PCI-Express”), AGP (“Accelerated Graphics Port”), HyperTransport, or any other bus or point-to-point communication protocol(s). In at least one embodiment, computer system 1500 includes, without limitation, a main memory 1504 and control logic (e.g., implemented as hardware, software, or a combination thereof) and data are stored in main memory 1504, which may take form of random access memory (“RAM”). In at least one embodiment, a network interface subsystem (“network interface”) 1522 provides an interface to other computing devices and networks for receiving data from and transmitting data to other systems with computer system 1500.

[0316] In at least one embodiment, computer system 1500, in at least one embodiment, includes, without limitation, input devices 1508, a parallel processing system 1512, and display devices 1506 that can be implemented using a conventional cathode ray tube (“CRT”), a liquid crystal display (“LCD”), a light emitting diode (“LED”) display, a plasma display, or other

suitable display technologies. In at least one embodiment, user input is received from input devices 1508 such as keyboard, mouse, touchpad, microphone, etc. In at least one embodiment, each module described herein can be situated on a single semiconductor platform to form a processing system.

[0317] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in computer 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.

[0318] In at least one embodiment, at least one component shown or described with respect to FIG. 15 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, computer system 1500 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0319] FIG. 16 illustrates a computer system 1600, according to at least one embodiment. In at least one embodiment, computer system 1600 includes, without limitation, a computer 1610 and a USB stick 1620. In at least one embodiment, computer 1610 may include, without limitation, any number and type of processor(s) (not shown) and a memory (not shown). In at least one embodiment, computer 1610 includes, without limitation, a server, a cloud instance, a laptop, and a desktop computer.

[0320] In at least one embodiment, USB stick 1620 includes, without limitation, a processing unit 1630, a USB interface 1640, and USB interface logic 1650. In at least one embodiment, processing unit 1630 may be any instruction execution system, apparatus, or device capable of executing instructions. In at least one embodiment, processing unit 1630 may include, without limitation, any number and type of processing cores (not shown). In at least one embodiment, processing unit 1630 comprises an application specific integrated circuit (“ASIC”) that is optimized to perform any amount and type of operations associated with machine learning. For instance, in at least one embodiment, processing unit 1630 is a tensor processing unit (“TPU”)

that is optimized to perform machine learning inference operations. In at least one embodiment, processing unit 1630 is a vision processing unit (“VPU”) that is optimized to perform machine vision and machine learning inference operations.

[0321] In at least one embodiment, USB interface 1640 may be any type of USB connector or USB socket. For instance, in at least one embodiment, USB interface 1640 is a USB 3.0 Type-C socket for data and power. In at least one embodiment, USB interface 1640 is a USB 3.0 Type-A connector. In at least one embodiment, USB interface logic 1650 may include any amount and type of logic that enables processing unit 1630 to interface with devices (e.g., computer 1610) via USB connector 1640.

[0322] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in computer system 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.

[0323] In at least one embodiment, at least one component shown or described with respect to FIG. 16 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, computer system 1600 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0324] FIG. 17A illustrates an exemplary architecture in which a plurality of GPUs 1710(1)-1710(N) is communicatively coupled to a plurality of multi-core processors 1705(1)-1705(M) over high-speed links 1740(1)-1740(N) (e.g., buses, point-to-point interconnects, etc.). In at least one embodiment, high-speed links 1740(1)-1740(N) support a communication throughput of 4 GB/s, 30 GB/s, 80 GB/s or higher. In at least one embodiment, various interconnect protocols may be used including, but not limited to, PCIe 4.0 or 5.0 and NVLink 2.0. In various figures, “N” and “M” represent positive integers, values of which may be different from figure to figure. In at least one embodiment, one or more GPUs in a plurality of GPUs 1710(1)-1710(N) includes one or more graphics cores (also referred to simply as “cores”) 2000 as disclosed in

Figures 20A and 20B. In at least one embodiment, one or more graphics cores 2000 may be referred to as streaming multiprocessors (“SMs”), stream processors (“SPs”), stream processing units (“SPUs”), compute units (“CUs”), execution units (“EUs”), and/or slices, where a slice in this context can refer to a portion of processing resources in a processing unit (e.g., 16 cores, a ray tracing unit, a thread director or scheduler).

[0325] In addition, and in at least one embodiment, two or more of GPUs 1710 are interconnected over high-speed links 1729(1)-1729(2), which may be implemented using similar or different protocols/links than those used for high-speed links 1740(1)-1740(N). Similarly, two or more of multi-core processors 1705 may be connected over a high-speed link 1728 which may be symmetric multi-processor (SMP) buses operating at 20 GB/s, 30 GB/s, 120 GB/s or higher. Alternatively, all communication between various system components shown in FIG. 17A may be accomplished using similar protocols/links (e.g., over a common interconnection fabric).

[0326] In at least one embodiment, each multi-core processor 1705 is communicatively coupled to a processor memory 1701(1)-1701(M), via memory interconnects 1726(1)-1726(M), respectively, and each GPU 1710(1)-1710(N) is communicatively coupled to GPU memory 1720(1)-1720(N) over GPU memory interconnects 1750(1)-1750(N), respectively. In at least one embodiment, memory interconnects 1726 and 1750 may utilize similar or different memory access technologies. By way of example, and not limitation, processor memories 1701(1)-1701(M) and GPU memories 1720 may be volatile memories such as dynamic random access memories (DRAMs) (including stacked DRAMs), Graphics DDR SDRAM (GDDR) (e.g., GDDR5, GDDR6), or High Bandwidth Memory (HBM) and/or may be non-volatile memories such as 3D XPoint or Nano-Ram. In at least one embodiment, some portion of processor memories 1701 may be volatile memory and another portion may be non-volatile memory (e.g., using a two-level memory (2LM) hierarchy).

[0327] As described herein, although various multi-core processors 1705 and GPUs 1710 may be physically coupled to a particular memory 1701, 1720, respectively, and/or a unified memory architecture may be implemented in which a virtual system address space (also referred to as “effective address” space) is distributed among various physical memories. For example, processor memories 1701(1)-1701(M) may each comprise 64 GB of system memory address space and GPU memories 1720(1)-1720(N) may each comprise 32 GB of system memory

address space resulting in a total of 256 GB addressable memory when M=2 and N=4. Other values for N and M are possible.

[0328] FIG. 17B illustrates additional details for an interconnection between a multi-core processor 1707 and a graphics acceleration module 1746 in accordance with one exemplary embodiment. In at least one embodiment, graphics acceleration module 1746 may include one or more GPU chips integrated on a line card which is coupled to processor 1707 via high-speed link 1740 (e.g., a PCIe bus, NVLink, etc.). In at least one embodiment, graphics acceleration module 1746 may alternatively be integrated on a package or chip with processor 1707.

[0329] In at least one embodiment, processor 1707 includes a plurality of cores 1760A-1760D (which may be referred to as “execution units”), each with a translation lookaside buffer (“TLB”) 1761A-1761D and one or more caches 1762A-1762D. In at least one embodiment, cores 1760A-1760D may include various other components for executing instructions and processing data that are not illustrated. In at least one embodiment, caches 1762A-1762D may comprise Level 1 (L1) and Level 2 (L2) caches. In addition, one or more shared caches 1756 may be included in caches 1762A-1762D and shared by sets of cores 1760A-1760D. For example, one embodiment of processor 1707 includes 24 cores, each with its own L1 cache, twelve shared L2 caches, and twelve shared L3 caches. In this embodiment, one or more L2 and L3 caches are shared by two adjacent cores. In at least one embodiment, processor 1707 and graphics acceleration module 1746 connect with system memory 1714, which may include processor memories 1701(1)-1701(M) of FIG. 17A.

[0330] In at least one embodiment, coherency is maintained for data and instructions stored in various caches 1762A-1762D, 1756 and system memory 1714 via inter-core communication over a coherence bus 1764. In at least one embodiment, for example, each cache may have cache coherency logic/circuitry associated therewith to communicate to over coherence bus 1764 in response to detected reads or writes to particular cache lines. In at least one embodiment, a cache snooping protocol is implemented over coherence bus 1764 to snoop cache accesses.

[0331] In at least one embodiment, a proxy circuit 1725 communicatively couples graphics acceleration module 1746 to coherence bus 1764, allowing graphics acceleration module 1746 to participate in a cache coherence protocol as a peer of cores 1760A-1760D. In particular, in at least one embodiment, an interface 1735 provides connectivity to proxy circuit 1725 over high-

speed link 1740 and an interface 1737 connects graphics acceleration module 1746 to high-speed link 1740.

[0332] In at least one embodiment, an accelerator integration circuit 1736 provides cache management, memory access, context management, and interrupt management services on behalf of a plurality of graphics processing engines 1731(1)-1731(N) of graphics acceleration module 1746. In at least one embodiment, graphics processing engines 1731(1)-1731(N) may each comprise a separate graphics processing unit (GPU). In at least one embodiment, plurality of graphics processing engines 1731(1)-1731(N) of graphics acceleration module 1746 include one or more graphics cores 2000 as discussed in connection with Figures 20A and 20B. In at least one embodiment, graphics processing engines 1731(1)-1731(N) alternatively may comprise different types of graphics processing engines within a GPU, such as graphics execution units, media processing engines (e.g., video encoders/decoders), samplers, and blit engines. In at least one embodiment, graphics acceleration module 1746 may be a GPU with a plurality of graphics processing engines 1731(1)-1731(N) or graphics processing engines 1731(1)-1731(N) may be individual GPUs integrated on a common package, line card, or chip.

[0333] In at least one embodiment, accelerator integration circuit 1736 includes a memory management unit (MMU) 1739 for performing various memory management functions such as virtual-to-physical memory translations (also referred to as effective-to-real memory translations) and memory access protocols for accessing system memory 1714. In at least one embodiment, MMU 1739 may also include a translation lookaside buffer (TLB) (not shown) for caching virtual/effective to physical/real address translations. In at least one embodiment, a cache 1738 can store commands and data for efficient access by graphics processing engines 1731(1)-1731(N). In at least one embodiment, data stored in cache 1738 and graphics memories 1733(1)-1733(M) is kept coherent with core caches 1762A-1762D, 1756 and system memory 1714, possibly using a fetch unit 1744. As mentioned, this may be accomplished via proxy circuit 1725 on behalf of cache 1738 and memories 1733(1)-1733(M) (e.g., sending updates to cache 1738 related to modifications/accesses of cache lines on processor caches 1762A-1762D, 1756 and receiving updates from cache 1738).

[0334] In at least one embodiment, a set of registers 1745 store context data for threads executed by graphics processing engines 1731(1)-1731(N) and a context management circuit

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

[0335] In at least one embodiment, virtual/effective addresses from a graphics processing engine 1731 are translated to real/physical addresses in system memory 1714 by MMU 1739. In at least one embodiment, accelerator integration circuit 1736 supports multiple (e.g., 4, 8, 16) graphics accelerator modules 1746 and/or other accelerator devices. In at least one embodiment, graphics accelerator module 1746 may be dedicated to a single application executed on processor 1707 or may be shared between multiple applications. In at least one embodiment, a virtualized graphics execution environment is presented in which resources of graphics processing engines 1731(1)-1731(N) are shared with multiple applications or virtual machines (VMs). In at least one embodiment, resources may be subdivided into “slices” which are allocated to different VMs and/or applications based on processing requirements and priorities associated with VMs and/or applications.

[0336] In at least one embodiment, accelerator integration circuit 1736 performs as a bridge to a system for graphics acceleration module 1746 and provides address translation and system memory cache services. In addition, in at least one embodiment, accelerator integration circuit 1736 may provide virtualization facilities for a host processor to manage virtualization of graphics processing engines 1731(1)-1731(N), interrupts, and memory management.

[0337] In at least one embodiment, because hardware resources of graphics processing engines 1731(1)-1731(N) are mapped explicitly to a real address space seen by host processor 1707, any host processor can address these resources directly using an effective address value. In at least one embodiment, one function of accelerator integration circuit 1736 is physical separation of graphics processing engines 1731(1)-1731(N) so that they appear to a system as independent units.

[0338] In at least one embodiment, one or more graphics memories 1733(1)-1733(M) are coupled to each of graphics processing engines 1731(1)-1731(N), respectively and $N=M$. In at least one embodiment, graphics memories 1733(1)-1733(M) store instructions and data being processed by each of graphics processing engines 1731(1)-1731(N). In at least one embodiment, graphics memories 1733(1)-1733(M) may be volatile memories such as DRAMs (including stacked DRAMs), GDDR memory (e.g., GDDR5, GDDR6), or HBM, and/or may be non-volatile memories such as 3D XPoint or Nano-Ram.

[0339] In at least one embodiment, to reduce data traffic over high-speed link 1740, biasing techniques can be used to ensure that data stored in graphics memories 1733(1)-1733(M) is data that will be used most frequently by graphics processing engines 1731(1)-1731(N) and preferably not used by cores 1760A-1760D (at least not frequently). Similarly, in at least one embodiment, a biasing mechanism attempts to keep data needed by cores (and preferably not graphics processing engines 1731(1)-1731(N)) within caches 1762A-1762D, 1756 and system memory 1714.

[0340] FIG. 17C illustrates another exemplary embodiment in which accelerator integration circuit 1736 is integrated within processor 1707. In this embodiment, graphics processing engines 1731(1)-1731(N) communicate directly over high-speed link 1740 to accelerator integration circuit 1736 via interface 1737 and interface 1735 (which, again, may be any form of bus or interface protocol). In at least one embodiment, accelerator integration circuit 1736 may perform similar operations as those described with respect to FIG. 17B, but potentially at a higher throughput given its close proximity to coherence bus 1764 and caches 1762A-1762D, 1756. In at least one embodiment, an accelerator integration circuit supports different programming models including a dedicated-process programming model (no graphics acceleration module virtualization) and shared programming models (with virtualization), which may include programming models which are controlled by accelerator integration circuit 1736 and programming models which are controlled by graphics acceleration module 1746.

[0341] In at least one embodiment, graphics processing engines 1731(1)-1731(N) are dedicated to a single application or process under a single operating system. In at least one embodiment, a single application can funnel other application requests to graphics processing engines 1731(1)-1731(N), providing virtualization within a VM/partition.

[0342] In at least one embodiment, graphics processing engines 1731(1)-1731(N), may be shared by multiple VM/application partitions. In at least one embodiment, shared models may use a system hypervisor to virtualize graphics processing engines 1731(1)-1731(N) to allow access by each operating system. In at least one embodiment, for single-partition systems without a hypervisor, graphics processing engines 1731(1)-1731(N) are owned by an operating system. In at least one embodiment, an operating system can virtualize graphics processing engines 1731(1)-1731(N) to provide access to each process or application.

[0343] In at least one embodiment, graphics acceleration module 1746 or an individual graphics processing engine 1731(1)-1731(N) selects a process element using a process handle. In at least one embodiment, process elements are stored in system memory 1714 and are addressable using an effective address to real address translation technique described herein. In at least one embodiment, a process handle may be an implementation-specific value provided to a host process when registering its context with graphics processing engine 1731(1)-1731(N) (that is, calling system software to add a process element to a process element linked list). In at least one embodiment, a lower 16-bits of a process handle may be an offset of a process element within a process element linked list.

[0344] FIG. 17D illustrates an exemplary accelerator integration slice 1790. In at least one embodiment, a “slice” comprises a specified portion of processing resources of accelerator integration circuit 1736. In at least one embodiment, an application is effective address space 1782 within system memory 1714 stores process elements 1783. In at least one embodiment, process elements 1783 are stored in response to GPU invocations 1781 from applications 1780 executed on processor 1707. In at least one embodiment, a process element 1783 contains process state for corresponding application 1780. In at least one embodiment, a work descriptor (WD) 1784 contained in process element 1783 can be a single job requested by an application or may contain a pointer to a queue of jobs. In at least one embodiment, WD 1784 is a pointer to a job request queue in an application’s effective address space 1782.

[0345] In at least one embodiment, graphics acceleration module 1746 and/or individual graphics processing engines 1731(1)-1731(N) can be shared by all or a subset of processes in a system. In at least one embodiment, an infrastructure for setting up process states and sending a

WD 1784 to a graphics acceleration module 1746 to start a job in a virtualized environment may be included.

[0346] In at least one embodiment, a dedicated-process programming model is implementation-specific. In at least one embodiment, in this model, a single process owns graphics acceleration module 1746 or an individual graphics processing engine 1731. In at least one embodiment, when graphics acceleration module 1746 is owned by a single process, a hypervisor initializes accelerator integration circuit 1736 for an owning partition and an operating system initializes accelerator integration circuit 1736 for an owning process when graphics acceleration module 1746 is assigned.

[0347] In at least one embodiment, in operation, a WD fetch unit 1791 in accelerator integration slice 1790 fetches next WD 1784, which includes an indication of work to be done by one or more graphics processing engines of graphics acceleration module 1746. In at least one embodiment, data from WD 1784 may be stored in registers 1745 and used by MMU 1739, interrupt management circuit 1747 and/or context management circuit 1748 as illustrated. For example, one embodiment of MMU 1739 includes segment/page walk circuitry for accessing segment/page tables 1786 within an OS virtual address space 1785. In at least one embodiment, interrupt management circuit 1747 may process interrupt events 1792 received from graphics acceleration module 1746. In at least one embodiment, when performing graphics operations, an effective address 1793 generated by a graphics processing engine 1731(1)-1731(N) is translated to a real address by MMU 1739.

[0348] In at least one embodiment, registers 1745 are duplicated for each graphics processing engine 1731(1)-1731(N) and/or graphics acceleration module 1746 and may be 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 1790. Exemplary registers that may be initialized by a hypervisor are shown in Table 1.

Table 1 – Hypervisor Initialized Registers

<u>Register #</u>	<u>Description</u>
1	Slice Control Register
2	Real Address (RA) Scheduled Processes Area Pointer

3	Authority Mask Override Register
4	Interrupt Vector Table Entry Offset
5	Interrupt Vector Table Entry Limit
6	State Register
7	Logical Partition ID
8	Real address (RA) Hypervisor Accelerator Utilization Record Pointer
9	Storage Description Register

[0349] 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

[0350] In at least one embodiment, each WD 1784 is specific to a particular graphics acceleration module 1746 and/or graphics processing engines 1731(1)-1731(N). In at least one embodiment, it contains all information required by a graphics processing engine 1731(1)-1731(N) to do work, or it can be a pointer to a memory location where an application has set up a command queue of work to be completed.

[0351] FIG. 17E illustrates additional details for one exemplary embodiment of a shared model. This embodiment includes a hypervisor real address space 1798 in which a process element list 1799 is stored. In at least one embodiment, hypervisor real address space 1798 is accessible via a hypervisor 1796 which virtualizes graphics acceleration module engines for operating system 1795.

[0352] In at least one embodiment, shared programming models allow for all or a subset of processes from all or a subset of partitions in a system to use a graphics acceleration module 1746. In at least one embodiment, there are two programming models where graphics acceleration module 1746 is shared by multiple processes and partitions, namely time-sliced shared and graphics directed shared.

[0353] In at least one embodiment, in this model, system hypervisor 1796 owns graphics acceleration module 1746 and makes its function available to all operating systems 1795. In at least one embodiment, for a graphics acceleration module 1746 to support virtualization by system hypervisor 1796, graphics acceleration module 1746 may adhere to certain requirements, such as (1) an application's job request must be autonomous (that is, state does not need to be maintained between jobs), or graphics acceleration module 1746 must provide a context save and restore mechanism, (2) an application's job request is guaranteed by graphics acceleration module 1746 to complete in a specified amount of time, including any translation faults, or graphics acceleration module 1746 provides an ability to preempt processing of a job, and (3) graphics acceleration module 1746 must be guaranteed fairness between processes when operating in a directed shared programming model.

[0354] In at least one embodiment, application 1780 is required to make an operating system 1795 system call with a graphics acceleration module type, a work descriptor (WD), an authority mask register (AMR) value, and a context save/restore area pointer (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 1746 and can be in a form of a graphics acceleration module 1746 command, an effective address pointer to a user-defined structure, an effective address pointer to a queue of commands, or any other data structure to describe work to be done by graphics acceleration module 1746.

[0355] 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 1736 (not shown) and graphics acceleration module 1746 implementations do not support a User Authority Mask

Override Register (UAMOR), an operating system may apply a current UAMOR value to an AMR value before passing an AMR in a hypervisor call. In at least one embodiment, hypervisor 1796 may optionally apply a current Authority Mask Override Register (AMOR) value before placing an AMR into process element 1783. In at least one embodiment, CSRP is one of registers 1745 containing an effective address of an area in an application's effective address space 1782 for graphics acceleration module 1746 to save and restore context state. In at least one embodiment, this pointer is optional if no state is required to be saved between jobs or when a job is preempted. In at least one embodiment, context save/restore area may be pinned system memory.

[0356] Upon receiving a system call, operating system 1795 may verify that application 1780 has registered and been given authority to use graphics acceleration module 1746. In at least one embodiment, operating system 1795 then calls hypervisor 1796 with information shown in Table 3.

Table 3 – OS to Hypervisor Call Parameters

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

[0357] In at least one embodiment, upon receiving a hypervisor call, hypervisor 1796 verifies that operating system 1795 has registered and been given authority to use graphics acceleration module 1746. In at least one embodiment, hypervisor 1796 then puts process element 1783 into a process element linked list for a corresponding graphics acceleration module 1746 type. In at least one embodiment, a process element may include information shown in Table 4.

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)

[0358] In at least one embodiment, hypervisor initializes a plurality of accelerator integration slice 1790 registers 1745.

[0359] As illustrated in FIG. 17F, in at least one embodiment, a unified memory is used, addressable via a common virtual memory address space used to access physical processor memories 1701(1)-1701(N) and GPU memories 1720(1)-1720(N). In this implementation, operations executed on GPUs 1710(1)-1710(N) utilize a same virtual/effective memory address space to access processor memories 1701(1)-1701(M) and vice versa, thereby simplifying programmability. In at least one embodiment, a first portion of a virtual/effective address space is allocated to processor memory 1701(1), a second portion to second processor memory 1701(N), a third portion to GPU memory 1720(1), and so on. In at least one embodiment, an entire virtual/effective memory space (sometimes referred to as an effective address space) is thereby distributed across each of processor memories 1701 and GPU memories 1720, allowing any processor or GPU to access any physical memory with a virtual address mapped to that memory.

[0360] In at least one embodiment, bias/coherence management circuitry 1794A-1794E within one or more of MMUs 1739A-1739E ensures cache coherence between caches of one or more

host processors (e.g., 1705) and GPUs 1710 and implements biasing techniques indicating physical memories in which certain types of data should be stored. In at least one embodiment, while multiple instances of bias/coherence management circuitry 1794A-1794E are illustrated in FIG. 17F, bias/coherence circuitry may be implemented within an MMU of one or more host processors 1705 and/or within accelerator integration circuit 1736.

[0361] One embodiment allows GPU memories 1720 to be mapped as part of system memory, and accessed using shared virtual memory (SVM) technology, but without suffering performance drawbacks associated with full system cache coherence. In at least one embodiment, an ability for GPU memories 1720 to be accessed as system memory without onerous cache coherence overhead provides a beneficial operating environment for GPU offload. In at least one embodiment, this arrangement allows software of host processor 1705 to setup operands and access computation results, without overhead of traditional I/O DMA data copies. In at least one embodiment, such traditional copies involve driver calls, interrupts and memory mapped I/O (MMIO) accesses that are all inefficient relative to simple memory accesses. In at least one embodiment, an ability to access GPU memories 1720 without cache coherence overheads can be critical to execution time of an offloaded computation. In at least one embodiment, in cases with substantial streaming write memory traffic, for example, cache coherence overhead can significantly reduce an effective write bandwidth seen by a GPU 1710. In at least one embodiment, efficiency of operand setup, efficiency of results access, and efficiency of GPU computation may play a role in determining effectiveness of a GPU offload.

[0362] In at least one embodiment, selection of GPU bias and host processor bias is driven by a bias tracker data structure. In at least one embodiment, a bias table may be used, for example, which may be a page-granular structure (e.g., controlled at a granularity of a memory page) that includes 1 or 2 bits per GPU-attached memory page. In at least one embodiment, a bias table may be implemented in a stolen memory range of one or more GPU memories 1720, with or without a bias cache in a GPU 1710 (e.g., to cache frequently/recently used entries of a bias table). Alternatively, in at least one embodiment, an entire bias table may be maintained within a GPU.

[0363] In at least one embodiment, a bias table entry associated with each access to a GPU attached memory 1720 is accessed prior to actual access to a GPU memory, causing following

operations. In at least one embodiment, local requests from a GPU 1710 that find their page in GPU bias are forwarded directly to a corresponding GPU memory 1720. In at least one embodiment, local requests from a GPU that find their page in host bias are forwarded to processor 1705 (e.g., over a high-speed link as described herein). In at least one embodiment, requests from processor 1705 that find a requested page in host processor bias complete a request like a normal memory read. Alternatively, requests directed to a GPU-biased page may be forwarded to a GPU 1710. In at least one embodiment, a GPU may then 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.

[0364] 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 1705 bias to GPU bias, but is not for an opposite transition.

[0365] In at least one embodiment, cache coherency is maintained by temporarily rendering GPU-biased pages uncacheable by host processor 1705. In at least one embodiment, to access these pages, processor 1705 may request access from GPU 1710, which may or may not grant access right away. In at least one embodiment, thus, to reduce communication between processor 1705 and GPU 1710 it is beneficial to ensure that GPU-biased pages are those which are required by a GPU but not host processor 1705 and vice versa.

[0366] Hardware structure(s) 915 are used to perform one or more embodiments. Details regarding a hardware structure(s) 915 may be provided herein in conjunction with FIGS. 9A and/or 9B.

[0367] FIG. 18 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.

[0368] FIG. 18 is a block diagram illustrating an exemplary system on a chip integrated circuit 1800 that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, integrated circuit 1800 includes one or more application processor(s) 1805 (e.g., CPUs), at least one graphics processor 1810, and may additionally include an image processor 1815 and/or a video processor 1820, any of which may be a modular IP core. In at least one embodiment, integrated circuit 1800 includes peripheral or bus logic including a USB controller 1825, a UART controller 1830, an SPI/SDIO controller 1835, and an I22S/I22C controller 1840. In at least one embodiment, integrated circuit 1800 can include a display device 1845 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1850 and a mobile industry processor interface (MIPI) display interface 1855. In at least one embodiment, storage may be provided by a flash memory subsystem 1860 including flash memory and a flash memory controller. In at least one embodiment, a memory interface may be provided via a memory controller 1865 for access to SDRAM or SRAM memory devices. In at least one embodiment, some integrated circuits additionally include an embedded security engine 1870.

[0369] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in integrated circuit 1800 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0370] In at least one embodiment, at least one component shown or described with respect to FIGS. 17A-18 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, SOC 1800 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0371] FIGS. 19A-19B illustrate exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included in at

least one embodiment, including additional graphics processors/cores, peripheral interface controllers, or general-purpose processor cores.

[0372] FIGS. 19A-19B are block diagrams illustrating exemplary graphics processors for use within an SoC, according to embodiments described herein. FIG. 19A illustrates an exemplary graphics processor 1910 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. FIG. 19B illustrates an additional exemplary graphics processor 1940 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, graphics processor 1910 of FIG. 19A is a low power graphics processor core. In at least one embodiment, graphics processor 1940 of FIG. 19B is a higher performance graphics processor core. In at least one embodiment, each of graphics processors 1910, 1940 can be variants of graphics processor 1810 of FIG. 18.

[0373] In at least one embodiment, graphics processor 1910 includes a vertex processor 1905 and one or more fragment processor(s) 1915A-1915N (e.g., 1915A, 1915B, 1915C, 1915D, through 1915N-1, and 1915N). In at least one embodiment, graphics processor 1910 can execute different shader programs via separate logic, such that vertex processor 1905 is optimized to execute operations for vertex shader programs, while one or more fragment processor(s) 1915A-1915N execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. In at least one embodiment, vertex processor 1905 performs a vertex processing stage of a 3D graphics pipeline and generates primitives and vertex data. In at least one embodiment, fragment processor(s) 1915A-1915N use primitive and vertex data generated by vertex processor 1905 to produce a framebuffer that is displayed on a display device. In at least one embodiment, fragment processor(s) 1915A-1915N are optimized to execute fragment shader programs as provided for in an OpenGL API, which may be used to perform similar operations as a pixel shader program as provided for in a Direct 3D API.

[0374] In at least one embodiment, graphics processor 1910 additionally includes one or more memory management units (MMUs) 1920A-1920B, cache(s) 1925A-1925B, and circuit interconnect(s) 1930A-1930B. In at least one embodiment, one or more MMU(s) 1920A-1920B provide for virtual to physical address mapping for graphics processor 1910, including for vertex processor 1905 and/or fragment processor(s) 1915A-1915N, which may reference vertex or

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

[0375] In at least one embodiment, graphics processor 1940 includes one or more shader core(s) 1955A-1955N (e.g., 1955A, 1955B, 1955C, 1955D, 1955E, 1955F, through 1955N-1, and 1955N) as shown in FIG. 19B, which provides for a unified shader core architecture in 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 1940 includes an inter-core task manager 1945, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores 1955A-1955N and a tiling unit 1958 to accelerate tiling operations for tile-based rendering, in which rendering operations for a scene are subdivided in image space, for example to exploit local spatial coherence within a scene or to optimize use of internal caches.

[0376] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in graphic processor 1910 and/or 1940 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.

[0377] In at least one embodiment, at least one component shown or described with respect to FIGS. 19A-19B is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics processor 1910 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to

creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0378] FIGS. 20A-20B illustrate additional exemplary graphics processor logic according to embodiments described herein. In at least one embodiment, components illustrated in and described in connection with FIGS. 20A-20B are integrated into a single system, such as a graphics processing unit (GPU), SoC, or another type of processor. FIG. 20A illustrates a graphics core 2000 that may be included within graphics processor 1810 of FIG. 18, in at least one embodiment, and may be a unified shader core 1955A-1955N as in FIG. 19B in at least one embodiment. FIG. 20B illustrates a highly-parallel general-purpose graphics processing unit (“GPGPU”, which can also be referred to as a “graphics processing unit”) 2030 suitable for deployment on a multi-chip module in at least one embodiment. In at least one embodiment, graphics processing unit 2030 is a GPGPU that comprises a graphics processor. In at least one embodiment, integrated circuit 1800 comprises graphics core 2000, e.g., to form an integrated circuit and/or to form an SoC, where such an integrated circuit and/or such an SoC perform operations described herein.

[0379] In at least one embodiment, graphics core 2000 includes a shared instruction cache 2002, a texture unit 2018, and a cache/shared memory 2020 (e.g., including L1, L2, L3, last level cache, or other caches) that are common to execution resources within graphics core 2000. In at least one embodiment, graphics core 2000 can include multiple slices 2001A-2001N or a partition for each core, and a graphics processor can include multiple instances of graphics core 2000. In at least one embodiment, each slice 2001A-2001N refers to graphics core 2000. In at least one embodiment, slices 2001A-2001N have sub-slices, which are part of a slice 2001A-2001N. In at least one embodiment, slices 2001A-2001N are independent of other slices or dependent on other slices. In at least one embodiment, slices 2001A-2001N can include support logic including a local instruction cache 2004A-2004N, a thread scheduler (sequencer) 2006A-2006N, a thread dispatcher 2008A-2008N, and a set of registers 2010A-2010N. In at least one embodiment, slices 2001A-2001N can include a set of additional function units (AFUs 2012A-2012N), floating-point units (FPUs 2014A-2014N), integer arithmetic logic units (ALUs 2016A-2016N), address computational units (ACUs 2013A-2013N), double-precision floating-point

units (DPFPUs 2015A-2015N), and matrix processing units (MPUs 2017A-2017N). In at least one embodiment, MPUs 2017A-2017N are referred to as matrix engines.

[0380] In at least one embodiment, each slice 2001A-2001N includes one or more engines for floating point and integer vector operations and one or more engines to accelerate convolution and matrix operations in AI, machine learning, or large dataset workloads. In at least one embodiment, one or more slices 2001A-2001N include one or more vector engines to compute a vector (e.g., compute mathematical operations for vectors). In at least one embodiment, a vector engine can compute a vector operation in 16-bit floating point (also referred to as “FP16”), 32-bit floating point (also referred to as “FP32”), or 64-bit floating point (also referred to as “FP64”). In at least one embodiment, one or more slices 2001A-2001N includes 16 vector engines that are paired with 16 matrix math units to compute matrix/tensor operations, where vector engines and math units are exposed via matrix extensions. In at least one embodiment, a slice a specified portion of processing resources of a processing unit, e.g., 16 cores and a ray tracing unit or 8 cores, a thread scheduler, a thread dispatcher, and additional functional units for a processor. In at least one embodiment, graphics core 2000 includes one or more matrix engines to compute matrix operations, e.g., when computing tensor operations.

[0381] In at least one embodiment, one or more slices 2001A-2001N includes one or more ray tracing units to compute ray tracing operations (e.g., 16 ray tracing units per slice slices 2001A-2001N). In at least one embodiment, a ray tracing unit computes ray traversal, triangle intersection, bounding box intersect, or other ray tracing operations.

[0382] In at least one embodiment, one or more slices 2001A-2001N includes a media slice that encodes, decodes, and/or transcodes data; scales and/or format converts data; and/or performs video quality operations on video data.

[0383] In at least one embodiment, one or more slices 2001A-2001N are linked to L2 cache and memory fabric, link connectors, high-bandwidth memory (HBM) (e.g., HBM2e, HDM3) stacks, and a media engine. In at least one embodiment, one or more slices 2001A-2001N include multiple cores (e.g., 16 cores) and multiple ray tracing units (e.g., 16) paired to each core. In at least one embodiment, one or more slices 2001A-2001N has one or more L1 caches. In at least one embodiment, one or more slices 2001A-2001N include one or more vector engines; one or more instruction caches to store instructions; one or more L1 caches to cache data; one or

more shared local memories (SLMs) to store data, e.g., corresponding to instructions; one or more samplers to sample data; one or more ray tracing units to perform ray tracing operations; one or more geometries to perform operations in geometry pipelines and/or apply geometric transformations to vertices or polygons; one or more rasterizers to describe an image in vector graphics format (e.g., shape) and convert it into a raster image (e.g., a series of pixels, dots, or lines, which when displayed together, create an image that is represented by shapes) ; one or more a Hierarchical Depth Buffer (Hiz) to buffer data; and/or one or more pixel backends. In at least one embodiment, a slice 2001A-2001N includes a memory fabric, e.g., an L2 cache.

[0384] In at least one embodiment, FPUs 2014A-2014N can perform single-precision (32-bit) and half-precision (16-bit) floating point operations, while DPFPUs 2015A-2015N perform double precision (64-bit) floating point operations. In at least one embodiment, ALUs 2016A-2016N can perform variable precision integer operations at 8-bit, 16-bit, and 32-bit precision, and can be configured for mixed precision operations. In at least one embodiment, MPUs 2017A-2017N can also be configured for mixed precision matrix operations, including half-precision floating point and 8-bit integer operations. In at least one embodiment, MPUs 2017-2017N can perform a variety of matrix operations to accelerate machine learning application frameworks, including enabling support for accelerated general matrix to matrix multiplication (GEMM). In at least one embodiment, AFUs 2012A-2012N can perform additional logic operations not supported by floating-point or integer units, including trigonometric operations (e.g., sine, cosine, etc.).

[0385] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in graphics core 2000 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.

[0386] In at least one embodiment, graphics core 2000 includes an interconnect and a link fabric sublayer that is attached to a switch and a GPU-GPU bridge that enables multiple graphics processors 2000 (e.g., 8) to be interlinked without glue to each other with load/store units (LSUs), data transfer units, and sync semantics across multiple graphics processors 2000. In at least one

embodiment, interconnects include standardized interconnects (e.g., PCIe) or some combination thereof.

[0387] In at least one embodiment, graphics core 2000 includes multiple tiles. In at least one embodiment, a tile is an individual die or one or more dies, where individual dies can be connected with an interconnect (e.g., embedded multi-die interconnect bridge (EMIB)). In at least one embodiment, graphics core 2000 includes a compute tile, a memory tile (e.g., where a memory tile can be exclusively accessed by different tiles or different chipsets such as a Rambo tile), substrate tile, a base tile, a HMB tile, a link tile, and EMIB tile, where all tiles are packaged together in graphics core 2000 as part of a GPU. In at least one embodiment, graphics core 2000 can include multiple tiles in a single package (also referred to as a “multi tile package”). In at least one embodiment, a compute tile can have 8 graphics cores 2000, an L1 cache; and a base tile can have a host interface with PCIe 5.0, HBM2e, MDFI, and EMIB, a link tile with 8 links, 8 ports with an embedded switch. In at least one embodiment, tiles are connected with face-to-face (F2F) chip-on-chip bonding through fine-pitched, 36-micron, microbumps (e.g., copper pillars). In at least one embodiment, graphics core 2000 includes memory fabric, which includes memory, and is tile that is accessible by multiple tiles. In at least one embodiment, graphics core 2000 stores, accesses, or loads its own hardware contexts in memory, where a hardware context is a set of data loaded from registers before a process resumes, and where a hardware context can indicate a state of hardware (e.g., state of a GPU).

[0388] In at least one embodiment, graphics core 2000 includes serializer/deserializer (SERDES) circuitry that converts a serial data stream to a parallel data stream, or converts a parallel data stream to a serial data stream.

[0389] In at least one embodiment, graphics core 2000 includes a high speed coherent unified fabric (GPU to GPU), load/store units, bulk data transfer and sync semantics, and connected GPUs through an embedded switch, where a GPU-GPU bridge is controlled by a controller.

[0390] In at least one embodiment, graphics core 2000 performs an API, where said API abstracts hardware of graphics core 2000 and access libraries with instructions to perform math operations (e.g., math kernel library), deep neural network operations (e.g., deep neural network library), vector operations, collective communications, thread building blocks, video processing, data analytics library, and/or ray tracing operations.

[0391] In at least one embodiment, at least one component shown or described with respect to FIG. 20A is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics core 2000 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0392] FIG. 20B illustrates GPGPU 2030 that can be configured to enable highly-parallel compute operations to be performed by an array of graphics processing units, in at least one embodiment. In at least one embodiment, GPGPU 2030 can be linked directly to other instances of GPGPU 2030 to create a multi-GPU cluster to improve training speed for deep neural networks. In at least one embodiment, GPGPU 2030 includes a host interface 2032 to enable a connection with a host processor. In at least one embodiment, host interface 2032 is a PCI Express interface. In at least one embodiment, host interface 2032 can be a vendor-specific communications interface or communications fabric. In at least one embodiment, GPGPU 2030 receives commands from a host processor and uses a global scheduler 2034 (which may be referred to as a thread sequencer and/or asynchronous compute engine) to distribute execution threads associated with those commands to a set of compute clusters 2036A-2036H. In at least one embodiment, compute clusters 2036A-2036H share a cache memory 2038. In at least one embodiment, cache memory 2038 can serve as a higher-level cache for cache memories within compute clusters 2036A-2036H. In at least one embodiment, compute clusters 2036A-2036H comprise a slice or are referred to as “slices.” In at least one embodiment, GPGPU 2030 is part of an SoC such as part of integrated circuit 1800 (FIG. 18).

[0393] In at least one embodiment, GPGPU 2030 includes memory 2044A-2044B coupled with compute clusters 2036A-2036H via a set of memory controllers 2042A-2042B (e.g., one or more controllers for HBM2e). In at least one embodiment, memory 2044A-2044B can include various types of memory devices including dynamic random access memory (DRAM) or graphics random access memory, such as synchronous graphics random access memory (SGRAM), including graphics double data rate (GDDR) memory.

[0394] In at least one embodiment, compute clusters 2036A-2036H each include a set of graphics cores, such as graphics core 2000 of FIG. 20A, which can include multiple types of

integer and floating point logic units that can perform computational operations at a range of precisions including suited for machine learning computations. For example, in at least one embodiment, at least a subset of floating point units in each of compute clusters 2036A-2036H can be configured to perform 16-bit or 32-bit floating point operations, while a different subset of floating point units can be configured to perform 64-bit floating point operations.

[0395] In at least one embodiment, multiple instances of GPGPU 2030 can be configured to operate as a compute cluster. In at least one embodiment, communication used by compute clusters 2036A-2036H for synchronization and data exchange varies across embodiments. In at least one embodiment, multiple instances of GPGPU 2030 communicate over host interface 2032. In at least one embodiment, GPGPU 2030 includes an I/O hub 2039 that couples GPGPU 2030 with a GPU link 2040 that enables a direct connection to other instances of GPGPU 2030. In at least one embodiment, GPU link 2040 is coupled to a dedicated GPU-to-GPU bridge that enables communication and synchronization between multiple instances of GPGPU 2030. In at least one embodiment, GPU link 2040 couples with a high-speed interconnect to transmit and receive data to other GPGPUs or parallel processors. In at least one embodiment, multiple instances of GPGPU 2030 are located in separate data processing systems and communicate via a network device that is accessible via host interface 2032. In at least one embodiment GPU link 2040 can be configured to enable a connection to a host processor in addition to or as an alternative to host interface 2032.

[0396] In at least one embodiment, GPGPU 2030 can be configured to train neural networks. In at least one embodiment, GPGPU 2030 can be used within an inferencing platform. In at least one embodiment, in which GPGPU 2030 is used for inferencing, GPGPU 2030 may include fewer compute clusters 2036A-2036H relative to when GPGPU 2030 is used for training a neural network. In at least one embodiment, memory technology associated with memory 2044A-2044B may differ between inferencing and training configurations, with higher bandwidth memory technologies devoted to training configurations. In at least one embodiment, an inferencing configuration of GPGPU 2030 can support inferencing specific instructions. For example, in at least one embodiment, an inferencing configuration can provide support for one or more 8-bit integer dot product instructions, which may be used during inferencing operations for deployed neural networks.

[0397] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in GPGPU 2030 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0398] In at least one embodiment, at least one component shown or described with respect to FIG. 20B is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, GPGPU 2030 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0399] FIG. 21 is a block diagram illustrating a computing system 2100 according to at least one embodiment. In at least one embodiment, computing system 2100 includes a processing subsystem 2101 having one or more processor(s) 2102 and a system memory 2104 communicating via an interconnection path that may include a memory hub 2105. In at least one embodiment, memory hub 2105 may be a separate component within a chipset component or may be integrated within one or more processor(s) 2102. In at least one embodiment, memory hub 2105 couples with an I/O subsystem 2111 via a communication link 2106. In at least one embodiment, I/O subsystem 2111 includes an I/O hub 2107 that can enable computing system 2100 to receive input from one or more input device(s) 2108. In at least one embodiment, I/O hub 2107 can enable a display controller, which may be included in one or more processor(s) 2102, to provide outputs to one or more display device(s) 2110A. In at least one embodiment, one or more display device(s) 2110A coupled with I/O hub 2107 can include a local, internal, or embedded display device.

[0400] In at least one embodiment, processing subsystem 2101 includes one or more parallel processor(s) 2112 coupled to memory hub 2105 via a bus or other communication link 2113. In at least one embodiment, communication link 2113 may use one of any number of standards based communication link technologies or protocols, such as, but not limited to PCI Express, or may be a vendor-specific communications interface or communications fabric. In at least one

embodiment, one or more parallel processor(s) 2112 form a computationally focused parallel or vector processing system that can include a large number of processing cores and/or processing clusters, such as a many-integrated core (MIC) processor. In at least one embodiment, some or all of parallel processor(s) 2112 form a graphics processing subsystem that can output pixels to one of one or more display device(s) 2110A coupled via I/O Hub 2107. In at least one embodiment, parallel processor(s) 2112 can also include a display controller and display interface (not shown) to enable a direct connection to one or more display device(s) 2110B. In at least one embodiment, parallel processor(s) 2112 include one or more cores, such as graphics cores 2000 discussed herein.

[0401] In at least one embodiment, a system storage unit 2114 can connect to I/O hub 2107 to provide a storage mechanism for computing system 2100. In at least one embodiment, an I/O switch 2116 can be used to provide an interface mechanism to enable connections between I/O hub 2107 and other components, such as a network adapter 2118 and/or a wireless network adapter 2119 that may be integrated into platform, and various other devices that can be added via one or more add-in device(s) 2120. In at least one embodiment, network adapter 2118 can be an Ethernet adapter or another wired network adapter. In at least one embodiment, wireless network adapter 2119 can include one or more of a Wi-Fi, Bluetooth, near field communication (NFC), or other network device that includes one or more wireless radios.

[0402] In at least one embodiment, computing system 2100 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 2107. In at least one embodiment, communication paths interconnecting various components in FIG. 21 may be implemented using any suitable protocols, such as PCI (Peripheral Component Interconnect) based protocols (e.g., PCI-Express), or other bus or point-to-point communication interfaces and/or protocol(s), such as NV-Link high-speed interconnect, or interconnect protocols.

[0403] In at least one embodiment, parallel processor(s) 2112 incorporate circuitry optimized for graphics and video processing, including, for example, video output circuitry, and constitutes a graphics processing unit (GPU), e.g., parallel processor(s) 2112 includes graphics core 2000. In at least one embodiment, parallel processor(s) 2112 incorporate circuitry optimized for general purpose processing. In at least one embodiment, components of computing system 2100 may

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

[0404] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in computing system 2100 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0405] In at least one embodiment, at least one component shown or described with respect to FIG. 21 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, parallel processors 2112 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

PROCESSORS

[0406] FIG. 22A illustrates a parallel processor 2200 according to at least one embodiment. In at least one embodiment, various components of parallel processor 2200 may be implemented using one or more integrated circuit devices, such as programmable processors, application specific integrated circuits (ASICs), or field programmable gate arrays (FPGA). In at least one embodiment, illustrated parallel processor 2200 is a variant of one or more parallel processor(s) 2112 shown in FIG. 21 according to an exemplary embodiment. In at least one embodiment, a parallel processor 2200 includes one or more graphics cores 2000.

[0407] In at least one embodiment, parallel processor 2200 includes a parallel processing unit 2202. In at least one embodiment, parallel processing unit 2202 includes an I/O unit 2204 that

enables communication with other devices, including other instances of parallel processing unit 2202. In at least one embodiment, I/O unit 2204 may be directly connected to other devices. In at least one embodiment, I/O unit 2204 connects with other devices via use of a hub or switch interface, such as a memory hub 2205. In at least one embodiment, connections between memory hub 2205 and I/O unit 2204 form a communication link 2213. In at least one embodiment, I/O unit 2204 connects with a host interface 2206 and a memory crossbar 2216, where host interface 2206 receives commands directed to performing processing operations and memory crossbar 2216 receives commands directed to performing memory operations.

[0408] In at least one embodiment, when host interface 2206 receives a command buffer via I/O unit 2204, host interface 2206 can direct work operations to perform those commands to a front end 2208. In at least one embodiment, front end 2208 couples with a scheduler 2210 (which may be referred to as a sequencer), which is configured to distribute commands or other work items to a processing cluster array 2212. In at least one embodiment, scheduler 2210 ensures that processing cluster array 2212 is properly configured and in a valid state before tasks are distributed to a cluster of processing cluster array 2212. In at least one embodiment, scheduler 2210 is implemented via firmware logic executing on a microcontroller. In at least one embodiment, microcontroller implemented scheduler 2210 is configurable to perform complex scheduling and work distribution operations at coarse and fine granularity, enabling rapid preemption and context switching of threads executing on processing array 2212. In at least one embodiment, host software can provide workloads for scheduling on processing cluster array 2212 via one of multiple graphics processing paths. In at least one embodiment, workloads can then be automatically distributed across processing array cluster 2212 by scheduler 2210 logic within a microcontroller including scheduler 2210.

[0409] In at least one embodiment, processing cluster array 2212 can include up to “N” processing clusters (e.g., cluster 2214A, cluster 2214B, through cluster 2214N), where “N” represents a positive integer (which may be a different integer “N” than used in other figures). In at least one embodiment, each cluster 2214A-2214N of processing cluster array 2212 can execute a large number of concurrent threads. In at least one embodiment, scheduler 2210 can allocate work to clusters 2214A-2214N of processing cluster array 2212 using various scheduling and/or work distribution algorithms, which may vary depending on workload arising

for each type of program or computation. In at least one embodiment, scheduling can be handled dynamically by scheduler 2210, or can be assisted in part by compiler logic during compilation of program logic configured for execution by processing cluster array 2212. In at least one embodiment, different clusters 2214A-2214N of processing cluster array 2212 can be allocated for processing different types of programs or for performing different types of computations.

[0410] In at least one embodiment, processing cluster array 2212 can be configured to perform various types of parallel processing operations. In at least one embodiment, processing cluster array 2212 is configured to perform general-purpose parallel compute operations. For example, in at least one embodiment, processing cluster array 2212 can include logic to execute processing tasks including filtering of video and/or audio data, performing modeling operations, including physics operations, and performing data transformations.

[0411] In at least one embodiment, processing cluster array 2212 is configured to perform parallel graphics processing operations. In at least one embodiment, processing cluster array 2212 can include additional logic to support execution of such graphics processing operations, including but not limited to, texture sampling logic to perform texture operations, as well as tessellation logic and other vertex processing logic. In at least one embodiment, processing cluster array 2212 can be configured to execute graphics processing related shader programs such as, but not limited to, vertex shaders, tessellation shaders, geometry shaders, and pixel shaders. In at least one embodiment, parallel processing unit 2202 can transfer data from system memory via I/O unit 2204 for processing. In at least one embodiment, during processing, transferred data can be stored to on-chip memory (e.g., parallel processor memory 2222) during processing, then written back to system memory.

[0412] In at least one embodiment, when parallel processing unit 2202 is used to perform graphics processing, scheduler 2210 can be configured to divide a processing workload into approximately equal sized tasks, to better enable distribution of graphics processing operations to multiple clusters 2214A-2214N of processing cluster array 2212. In at least one embodiment, portions of processing cluster array 2212 can be configured to perform different types of processing. For example, in at least one embodiment, a first portion may be configured to perform vertex shading and topology generation, a second portion may be configured to perform tessellation and geometry shading, and a third portion may be configured to perform pixel

shading or other screen space operations, to produce a rendered image for display. In at least one embodiment, intermediate data produced by one or more of clusters 2214A-2214N may be stored in buffers to allow intermediate data to be transmitted between clusters 2214A-2214N for further processing.

[0413] In at least one embodiment, processing cluster array 2212 can receive processing tasks to be executed via scheduler 2210, which receives commands defining processing tasks from front end 2208. 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 2210 may be configured to fetch indices corresponding to tasks or may receive indices from front end 2208. In at least one embodiment, front end 2208 can be configured to ensure processing cluster array 2212 is configured to a valid state before a workload specified by incoming command buffers (e.g., batch-buffers, push buffers, etc.) is initiated.

[0414] In at least one embodiment, each of one or more instances of parallel processing unit 2202 can couple with a parallel processor memory 2222. In at least one embodiment, parallel processor memory 2222 can be accessed via memory crossbar 2216, which can receive memory requests from processing cluster array 2212 as well as I/O unit 2204. In at least one embodiment, memory crossbar 2216 can access parallel processor memory 2222 via a memory interface 2218. In at least one embodiment, memory interface 2218 can include multiple partition units (e.g., partition unit 2220A, partition unit 2220B, through partition unit 2220N) that can each couple to a portion (e.g., memory unit) of parallel processor memory 2222. In at least one embodiment, a number of partition units 2220A-2220N is configured to be equal to a number of memory units, such that a first partition unit 2220A has a corresponding first memory unit 2224A, a second partition unit 2220B has a corresponding memory unit 2224B, and an N-th partition unit 2220N has a corresponding N-th memory unit 2224N. In at least one embodiment, a number of partition units 2220A-2220N may not be equal to a number of memory units.

[0415] In at least one embodiment, memory units 2224A-2224N can include various types of memory devices, including dynamic random access memory (DRAM) or graphics random access memory, such as synchronous graphics random access memory (SGRAM), including graphics

double data rate (GDDR) memory. In at least one embodiment, memory units 2224A-2224N may also include 3D stacked memory, including but not limited to high bandwidth memory (HBM), HBM2e, or HBM3. In at least one embodiment, render targets, such as frame buffers or texture maps may be stored across memory units 2224A-2224N, allowing partition units 2220A-2220N to write portions of each render target in parallel to efficiently use available bandwidth of parallel processor memory 2222. In at least one embodiment, a local instance of parallel processor memory 2222 may be excluded in favor of a unified memory design that utilizes system memory in conjunction with local cache memory.

[0416] In at least one embodiment, any one of clusters 2214A-2214N of processing cluster array 2212 can process data that will be written to any of memory units 2224A-2224N within parallel processor memory 2222. In at least one embodiment, memory crossbar 2216 can be configured to transfer an output of each cluster 2214A-2214N to any partition unit 2220A-2220N or to another cluster 2214A-2214N, which can perform additional processing operations on an output. In at least one embodiment, each cluster 2214A-2214N can communicate with memory interface 2218 through memory crossbar 2216 to read from or write to various external memory devices. In at least one embodiment, memory crossbar 2216 has a connection to memory interface 2218 to communicate with I/O unit 2204, as well as a connection to a local instance of parallel processor memory 2222, enabling processing units within different processing clusters 2214A-2214N to communicate with system memory or other memory that is not local to parallel processing unit 2202. In at least one embodiment, memory crossbar 2216 can use virtual channels to separate traffic streams between clusters 2214A-2214N and partition units 2220A-2220N.

[0417] In at least one embodiment, multiple instances of parallel processing unit 2202 can be provided on a single add-in card, or multiple add-in cards can be interconnected. In at least one embodiment, different instances of parallel processing unit 2202 can be configured to interoperate even if different instances have different numbers of processing cores, different amounts of local parallel processor memory, and/or other configuration differences. For example, in at least one embodiment, some instances of parallel processing unit 2202 can include higher precision floating point units relative to other instances. In at least one embodiment, systems incorporating one or more instances of parallel processing unit 2202 or parallel

processor 2200 can be implemented in a variety of configurations and form factors, including but not limited to desktop, laptop, or handheld personal computers, servers, workstations, game consoles, and/or embedded systems.

[0418] FIG. 22B is a block diagram of a partition unit 2220 according to at least one embodiment. In at least one embodiment, partition unit 2220 is an instance of one of partition units 2220A-2220N of FIG. 22A. In at least one embodiment, partition unit 2220 includes an L2 cache 2221, a frame buffer interface 2225, and a ROP 2226 (raster operations unit). In at least one embodiment, L2 cache 2221 is a read/write cache that is configured to perform load and store operations received from memory crossbar 2216 and ROP 2226. In at least one embodiment, read misses and urgent write-back requests are output by L2 cache 2221 to frame buffer interface 2225 for processing. In at least one embodiment, updates can also be sent to a frame buffer via frame buffer interface 2225 for processing. In at least one embodiment, frame buffer interface 2225 interfaces with one of memory units in parallel processor memory, such as memory units 2224A-2224N of FIG. 22A (e.g., within parallel processor memory 2222).

[0419] In at least one embodiment, ROP 2226 is a processing unit that performs raster operations such as stencil, z test, blending, etc. In at least one embodiment, ROP 2226 then outputs processed graphics data that is stored in graphics memory. In at least one embodiment, ROP 2226 includes compression logic to compress depth or color data that is written to memory and decompress depth or color data that is read from memory. In at least one embodiment, compression logic can be lossless compression logic that makes use of one or more of multiple compression algorithms. In at least one embodiment, a type of compression that is performed by ROP 2226 can vary based on statistical characteristics of data to be compressed. For example, in at least one embodiment, delta color compression is performed on depth and color data on a per-tile basis.

[0420] In at least one embodiment, ROP 2226 is included within each processing cluster (e.g., cluster 2214A-2214N of FIG. 22A) instead of within partition unit 2220. In at least one embodiment, read and write requests for pixel data are transmitted over memory crossbar 2216 instead of pixel fragment data. In at least one embodiment, processed graphics data may be displayed on a display device, such as one of one or more display device(s) 2110 of FIG. 21,

routed for further processing by processor(s) 2102, or routed for further processing by one of processing entities within parallel processor 2200 of FIG. 22A.

[0421] FIG. 22C is a block diagram of a processing cluster 2214 within a parallel processing unit according to at least one embodiment. In at least one embodiment, a processing cluster is an instance of one of processing clusters 2214A-2214N of FIG. 22A. In at least one embodiment, processing cluster 2214 can be configured to execute many threads in parallel, where “thread” refers to an instance of a particular program executing on a particular set of input data. In at least one embodiment, single-instruction, multiple-data (SIMD) instruction issue techniques are used to support parallel execution of a large number of threads without providing multiple independent instruction units. In at least one embodiment, single-instruction, multiple-thread (SIMT) techniques are used to support parallel execution of a large number of generally synchronized threads, using a common instruction unit configured to issue instructions to a set of processing engines within each one of processing clusters.

[0422] In at least one embodiment, operation of processing cluster 2214 can be controlled via a pipeline manager 2232 that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager 2232 receives instructions from scheduler 2210 of FIG. 22A and manages execution of those instructions via a graphics multiprocessor 2234 and/or a texture unit 2236. In at least one embodiment, graphics multiprocessor 2234 is an exemplary instance of a SIMT parallel processor. However, in at least one embodiment, various types of SIMT parallel processors of differing architectures may be included within processing cluster 2214. In at least one embodiment, one or more instances of graphics multiprocessor 2234 can be included within a processing cluster 2214. In at least one embodiment, graphics multiprocessor 2234 can process data and a data crossbar 2240 can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager 2232 can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data crossbar 2240.

[0423] In at least one embodiment, each graphics multiprocessor 2234 within processing cluster 2214 can include an identical set of functional execution logic (e.g., arithmetic logic units, load-store units, etc.). In at least one embodiment, functional execution logic can be configured in a pipelined manner in which new instructions can be issued before previous instructions are

complete. In at least one embodiment, functional execution logic supports a variety of operations including integer and floating point arithmetic, comparison operations, Boolean operations, bit-shifting, and computation of various algebraic functions. In at least one embodiment, same functional-unit hardware can be leveraged to perform different operations and any combination of functional units may be present.

[0424] In at least one embodiment, instructions transmitted to processing cluster 2214 constitute a thread. In at least one embodiment, a set of threads executing across a set of parallel processing engines is a thread group. In at least one embodiment, a thread group executes a common program on different input data. In at least one embodiment, each thread within a thread group can be assigned to a different processing engine within a graphics multiprocessor 2234. In at least one embodiment, a thread group may include fewer threads than a number of processing engines within graphics multiprocessor 2234. In at least one 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 2234. In at least one embodiment, when a thread group includes more threads than number of processing engines within graphics multiprocessor 2234, processing can be performed over consecutive clock cycles. In at least one embodiment, multiple thread groups can be executed concurrently on a graphics multiprocessor 2234.

[0425] In at least one embodiment, graphics multiprocessor 2234 includes an internal cache memory to perform load and store operations. In at least one embodiment, graphics multiprocessor 2234 can forego an internal cache and use a cache memory (e.g., L1 cache 2248) within processing cluster 2214. In at least one embodiment, each graphics multiprocessor 2234 also has access to L2 caches within partition units (e.g., partition units 2220A-2220N of FIG. 22A) that are shared among all processing clusters 2214 and may be used to transfer data between threads. In at least one embodiment, graphics multiprocessor 2234 may also access off-chip global memory, which can include one or more of local parallel processor memory and/or system memory. In at least one embodiment, any memory external to parallel processing unit 2202 may be used as global memory. In at least one embodiment, processing cluster 2214

includes multiple instances of graphics multiprocessor 2234 and can share common instructions and data, which may be stored in L1 cache 2248.

[0426] In at least one embodiment, each processing cluster 2214 may include an MMU 2245 (memory management unit) that is configured to map virtual addresses into physical addresses. In at least one embodiment, one or more instances of MMU 2245 may reside within memory interface 2218 of FIG. 22A. In at least one embodiment, MMU 2245 includes a set of page table entries (PTEs) used to map a virtual address to a physical address of a tile and optionally a cache line index. In at least one embodiment, MMU 2245 may include address translation lookaside buffers (TLB) or caches that may reside within graphics multiprocessor 2234 or L1 2248 cache or processing cluster 2214. In at least one embodiment, a physical address is processed to distribute surface data access locally to allow for efficient request interleaving among partition units. In at least one embodiment, a cache line index may be used to determine whether a request for a cache line is a hit or miss.

[0427] In at least one embodiment, a processing cluster 2214 may be configured such that each graphics multiprocessor 2234 is coupled to a texture unit 2236 for performing texture mapping operations, e.g., determining texture sample positions, reading texture data, and filtering texture data. In at least one embodiment, texture data is read from an internal texture L1 cache (not shown) or from an L1 cache within graphics multiprocessor 2234 and is fetched from an L2 cache, local parallel processor memory, or system memory, as needed. In at least one embodiment, each graphics multiprocessor 2234 outputs processed tasks to data crossbar 2240 to provide processed task to another processing cluster 2214 for further processing or to store processed task in an L2 cache, local parallel processor memory, or system memory via memory crossbar 2216. In at least one embodiment, a preROP 2242 (pre-raster operations unit) is configured to receive data from graphics multiprocessor 2234, and direct data to ROP units, which may be located with partition units as described herein (e.g., partition units 2220A-2220N of FIG. 22A). In at least one embodiment, preROP 2242 unit can perform optimizations for color blending, organizing pixel color data, and performing address translations.

[0428] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in graphics processing

cluster 2214 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0429] In at least one embodiment, at least one component shown or described with respect to FIGS. 22A-22C is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, parallel processor 2200 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0430] FIG. 22D shows a graphics multiprocessor 2234 according to at least one embodiment. In at least one embodiment, graphics multiprocessor 2234 couples with pipeline manager 2232 of processing cluster 2214. In at least one embodiment, graphics multiprocessor 2234 has an execution pipeline including but not limited to an instruction cache 2252, an instruction unit 2254, an address mapping unit 2256, a register file 2258, one or more general purpose graphics processing unit (GPGPU) cores 2262, and one or more load/store units 2266, where one or more load/store units 2266 can perform load/store operations to load/store instructions corresponding to performing an operation. In at least one embodiment, GPGPU cores 2262 and load/store units 2266 are coupled with cache memory 2272 and shared memory 2270 via a memory and cache interconnect 2268. In at least one embodiment, GPGPU cores 2262 are part of an SoC such as part of integrated circuit 1800 in FIG. 18.

[0431] In at least one embodiment, instruction cache 2252 receives a stream of instructions to execute from pipeline manager 2232. In at least one embodiment, instructions are cached in instruction cache 2252 and dispatched for execution by an instruction unit 2254. In at least one embodiment, instruction unit 2254 can dispatch instructions as thread groups (e.g., warps, wavefronts, waves), with each thread of thread group assigned to a different execution unit within GPGPU cores 2262. In at least one embodiment, an instruction can access any of a local, shared, or global address space by specifying an address within a unified address space. In at least one embodiment, address mapping unit 2256 can be used to translate addresses in a unified address space into a distinct memory address that can be accessed by load/store units 2266.

[0432] In at least one embodiment, register file 2258 provides a set of registers for functional units of graphics multiprocessor 2234. In at least one embodiment, register file 2258 provides temporary storage for operands connected to data paths of functional units (e.g., GPGPU cores 2262, load/store units 2266) of graphics multiprocessor 2234. In at least one embodiment, register file 2258 is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file 2258. In at least one embodiment, register file 2258 is divided between different warps (which may be referred to as wavefronts and/or waves) being executed by graphics multiprocessor 2234.

[0433] In at least one embodiment, GPGPU cores 2262 can each include floating point units (FPUs) and/or integer arithmetic logic units (ALUs) that are used to execute instructions of graphics multiprocessor 2234. In at least one embodiment, GPGPU cores 2262 can be similar in architecture or can differ in architecture. In at least one embodiment, a first portion of GPGPU cores 2262 include a single precision FPU and an integer ALU while a second portion of GPGPU cores include a double precision FPU. In at least one embodiment, FPUs can implement IEEE 754-2008 standard floating point arithmetic or enable variable precision floating point arithmetic. In at least one embodiment, graphics multiprocessor 2234 can additionally include one or more fixed function or special function units to perform specific functions such as copy rectangle or pixel blending operations. In at least one embodiment, one or more of GPGPU cores 2262 can also include fixed or special function logic.

[0434] In at least one embodiment, GPGPU cores 2262 include SIMD logic capable of performing a single instruction on multiple sets of data. In at least one embodiment, GPGPU cores 2262 can physically execute SIMD4, SIMD8, and SIMD16 instructions and logically 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.

[0435] In at least one embodiment, memory and cache interconnect 2268 is an interconnect network that connects each functional unit of graphics multiprocessor 2234 to register file 2258 and to shared memory 2270. In at least one embodiment, memory and cache interconnect 2268 is a crossbar interconnect that allows load/store unit 2266 to implement load and store operations between shared memory 2270 and register file 2258. In at least one embodiment, register file 2258 can operate at a same frequency as GPGPU cores 2262, thus data transfer between GPGPU cores 2262 and register file 2258 can have very low latency. In at least one embodiment, shared memory 2270 can be used to enable communication between threads that execute on functional units within graphics multiprocessor 2234. In at least one embodiment, cache memory 2272 can be used as a data cache for example, to cache texture data communicated between functional units and texture unit 2236. In at least one embodiment, shared memory 2270 can also be used as a program managed cache. In at least one embodiment, threads executing on GPGPU cores 2262 can programmatically store data within shared memory in addition to automatically cached data that is stored within cache memory 2272.

[0436] 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, an SoC comprises a parallel processor or GPGPU as described herein, where said parallel processor or said GPGPU is performed on said SoC. 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.

[0437] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in graphics

multiprocessor 2234 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0438] In at least one embodiment, at least one component shown or described with respect to FIGS. 22D is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics multiprocessor 2234 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0439] FIG. 23 illustrates a multi-GPU computing system 2300, according to at least one embodiment. In at least one embodiment, multi-GPU computing system 2300 can include a processor 2302 coupled to multiple general purpose graphics processing units (GPGPUs) 2306A-D via a host interface switch 2304. In at least one embodiment, host interface switch 2304 is a PCI express switch device that couples processor 2302 to a PCI express bus over which processor 2302 can communicate with GPGPUs 2306A-D. In at least one embodiment, GPGPUs 2306A-D can interconnect via a set of high-speed point-to-point GPU-to-GPU links 2316. In at least one embodiment, GPU-to-GPU links 2316 connect to each of GPGPUs 2306A-D via a dedicated GPU link. In at least one embodiment, P2P GPU links 2316 enable direct communication between each of GPGPUs 2306A-D without requiring communication over host interface bus 2304 to which processor 2302 is connected. In at least one embodiment, with GPU-to-GPU traffic directed to P2P GPU links 2316, host interface bus 2304 remains available for system memory access or to communicate with other instances of multi-GPU computing system 2300, for example, via one or more network devices. While in at least one embodiment GPGPUs 2306A-D connect to processor 2302 via host interface switch 2304, in at least one embodiment processor 2302 includes direct support for P2P GPU links 2316 and can connect directly to GPGPUs 2306A-D. In at least one embodiment, GPGPUs 2306A-D is part of an SoC such as part of integrated circuit 1800 in FIG. 18, wherein GPGPUs 2306A-D performs operations described herein.

[0440] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with

FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in multi-GPU computing system 2300 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0441] In at least one embodiment, multi-GPU computing system 2300 includes one or more graphics cores 2000.

[0442] In at least one embodiment, at least one component shown or described with respect to FIG. 23 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, processor 2302 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0443] FIG. 24 is a block diagram of a graphics processor 2400, according to at least one embodiment. In at least one embodiment, graphics processor 2400 includes a ring interconnect 2402, a pipeline front-end 2404, a media engine 2437, and graphics cores 2480A-2480N. In at least one embodiment, ring interconnect 2402 couples graphics processor 2400 to other processing units, including other graphics processors or one or more general-purpose processor cores. In at least one embodiment, graphics processor 2400 is one of many processors integrated within a multi-core processing system. In at least one embodiment, graphics processor 2400 includes graphics core 2000.

[0444] In at least one embodiment, graphics processor 2400 receives batches of commands via ring interconnect 2402. In at least one embodiment, incoming commands are interpreted by a command streamer 2403 in pipeline front-end 2404. In at least one embodiment, graphics processor 2400 includes scalable execution logic to perform 3D geometry processing and media processing via graphics core(s) 2480A-2480N. In at least one embodiment, for 3D geometry processing commands, command streamer 2403 supplies commands to geometry pipeline 2436. In at least one embodiment, for at least some media processing commands, command streamer 2403 supplies commands to a video front end 2434, which couples with media engine 2437. In at least one embodiment, media engine 2437 includes a Video Quality Engine (VQE) 2430 for video and image post-processing and a multi-format encode/decode (MFX) 2433 engine to

provide hardware-accelerated media data encoding and decoding. In at least one embodiment, geometry pipeline 2436 and media engine 2437 each generate execution threads for thread execution resources provided by at least one graphics core 2480.

[0445] In at least one embodiment, graphics processor 2400 includes scalable thread execution resources featuring graphics cores 2480A-2480N (which can be modular and are sometimes referred to as core slices), each having multiple sub-cores 2450A-2450N, 2460A-2460N (sometimes referred to as core sub-slices). In at least one embodiment, graphics processor 2400 can have any number of graphics cores 2480A. In at least one embodiment, graphics processor 2400 includes a graphics core 2480A having at least a first sub-core 2450A and a second sub-core 2460A. In at least one embodiment, graphics processor 2400 is a low power processor with a single sub-core (e.g., 2450A). In at least one embodiment, graphics processor 2400 includes multiple graphics cores 2480A-2480N, each including a set of first sub-cores 2450A-2450N and a set of second sub-cores 2460A-2460N. In at least one embodiment, each sub-core in first sub-cores 2450A-2450N includes at least a first set of execution units 2452A-2452N and media/texture samplers 2454A-2454N. In at least one embodiment, each sub-core in second sub-cores 2460A-2460N includes at least a second set of execution units 2462A-2462N and samplers 2464A-2464N. In at least one embodiment, each sub-core 2450A-2450N, 2460A-2460N shares a set of shared resources 2470A-2470N. In at least one embodiment, shared resources include shared cache memory and pixel operation logic. In at least one embodiment, graphics processor 2400 includes load/store units in pipeline front-end 2404.

[0446] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in graphics processor 2400 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.

[0447] In at least one embodiment, at least one component shown or described with respect to FIG. 24 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics processor 2400 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an

efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0448] FIG. 25 is a block diagram illustrating micro-architecture for a processor 2500 that may include logic circuits to perform instructions, according to at least one embodiment. In at least one embodiment, processor 2500 may perform instructions, including x86 instructions, ARM instructions, specialized instructions for application-specific integrated circuits (ASICs), etc. In at least one embodiment, processor 2500 may include registers to store packed data, such as 64-bit wide MMX™ registers in microprocessors enabled with MMX technology from Intel Corporation of Santa Clara, Calif. In at least one embodiment, MMX registers, available in both integer and floating point forms, may operate with packed data elements that accompany single instruction, multiple data (“SIMD”) and streaming SIMD extensions (“SSE”) instructions. In at least one embodiment, 128-bit wide XMM registers relating to SSE2, SSE3, SSE4, AVX, or beyond (referred to generically as “SSEx”) technology may hold such packed data operands. In at least one embodiment, processor 2500 may perform instructions to accelerate machine learning or deep learning algorithms, training, or inferencing.

[0449] In at least one embodiment, processor 2500 includes an in-order front end (“front end”) 2501 to fetch instructions to be executed and prepare instructions to be used later in a processor pipeline. In at least one embodiment, front end 2501 may include several units. In at least one embodiment, an instruction prefetcher 2526 fetches instructions from memory and feeds instructions to an instruction decoder 2528 which in turn decodes or interprets instructions. For example, in at least one embodiment, instruction decoder 2528 decodes a received instruction into one or more operations called “micro-instructions” or “micro-operations” (also called “micro ops” or “uops” or “μ-ops”) that a machine may execute. In at least one embodiment, instruction decoder 2528 parses an instruction into an opcode and corresponding data and control fields that may be used by micro-architecture to perform operations in accordance with at least one embodiment. In at least one embodiment, a trace cache 2530 may assemble decoded uops into program ordered sequences or traces in a uop queue 2534 for execution. In at least one embodiment, when trace cache 2530 encounters a complex instruction, a microcode ROM 2532 provides uops needed to complete an operation.

[0450] In at least one embodiment, some instructions may be converted into a single micro-op, whereas others need several micro-ops to complete full operation. In at least one embodiment, if more than four micro-ops are needed to complete an instruction, instruction decoder 2528 may access microcode ROM 2532 to perform that instruction. In at least one embodiment, an instruction may be decoded into a small number of micro-ops for processing at instruction decoder 2528. In at least one embodiment, an instruction may be stored within microcode ROM 2532 should a number of micro-ops be needed to accomplish such operation. In at least one embodiment, trace cache 2530 refers to an entry point programmable logic array (“PLA”) to determine a correct micro-instruction pointer for reading microcode sequences to complete one or more instructions from microcode ROM 2532 in accordance with at least one embodiment. In at least one embodiment, after microcode ROM 2532 finishes sequencing micro-ops for an instruction, front end 2501 of a machine may resume fetching micro-ops from trace cache 2530.

[0451] In at least one embodiment, out-of-order execution engine (“out of order engine”) 2503 may prepare instructions for execution. In at least one embodiment, out-of-order execution logic has a number of buffers to smooth out and re-order flow of instructions to optimize performance as they go down a pipeline and get scheduled for execution. In at least one embodiment, out-of-order execution engine 2503 includes, without limitation, an allocator/register renamer 2540, a memory uop queue 2542, an integer/floating point uop queue 2544, a memory scheduler 2546, a fast scheduler 2502, a slow/general floating point scheduler (“slow/general FP scheduler”) 2504, and a simple floating point scheduler (“simple FP scheduler”) 2506. In at least one embodiment, fast scheduler 2502, slow/general floating point scheduler 2504, and simple floating point scheduler 2506 are also collectively referred to herein as “uop schedulers 2502, 2504, 2506.” In at least one embodiment, allocator/register renamer 2540 allocates machine buffers and resources that each uop needs in order to execute. In at least one embodiment, allocator/register renamer 2540 renames logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer 2540 also allocates an entry for each uop in one of two uop queues, memory uop queue 2542 for memory operations and integer/floating point uop queue 2544 for non-memory operations, in front of memory scheduler 2546 and uop schedulers 2502, 2504, 2506. In at least one embodiment, uop schedulers 2502, 2504, 2506, determine when a uop is ready to execute based on readiness of their dependent input register operand sources and availability of execution resources uops need to complete their operation. In at least one

embodiment, fast scheduler 2502 may schedule on each half of a main clock cycle while slow/general floating point scheduler 2504 and simple floating point scheduler 2506 may schedule once per main processor clock cycle. In at least one embodiment, uop schedulers 2502, 2504, 2506 arbitrate for dispatch ports to schedule uops for execution.

[0452] In at least one embodiment, execution block 2511 includes, without limitation, an integer register file/bypass network 2508, a floating point register file/bypass network (“FP register file/bypass network”) 2510, address generation units (“AGUs”) 2512 and 2514, fast Arithmetic Logic Units (ALUs) (“fast ALUs”) 2516 and 2518, a slow Arithmetic Logic Unit (“slow ALU”) 2520, a floating point ALU (“FP”) 2522, and a floating point move unit (“FP move”) 2524. In at least one embodiment, integer register file/bypass network 2508 and floating point register file/bypass network 2510 are also referred to herein as “register files 2508, 2510.” In at least one embodiment, AGUSs 2512 and 2514, fast ALUs 2516 and 2518, slow ALU 2520, floating point ALU 2522, and floating point move unit 2524 are also referred to herein as “execution units 2512, 2514, 2516, 2518, 2520, 2522, and 2524.” In at least one embodiment, execution block 2511 may include, without limitation, any number (including zero) and type of register files, bypass networks, address generation units, and execution units, in any combination.

[0453] In at least one embodiment, register networks 2508, 2510 may be arranged between uop schedulers 2502, 2504, 2506, and execution units 2512, 2514, 2516, 2518, 2520, 2522, and 2524. In at least one embodiment, integer register file/bypass network 2508 performs integer operations. In at least one embodiment, floating point register file/bypass network 2510 performs floating point operations. In at least one embodiment, each of register networks 2508, 2510 may include, without limitation, a bypass network that may bypass or forward just completed results that have not yet been written into a register file to new dependent uops. In at least one embodiment, register networks 2508, 2510 may communicate data with each other. In at least one embodiment, integer register file/bypass network 2508 may include, without limitation, two separate register files, one register file for a low-order thirty-two bits of data and a second register file for a high order thirty-two bits of data. In at least one embodiment, floating point register file/bypass network 2510 may include, without limitation, 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

[0454] In at least one embodiment, execution units 2512, 2514, 2516, 2518, 2520, 2522, 2524 may execute instructions. In at least one embodiment, register networks 2508, 2510 store integer and floating point data operand values that micro-instructions need to execute. In at least one embodiment, processor 2500 may include, without limitation, any number and combination of execution units 2512, 2514, 2516, 2518, 2520, 2522, 2524. In at least one embodiment, floating point ALU 2522 and floating point move unit 2524, may execute floating point, MMX, SIMD, AVX and SSE, or other operations, including specialized machine learning instructions. In at least one embodiment, floating point ALU 2522 may include, without limitation, a 64-bit by 64-bit floating point divider to execute divide, square root, and remainder micro ops. In at least one embodiment, instructions involving a floating point value may be handled with floating point hardware. In at least one embodiment, ALU operations may be passed to fast ALUs 2516, 2518. In at least one embodiment, fast ALUS 2516, 2518 may execute fast operations with an effective latency of half a clock cycle. In at least one embodiment, most complex integer operations go to slow ALU 2520 as slow ALU 2520 may include, without limitation, integer execution hardware for long-latency type of operations, such as a multiplier, shifts, flag logic, and branch processing. In at least one embodiment, memory load/store operations may be executed by AGUs 2512, 2514. In at least one embodiment, fast ALU 2516, fast ALU 2518, and slow ALU 2520 may perform integer operations on 64-bit data operands. In at least one embodiment, fast ALU 2516, fast ALU 2518, and slow ALU 2520 may be implemented to support a variety of data bit sizes including sixteen, thirty-two, 128, 256, etc. In at least one embodiment, floating point ALU 2522 and floating point move unit 2524 may be implemented to support a range of operands having bits of various widths, such as 128-bit wide packed data operands in conjunction with SIMD and multimedia instructions.

[0455] In at least one embodiment, uop schedulers 2502, 2504, 2506 dispatch dependent operations before a parent load has finished executing. In at least one embodiment, as uops may be speculatively scheduled and executed in processor 2500, processor 2500 may also 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.

[0456] 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.

[0457] In at least one embodiment, processor 2500 or each core of processor 2500 includes one or more prefetchers, one or more fetchers, one or more pre-decoders, one or more decoders to decode data (e.g., instructions), one or more instruction queues to process instructions (e.g., corresponding to operations or API calls), one or more micro-operation (μ OP) cache to store μ OPs, one or more micro-operation (μ OP) queues, an in-order execution engine, one or more load buffers, one or more store buffers, one or more reorder buffers, one or more fill buffers, an out-of-order execution engine, one or more ports, one or more shift and/or shifter units, one or more fused multiply accumulate (FMA) units, one or more load and store units (“LSUs”) to perform load of store operations corresponding to loading/storing data (e.g., instructions) to perform an operation (e.g., perform an API, an API call), one or more matrix multiply accumulate (MMA) units, and/or one or more shuffle units to perform any function further described herein with respect to said processor 2500. In at least one embodiment processor 2500 can access, use, perform, or execute instructions corresponding to calling an API.

[0458] In at least one embodiment, processor 2500 includes one or more ultra path interconnects (UPIs), e.g., that is a point-to-point processor interconnect; one or more PCIe’s; one or more accelerators to accelerate computations or operations; and/or one or more memory controllers. In at least one embodiment, processor 2500 includes a shared last level cache (LLC)

that is coupled to one or more memory controllers, which can enable shared memory access across processor cores.

[0459] In at least one embodiment, processor 2500 or a core of processor 2500 has a mesh architecture where processor cores, on-chip caches, memory controllers, and I/O controllers are organized in rows and columns, with wires and switches connecting them at each intersection to allow for turns. In at least one embodiment, processor 2500 has one or more higher memory bandwidths (HMBs, e.g., HMB_e) to store data or cache data, e.g., in Double Data Rate 5 Synchronous Dynamic Random-Access Memory (DDR5 SDRAM). In at least one embodiment, one or more components of processor 2500 are interconnected using compute express link (CXL) interconnects. In at least one embodiment, a memory controller uses a "least recently used" (LRU) approach to determine what gets stored in a cache. In at least one embodiment, processor 2500 includes one or more PCIe's (e.g., PCIe 5.0).

[0460] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of logic 915 may be incorporated into execution block 2511 and other memory or registers shown or not shown. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs illustrated in execution block 2511. Moreover, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of execution block 2511 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0461] In at least one embodiment, at least one component shown or described with respect to FIG. 25 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, processor 2500 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0462] FIG. 26 illustrates a deep learning application processor 2600, according to at least one embodiment. In at least one embodiment, deep learning application processor 2600 uses instructions that, if executed by deep learning application processor 2600, cause deep learning

application processor 2600 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, deep learning application processor 2600 is an application-specific integrated circuit (ASIC). In at least one embodiment, application processor 2600 performs matrix multiply operations either “hard-wired” into hardware as a result of performing one or more instructions or both. In at least one embodiment, deep learning application processor 2600 includes, without limitation, processing clusters 2610(1)-2610(12), Inter-Chip Links (“ICLs”) 2620(1)-2620(12), Inter-Chip Controllers (“ICCs”) 2630(1)-2630(2), high-bandwidth memory second generation (“HBM2”) 2640(1)-2640(4), memory controllers (“Mem Ctrlrs”) 2642(1)-2642(4), high bandwidth memory physical layer (“HBM PHY”) 2644(1)-2644(4), a management-controller central processing unit (“management-controller CPU”) 2650, a Serial Peripheral Interface, Inter-Integrated Circuit, and General Purpose Input/Output block (“SPI, I2C, GPIO”) 2660, a peripheral component interconnect express controller and direct memory access block (“PCIe Controller and DMA”) 2670, and a sixteen-lane peripheral component interconnect express port (“PCI Express x 16”) 2680.

[0463] In at least one embodiment, processing clusters 2610 may perform deep learning operations, including inference or prediction operations based on weight parameters calculated one or more training techniques, including those described herein. In at least one embodiment, each processing cluster 2610 may include, without limitation, any number and type of processors. In at least one embodiment, deep learning application processor 2600 may include any number and type of processing clusters 2600. In at least one embodiment, Inter-Chip Links 2620 are bi-directional. In at least one embodiment, Inter-Chip Links 2620 and Inter-Chip Controllers 2630 enable multiple deep learning application processors 2600 to exchange information, including activation information resulting from performing one or more machine learning algorithms embodied in one or more neural networks. In at least one embodiment, deep learning application processor 2600 may include any number (including zero) and type of ICLs 2620 and ICCs 2630.

[0464] In at least one embodiment, HBM2s 2640 provide a total of 32 Gigabytes (GB) of memory. In at least one embodiment, HBM2 2640(i) is associated with both memory controller 2642(i) and HBM PHY 2644(i) where “i” is an arbitrary integer. In at least one embodiment, any number of HBM2s 2640 may provide any type and total amount of high bandwidth memory and may be associated with any number (including zero) and type of memory controllers 2642

and HBM PHYs 2644. In at least one embodiment, SPI, I2C, GPIO 2660, PCIe Controller and DMA 2670, and/or PCIe 2680 may be replaced with any number and type of blocks that enable any number and type of communication standards in any technically feasible fashion.

[0465] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to deep learning application processor 2600. In at least one embodiment, deep learning application processor 2600 is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by deep learning application processor 2600. In at least one embodiment, processor 2600 may be used to perform one or more neural network use cases described herein.

[0466] In at least one embodiment, at least one component shown or described with respect to FIG. 26 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, deep learning application processor 2600 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0467] FIG. 27 is a block diagram of a neuromorphic processor 2700, according to at least one embodiment. In at least one embodiment, neuromorphic processor 2700 may receive one or more inputs from sources external to neuromorphic processor 2700. In at least one embodiment, these inputs may be transmitted to one or more neurons 2702 within neuromorphic processor 2700. In at least one embodiment, neurons 2702 and components thereof may be implemented using circuitry or logic, including one or more arithmetic logic units (ALUs). In at least one embodiment, neuromorphic processor 2700 may include, without limitation, thousands or millions of instances of neurons 2702, but any suitable number of neurons 2702 may be used. In at least one embodiment, each instance of neuron 2702 may include a neuron input 2704 and a neuron output 2706. In at least one embodiment, neurons 2702 may generate outputs that may be transmitted to inputs of other instances of neurons 2702. For example, in at least one

embodiment, neuron inputs 2704 and neuron outputs 2706 may be interconnected via synapses 2708.

[0468] In at least one embodiment, neurons 2702 and synapses 2708 may be interconnected such that neuromorphic processor 2700 operates to process or analyze information received by neuromorphic processor 2700. In at least one embodiment, neurons 2702 may transmit an output pulse (or “fire” or “spike”) when inputs received through neuron input 2704 exceed a threshold. In at least one embodiment, neurons 2702 may sum or integrate signals received at neuron inputs 2704. For example, in at least one embodiment, neurons 2702 may be implemented as leaky integrate-and-fire neurons, wherein if a sum (referred to as a “membrane potential”) exceeds a threshold value, neuron 2702 may generate an output (or “fire”) using a transfer function such as a sigmoid or threshold function. In at least one embodiment, a leaky integrate-and-fire neuron may sum signals received at neuron inputs 2704 into a membrane potential and may also apply a decay factor (or leak) to reduce a membrane potential. In at least one embodiment, a leaky integrate-and-fire neuron may fire if multiple input signals are received at neuron inputs 2704 rapidly enough to exceed a threshold value (i.e., before a membrane potential decays too low to fire). In at least one embodiment, neurons 2702 may be implemented using circuits or logic that receive inputs, integrate inputs into a membrane potential, and decay a membrane potential. In at least one embodiment, inputs may be averaged, or any other suitable transfer function may be used. Furthermore, in at least one embodiment, neurons 2702 may include, without limitation, comparator circuits or logic that generate an output spike at neuron output 2706 when result of applying a transfer function to neuron input 2704 exceeds a threshold. In at least one embodiment, once neuron 2702 fires, it may disregard previously received input information by, for example, resetting a membrane potential to 0 or another suitable default value. In at least one embodiment, once membrane potential is reset to 0, neuron 2702 may resume normal operation after a suitable period of time (or refractory period).

[0469] In at least one embodiment, neurons 2702 may be interconnected through synapses 2708. In at least one embodiment, synapses 2708 may operate to transmit signals from an output of a first neuron 2702 to an input of a second neuron 2702. In at least one embodiment, neurons 2702 may transmit information over more than one instance of synapse 2708. In at least one embodiment, one or more instances of neuron output 2706 may be connected, via an instance of

synapse 2708, to an instance of neuron input 2704 in same neuron 2702. In at least one embodiment, an instance of neuron 2702 generating an output to be transmitted over an instance of synapse 2708 may be referred to as a “pre-synaptic neuron” with respect to that instance of synapse 2708. In at least one embodiment, an instance of neuron 2702 receiving an input transmitted over an instance of synapse 2708 may be referred to as a “post-synaptic neuron” with respect to that instance of synapse 2708. Because an instance of neuron 2702 may receive inputs from one or more instances of synapse 2708, and may also transmit outputs over one or more instances of synapse 2708, a single instance of neuron 2702 may therefore be both a “pre-synaptic neuron” and “post-synaptic neuron,” with respect to various instances of synapses 2708, in at least one embodiment.

[0470] In at least one embodiment, neurons 2702 may be organized into one or more layers. In at least one embodiment, each instance of neuron 2702 may have one neuron output 2706 that may fan out through one or more synapses 2708 to one or more neuron inputs 2704. In at least one embodiment, neuron outputs 2706 of neurons 2702 in a first layer 2710 may be connected to neuron inputs 2704 of neurons 2702 in a second layer 2712. In at least one embodiment, layer 2710 may be referred to as a “feed-forward layer.” In at least one embodiment, each instance of neuron 2702 in an instance of first layer 2710 may fan out to each instance of neuron 2702 in second layer 2712. In at least one embodiment, first layer 2710 may be referred to as a “fully connected feed-forward layer.” In at least one embodiment, each instance of neuron 2702 in an instance of second layer 2712 may fan out to fewer than all instances of neuron 2702 in a third layer 2714. In at least one embodiment, second layer 2712 may be referred to as a “sparsely connected feed-forward layer.” In at least one embodiment, neurons 2702 in second layer 2712 may fan out to neurons 2702 in multiple other layers, including to neurons 2702 also in second layer 2712. In at least one embodiment, second layer 2712 may be referred to as a “recurrent layer.” In at least one embodiment, neuromorphic processor 2700 may include, without limitation, any suitable combination of recurrent layers and feed-forward layers, including, without limitation, both sparsely connected feed-forward layers and fully connected feed-forward layers.

[0471] In at least one embodiment, neuromorphic processor 2700 may include, without limitation, a reconfigurable interconnect architecture or dedicated hard-wired interconnects to

connect synapse 2708 to neurons 2702. In at least one embodiment, neuromorphic processor 2700 may include, without limitation, circuitry or logic that allows synapses to be allocated to different neurons 2702 as needed based on neural network topology and neuron fan-in/out. For example, in at least one embodiment, synapses 2708 may be connected to neurons 2702 using an interconnect fabric, such as network-on-chip, or with dedicated connections. In at least one embodiment, synapse interconnections and components thereof may be implemented using circuitry or logic.

[0472] In at least one embodiment, at least one component shown or described with respect to FIGS. 27 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, neuromorphic processor 2700 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0473] FIG. 28 is a block diagram of a processing system, according to at least one embodiment. In at least one embodiment, system 2800 includes one or more processors 2802 and one or more graphics processors 2808, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 2802 or processor cores 2807. In at least one embodiment, system 2800 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices. In at least one embodiment, one or more graphics processors 2808 include one or more graphics cores 2000.

[0474] In at least one embodiment, system 2800 can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming console, a handheld game console, or an online game console. In at least one embodiment, system 2800 is a mobile phone, a smart phone, a tablet computing device or a mobile Internet device. In at least one embodiment, processing system 2800 can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, a smart eyewear device, an augmented reality device, or a virtual reality device. In at least one embodiment, processing system 2800 is a television or set top box device having one or more processors 2802 and a graphical interface generated by one or more graphics processors 2808.

[0475] In at least one embodiment, one or more processors 2802 each include one or more processor cores 2807 to process instructions which, when executed, perform operations for system and user software. In at least one embodiment, each of one or more processor cores 2807 is configured to process a specific instruction sequence 2809. In at least one embodiment, instruction sequence 2809 may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). In at least one embodiment, processor cores 2807 may each process a different instruction sequence 2809, which may include instructions to facilitate emulation of other instruction sequences. In at least one embodiment, processor core 2807 may also include other processing devices, such as a Digital Signal Processor (DSP).

[0476] In at least one embodiment, processor 2802 includes a cache memory 2804. In at least one embodiment, processor 2802 can have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory is shared among various components of processor 2802. In at least one embodiment, processor 2802 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among processor cores 2807 using known cache coherency techniques. In at least one embodiment, a register file 2806 is additionally included in processor 2802, which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). In at least one embodiment, register file 2806 may include general-purpose registers or other registers.

[0477] In at least one embodiment, one or more processor(s) 2802 are coupled with one or more interface bus(es) 2810 to transmit communication signals such as address, data, or control signals between processor 2802 and other components in system 2800. In at least one embodiment, interface bus 2810 can be a processor bus, such as a version of a Direct Media Interface (DMI) bus. In at least one embodiment, interface bus 2810 is not limited to a DMI bus, and may include one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express), memory busses, or other types of interface busses. In at least one embodiment processor(s) 2802 include an integrated memory controller 2816 and a platform controller hub 2830. In at least one embodiment, memory controller 2816 facilitates communication between a memory device and

other components of system 2800, while platform controller hub (PCH) 2830 provides connections to I/O devices via a local I/O bus.

[0478] In at least one embodiment, a memory device 2820 can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as process memory. In at least one embodiment, memory device 2820 can operate as system memory for system 2800, to store data 2822 and instructions 2821 for use when one or more processors 2802 executes an application or process. In at least one embodiment, memory controller 2816 also couples with an optional external graphics processor 2812, which may communicate with one or more graphics processors 2808 in processors 2802 to perform graphics and media operations. In at least one embodiment, a display device 2811 can connect to processor(s) 2802. In at least one embodiment, display device 2811 can include one or more of an internal display device, as in a mobile electronic device or a laptop device, or an external display device attached via a display interface (e.g., DisplayPort, etc.). In at least one embodiment, display device 2811 can include a head mounted display (HMD) such as a stereoscopic display device for use in virtual reality (VR) applications or augmented reality (AR) applications.

[0479] In at least one embodiment, platform controller hub 2830 enables peripherals to connect to memory device 2820 and processor 2802 via a high-speed I/O bus. In at least one embodiment, I/O peripherals include, but are not limited to, an audio controller 2846, a network controller 2834, a firmware interface 2828, a wireless transceiver 2826, touch sensors 2825, a data storage device 2824 (e.g., hard disk drive, flash memory, etc.). In at least one embodiment, data storage device 2824 can connect via a storage interface (e.g., SATA) or via a peripheral bus, such as a Peripheral Component Interconnect bus (e.g., PCI, PCI Express). In at least one embodiment, touch sensors 2825 can include touch screen sensors, pressure sensors, or fingerprint sensors. In at least one embodiment, wireless transceiver 2826 can be a Wi-Fi transceiver, a Bluetooth transceiver, or a mobile network transceiver such as a 3G, 4G, or Long Term Evolution (LTE) transceiver. In at least one embodiment, firmware interface 2828 enables communication with system firmware, and can be, for example, a unified extensible firmware interface (UEFI). In at least one embodiment, network controller 2834 can enable a network

connection to a wired network. In at least one embodiment, a high-performance network controller (not shown) couples with interface bus 2810. In at least one embodiment, audio controller 2846 is a multi-channel high definition audio controller. In at least one embodiment, system 2800 includes an optional legacy I/O controller 2840 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to system 2800. In at least one embodiment, platform controller hub 2830 can also connect to one or more Universal Serial Bus (USB) controllers 2842 connect input devices, such as keyboard and mouse 2843 combinations, a camera 2844, or other USB input devices.

[0480] In at least one embodiment, an instance of memory controller 2816 and platform controller hub 2830 may be integrated into a discreet external graphics processor, such as external graphics processor 2812. In at least one embodiment, platform controller hub 2830 and/or memory controller 2816 may be external to one or more processor(s) 2802. For example, in at least one embodiment, system 2800 can include an external memory controller 2816 and platform controller hub 2830, which may be configured as a memory controller hub and peripheral controller hub within a system chipset that is in communication with processor(s) 2802.

[0481] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of logic 915 may be incorporated into graphics processor 2808. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in a 3D pipeline. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 2808 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0482] In at least one embodiment, at least one component shown or described with respect to FIG. 28 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, processor(s) 2802 performs one or more operations related

to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0483] FIG. 29 is a block diagram of a processor 2900 having one or more processor cores 2902A-2902N, an integrated memory controller 2914, and an integrated graphics processor 2908, according to at least one embodiment. In at least one embodiment, processor 2900 can include additional cores up to and including additional core 2902N represented by dashed lined boxes. In at least one embodiment, each of processor cores 2902A-2902N includes one or more internal cache units 2904A-2904N. In at least one embodiment, each processor core also has access to one or more shared cache units 2906. In at least one embodiment, graphics processor 2908 includes one or more graphics cores 2000.

[0484] In at least one embodiment, internal cache units 2904A-2904N and shared cache units 2906 represent a cache memory hierarchy within processor 2900. In at least one embodiment, cache memory units 2904A-2904N may include at least one level of instruction and data cache within each processor core and one or more levels of shared mid-level cache, such as a Level 2 (L2), Level 3 (L3), Level 4 (L4), or other levels of cache, where a highest level of cache before external memory is classified as an LLC. In at least one embodiment, cache coherency logic maintains coherency between various cache units 2906 and 2904A-2904N.

[0485] In at least one embodiment, processor 2900 may also include a set of one or more bus controller units 2916 and a system agent core 2910. In at least one embodiment, bus controller units 2916 manage a set of peripheral buses, such as one or more PCI or PCI express busses. In at least one embodiment, system agent core 2910 provides management functionality for various processor components. In at least one embodiment, system agent core 2910 includes one or more integrated memory controllers 2914 to manage access to various external memory devices (not shown).

[0486] In at least one embodiment, one or more of processor cores 2902A-2902N include support for simultaneous multi-threading. In at least one embodiment, system agent core 2910 includes components for coordinating and operating cores 2902A-2902N during multi-threaded processing. In at least one embodiment, system agent core 2910 may additionally include a

power control unit (PCU), which includes logic and components to regulate one or more power states of processor cores 2902A-2902N and graphics processor 2908.

[0487] In at least one embodiment, processor 2900 additionally includes graphics processor 2908 to execute graphics processing operations. In at least one embodiment, graphics processor 2908 couples with shared cache units 2906, and system agent core 2910, including one or more integrated memory controllers 2914. In at least one embodiment, system agent core 2910 also includes a display controller 2911 to drive graphics processor output to one or more coupled displays. In at least one embodiment, display controller 2911 may also be a separate module coupled with graphics processor 2908 via at least one interconnect, or may be integrated within graphics processor 2908.

[0488] In at least one embodiment, a ring-based interconnect unit 2912 is used to couple internal components of processor 2900. In at least one embodiment, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques. In at least one embodiment, graphics processor 2908 couples with ring interconnect 2912 via an I/O link 2913.

[0489] In at least one embodiment, I/O link 2913 represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded memory module 2918, such as an eDRAM module. In at least one embodiment, each of processor cores 2902A-2902N and graphics processor 2908 use embedded memory module 2918 as a shared Last Level Cache.

[0490] In at least one embodiment, processor cores 2902A-2902N are homogeneous cores executing a common instruction set architecture. In at least one embodiment, processor cores 2902A-2902N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 2902A-2902N execute a common instruction set, while one or more other cores of processor cores 2902A-2902N executes a subset of a common instruction set or a different instruction set. In at least one embodiment, processor cores 2902A-2902N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. In at least one embodiment, processor 2900 can be implemented on one or more chips or as an SoC integrated circuit.

[0491] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of logic 915 may be incorporated into graphics processor 2908. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in a 3D pipeline, graphics core(s) 2902, shared function logic, or other logic in FIG. 29. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of processor 2900 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0492] In at least one embodiment, at least one component shown or described with respect to FIG. 29 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, processor 2900 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0493] FIG. 30 is a block diagram of a graphics processor 3000, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. In at least one embodiment, graphics processor 3000 communicates via a memory mapped I/O interface to registers on graphics processor 3000 and with commands placed into memory. In at least one embodiment, graphics processor 3000 includes a memory interface 3014 to access memory. In at least one embodiment, memory interface 3014 is an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory. In at least one embodiment, graphics processor 3000 includes graphics core 2000.

[0494] In at least one embodiment, graphics processor 3000 also includes a display controller 3002 to drive display output data to a display device 3020. In at least one embodiment, display controller 3002 includes hardware for one or more overlay planes for display device 3020 and composition of multiple layers of video or user interface elements. In at least one embodiment, display device 3020 can be an internal or external display device. In at least one embodiment,

display device 3020 is a head mounted display device, such as a virtual reality (VR) display device or an augmented reality (AR) display device. In at least one embodiment, graphics processor 3000 includes a video codec engine 3006 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

[0495] In at least one embodiment, graphics processor 3000 includes a block image transfer (BLIT) engine 3004 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in at least one embodiment, 2D graphics operations are performed using one or more components of a graphics processing engine (GPE) 3010. In at least one embodiment, GPE 3010 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0496] In at least one embodiment, GPE 3010 includes a 3D pipeline 3012 for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). In at least one embodiment, 3D pipeline 3012 includes programmable and fixed function elements that perform various tasks and/or spawn execution threads to a 3D/Media sub-system 3015. While 3D pipeline 3012 can be used to perform media operations, in at least one embodiment, GPE 3010 also includes a media pipeline 3016 that is used to perform media operations, such as video post-processing and image enhancement.

[0497] In at least one embodiment, media pipeline 3016 includes fixed function or programmable logic units to perform one or more specialized media operations, such as video decode acceleration, video de-interlacing, and video encode acceleration in place of, or on behalf of, video codec engine 3006. In at least one embodiment, media pipeline 3016 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 3015. In at least one embodiment, spawned threads perform computations for media operations on one or more graphics execution units included in 3D/Media sub-system 3015.

[0498] In at least one embodiment, 3D/Media subsystem 3015 includes logic for executing threads spawned by 3D pipeline 3012 and media pipeline 3016. In at least one embodiment, 3D pipeline 3012 and media pipeline 3016 send thread execution requests to 3D/Media subsystem 3015, which includes thread dispatch logic for arbitrating and dispatching various requests to available thread execution resources. In at least one embodiment, execution resources include an array of graphics execution units to process 3D and media threads. In at least one embodiment, 3D/Media subsystem 3015 includes one or more internal caches for thread instructions and data. In at least one embodiment, subsystem 3015 also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

[0499] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of logic 915 may be incorporated into graphics processor 3000. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in 3D pipeline 3012. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 3000 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0500] In at least one embodiment, at least one component shown or described with respect to FIG. 30 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics processor 3000 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0501] FIG. 31 is a block diagram of a graphics processing engine 3110 of a graphics processor in accordance with at least one embodiment. In at least one embodiment, graphics processing engine (GPE) 3110 is a version of GPE 3010 shown in FIG. 30. In at least one embodiment, a media pipeline 3116 is optional and may not be explicitly included within GPE

3110. In at least one embodiment, a separate media and/or image processor is coupled to GPE 3110.

[0502] In at least one embodiment, GPE 3110 is coupled to or includes a command streamer 3103, which provides a command stream to a 3D pipeline 3112 and/or media pipeline 3116. In at least one embodiment, command streamer 3103 is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In at least one embodiment, command streamer 3103 receives commands from memory and sends commands to 3D pipeline 3112 and/or media pipeline 3116. In at least one embodiment, commands are instructions, primitives, or micro-operations fetched from a ring buffer, which stores commands for 3D pipeline 3112 and media pipeline 3116. In at least one embodiment, a ring buffer can additionally include batch command buffers storing batches of multiple commands. In at least one embodiment, commands for 3D pipeline 3112 can also include references to data stored in memory, such as, but not limited to, vertex and geometry data for 3D pipeline 3112 and/or image data and memory objects for media pipeline 3116. In at least one embodiment, 3D pipeline 3112 and media pipeline 3116 process commands and data by performing operations or by dispatching one or more execution threads to a graphics core array 3114. In at least one embodiment, graphics core array 3114 includes one or more blocks of graphics cores (e.g., graphics core(s) 3115A, graphics core(s) 3115B), each block including one or more graphics cores. In at least one embodiment, graphics core(s) 3115A, 3115B may be referred to as execution units (“EUs”). In at least one embodiment, each graphics core includes a set of graphics execution resources that includes general-purpose and graphics specific execution logic to perform graphics and compute operations, as well as fixed function texture processing and/or machine learning and artificial intelligence acceleration logic, including inference and/or training logic 915 in FIG. 9A and FIG. 9B.

[0503] In at least one embodiment, 3D pipeline 3112 includes fixed function and programmable logic to process one or more shader programs, such as vertex shaders, geometry shaders, pixel shaders, fragment shaders, compute shaders, or other shader programs, by processing instructions and dispatching execution threads to graphics core array 3114. In at least one embodiment, graphics core array 3114 provides a unified block of execution resources for use in processing shader programs. In at least one embodiment, a multi-purpose execution logic

(e.g., execution units) within graphics core(s) 3115A-3115B of graphic core array 3114 includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

[0504] In at least one embodiment, graphics core array 3114 also includes execution logic to perform media functions, such as video and/or image processing. In at least one embodiment, execution units additionally include general-purpose logic that is programmable to perform parallel general-purpose computational operations, in addition to graphics processing operations.

[0505] In at least one embodiment, output data generated by threads executing on graphics core array 3114 can output data to memory in a unified return buffer (URB) 3118. In at least one embodiment, URB 3118 can store data for multiple threads. In at least one embodiment, URB 3118 may be used to send data between different threads executing on graphics core array 3114. In at least one embodiment, URB 3118 may additionally be used for synchronization between threads on graphics core array 3114 and fixed function logic within shared function logic 3120.

[0506] In at least one embodiment, graphics core array 3114 is scalable, such that graphics core array 3114 includes a variable number of graphics cores, each having a variable number of execution units based on a target power and performance level of GPE 3110. In at least one embodiment, execution resources are dynamically scalable, such that execution resources may be enabled or disabled as needed.

[0507] In at least one embodiment, graphics core array 3114 is coupled to shared function logic 3120 that includes multiple resources that are shared between graphics cores in graphics core array 3114. In at least one embodiment, shared functions performed by shared function logic 3120 are embodied in hardware logic units that provide specialized supplemental functionality to graphics core array 3114. In at least one embodiment, shared function logic 3120 includes but is not limited to a sampler unit 3121, a math unit 3122, and inter-thread communication (ITC) logic 3123. In at least one embodiment, one or more cache(s) 3125 are included in, or coupled to, shared function logic 3120.

[0508] In at least one embodiment, a shared function is used if demand for a specialized function is insufficient for inclusion within graphics core array 3114. In at least one embodiment, a single instantiation of a specialized function is used in shared function logic 3120 and shared

among other execution resources within graphics core array 3114. In at least one embodiment, specific shared functions within shared function logic 3120 that are used extensively by graphics core array 3114 may be included within shared function logic 3126 within graphics core array 3114. In at least one embodiment, shared function logic 3126 within graphics core array 3114 can include some or all logic within shared function logic 3120. In at least one embodiment, all logic elements within shared function logic 3120 may be duplicated within shared function logic 3126 of graphics core array 3114. In at least one embodiment, shared function logic 3120 is excluded in favor of shared function logic 3126 within graphics core array 3114.

[0509] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of logic 915 may be incorporated into graphics processor 3110. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in 3D pipeline 3112, graphics core(s) 3115, shared function logic 3126, shared function logic 3120, or other logic in FIG. 31. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor 3110 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0510] In at least one embodiment, at least one component shown or described with respect to FIG. 31 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics processing engine 3110 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0511] FIG. 32 is a block diagram of hardware logic of a graphics processor core 3200, according to at least one embodiment described herein. In at least one embodiment, graphics processor core 3200 includes graphics core 2000. In at least one embodiment, graphics processor core 3200 is included within a graphics core array. In at least one embodiment,

graphics processor core 3200, sometimes referred to as a core slice, can be one or multiple graphics cores within a modular graphics processor. In at least one embodiment, graphics processor core 3200 is exemplary of one graphics core slice, and a graphics processor as described herein may include multiple graphics core slices based on target power and performance envelopes. In at least one embodiment, each graphics core 3200 can include a fixed function block 3230 coupled with multiple sub-cores 3201A-3201F, also referred to as sub-slices, that include modular blocks of general-purpose and fixed function logic.

[0512] In at least one embodiment, fixed function block 3230 includes a geometry and fixed function pipeline 3236 that can be shared by all sub-cores in graphics processor 3200, for example, in lower performance and/or lower power graphics processor implementations. In at least one embodiment, geometry and fixed function pipeline 3236 includes a 3D fixed function pipeline, a video front-end unit, a thread spawner and thread dispatcher, and a unified return buffer manager, which manages unified return buffers.

[0513] In at least one embodiment, fixed function block 3230 also includes a graphics SoC interface 3237, a graphics microcontroller 3238, and a media pipeline 3239. In at least one embodiment, graphics SoC interface 3237 provides an interface between graphics core 3200 and other processor cores within a system on a chip integrated circuit. In at least one embodiment, graphics microcontroller 3238 is a programmable sub-processor that is configurable to manage various functions of graphics processor 3200, including thread dispatch, scheduling, and pre-emption. In at least one embodiment, media pipeline 3239 includes logic to facilitate decoding, encoding, pre-processing, and/or post-processing of multimedia data, including image and video data. In at least one embodiment, media pipeline 3239 implements media operations via requests to compute or sampling logic within sub-cores 3201A-3201F.

[0514] In at least one embodiment, SoC interface 3237 enables graphics core 3200 to communicate with general-purpose application processor cores (e.g., CPUs) and/or other components within an SoC, including memory hierarchy elements such as a shared last level cache memory, system RAM, and/or embedded on-chip or on-package DRAM. In at least one embodiment, SoC interface 3237 can also enable communication with fixed function devices within an SoC, such as camera imaging pipelines, and enables use of and/or implements global memory atomics that may be shared between graphics core 3200 and CPUs within an SoC. In at

least one embodiment, graphics SoC interface 3237 can also implement power management controls for graphics processor core 3200 and enable an interface between a clock domain of graphics processor core 3200 and other clock domains within an SoC. In at least one embodiment, SoC interface 3237 enables receipt of command buffers from a command streamer and global thread dispatcher that are configured to provide commands and instructions to each of one or more graphics cores within a graphics processor. In at least one embodiment, commands and instructions can be dispatched to media pipeline 3239, when media operations are to be performed, or a geometry and fixed function pipeline (e.g., geometry and fixed function pipeline 3236, and/or a geometry and fixed function pipeline 3214) when graphics processing operations are to be performed.

[0515] In at least one embodiment, graphics microcontroller 3238 can be configured to perform various scheduling and management tasks for graphics core 3200. In at least one embodiment, graphics microcontroller 3238 can perform graphics and/or compute workload scheduling on various graphics parallel engines within execution unit (EU) arrays 3202A-3202F, 3204A-3204F within sub-cores 3201A-3201F. In at least one embodiment, host software executing on a CPU core of an SoC including graphics core 3200 can submit workloads to one of multiple graphic processor paths, which invokes a scheduling operation on an appropriate graphics engine. In at least one embodiment, scheduling operations include determining which workload to run next, submitting a workload to a command streamer, pre-empting existing workloads running on an engine, monitoring progress of a workload, and notifying host software when a workload is complete. In at least one embodiment, graphics microcontroller 3238 can also facilitate low-power or idle states for graphics core 3200, providing graphics core 3200 with an ability to save and restore registers within graphics core 3200 across low-power state transitions independently from an operating system and/or graphics driver software on a system.

[0516] In at least one embodiment, graphics core 3200 may have greater than or fewer than illustrated sub-cores 3201A-3201F, up to N modular sub-cores. For each set of N sub-cores, in at least one embodiment, graphics core 3200 can also include shared function logic 3210, shared and/or cache memory 3212, geometry/fixed function pipeline 3214, as well as additional fixed function logic 3216 to accelerate various graphics and compute processing operations. In at least one embodiment, shared function logic 3210 can include logic units (e.g., sampler, math, and/or

inter-thread communication logic) that can be shared by each N sub-cores within graphics core 3200. In at least one embodiment, shared and/or cache memory 3212 can be a last-level cache for N sub-cores 3201A-3201F within graphics core 3200 and can also serve as shared memory that is accessible by multiple sub-cores. In at least one embodiment, geometry/fixed function pipeline 3214 can be included instead of geometry/fixed function pipeline 3236 within fixed function block 3230 and can include similar logic units.

[0517] In at least one embodiment, graphics core 3200 includes additional fixed function logic 3216 that can include various fixed function acceleration logic for use by graphics core 3200. In at least one embodiment, additional fixed function logic 3216 includes an additional geometry pipeline for use in position-only shading. In position-only shading, at least two geometry pipelines exist, whereas in a full geometry pipeline within geometry and fixed function pipelines 3214, 3236, and a cull pipeline, which is an additional geometry pipeline that may be included within additional fixed function logic 3216. In at least one embodiment, a cull pipeline is a trimmed down version of a full geometry pipeline. In at least one embodiment, a full pipeline and a cull pipeline can execute different instances of an application, each instance having a separate context. In at least one embodiment, position only shading can hide long cull runs of discarded triangles, enabling shading to be completed earlier in some instances. For example, in at least one embodiment, cull pipeline logic within additional fixed function logic 3216 can execute position shaders in parallel with a main application and generally generates critical results faster than a full pipeline, as a cull pipeline fetches and shades position attributes of vertices, without performing rasterization and rendering of pixels to a frame buffer. In at least one embodiment, a cull pipeline can use generated critical results to compute visibility information for all triangles without regard to whether those triangles are culled. In at least one embodiment, a full pipeline (which in this 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.

[0518] In at least one embodiment, additional fixed function logic 3216 can also include machine-learning acceleration logic, such as fixed function matrix multiplication logic, for implementations including optimizations for machine learning training or inferencing.

[0519] In at least one embodiment, within each graphics sub-core 3201A-3201F includes a set of execution resources that may be used to perform graphics, media, and compute operations in response to requests by graphics pipeline, media pipeline, or shader programs. In at least one embodiment, graphics sub-cores 3201A-3201F include multiple EU arrays 3202A-3202F, 3204A-3204F, thread dispatch and inter-thread communication (TD/IC) logic 3203A-3203F, a 3D (e.g., texture) sampler 3205A-3205F, a media sampler 3206A-3206F, a shader processor 3207A-3207F, and shared local memory (SLM) 3208A-3208F. In at least one embodiment, EU arrays 3202A-3202F, 3204A-3204F each include multiple execution units, which are general-purpose graphics processing units capable of performing floating-point and integer/fixed-point logic operations in service of a graphics, media, or compute operation, including graphics, media, or compute shader programs. In at least one embodiment, TD/IC logic 3203A-3203F performs local thread dispatch and thread control operations for execution units within a sub-core and facilitates communication between threads executing on execution units of a sub-core. In at least one embodiment, 3D samplers 3205A-3205F can read texture or other 3D graphics related data into memory. In at least one embodiment, 3D samplers can read texture data differently based on a configured sample state and texture format associated with a given texture. In at least one embodiment, media samplers 3206A-3206F can perform similar read operations based on a type and format associated with media data. In at least one embodiment, each graphics sub-core 3201A-3201F can alternately include a unified 3D and media sampler. In at least one embodiment, threads executing on execution units within each of sub-cores 3201A-3201F can make use of shared local memory 3208A-3208F within each sub-core, to enable threads executing within a thread group to execute using a common pool of on-chip memory.

[0520] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, portions or all of logic 915 may be incorporated into graphics processor 3200. 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 3238, geometry and fixed function pipeline 3214 and 3236, or other logic in FIG. 32. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory

and/or registers (shown or not shown) that configure ALUs of graphics processor 3200 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0521] In at least one embodiment, at least one component shown or described with respect to FIG. 32 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, graphics processor 3200 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0522] FIGS. 33A-33B illustrate thread execution logic 3300 including an array of processing elements of a graphics processor core according to at least one embodiment. FIG. 33A illustrates at least one embodiment, in which thread execution logic 3300 is used. FIG. 33B illustrates exemplary internal details of a graphics execution unit 3308, according to at least one embodiment.

[0523] As illustrated in FIG. 33A, in at least one embodiment, thread execution logic 3300 includes a shader processor 3302, a thread dispatcher 3304, an instruction cache 3306, a scalable execution unit array including a plurality of execution units 3307A-3307N and 3308A-3308N, a sampler 3310, a data cache 3312, and a data port 3314. In at least one embodiment, a scalable execution unit array can dynamically scale by enabling or disabling one or more execution units (e.g., any of execution unit 3308A-N or 3307A-N) based on computational requirements of a workload, for example. In at least one embodiment, scalable execution units are interconnected via an interconnect fabric that links to each execution unit. In at least one embodiment, thread execution logic 3300 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 3306, data port 3314, sampler 3310, and execution units 3307 or 3308. In at least one embodiment, each execution unit (e.g., 3307A) is a stand-alone programmable general-purpose computational unit that is capable of executing multiple simultaneous hardware threads while processing multiple data elements in parallel for each thread. In at least one embodiment, array of execution units 3307 and/or 3308 is scalable to include any number individual execution units.

[0524] In at least one embodiment, execution units 3307 and/or 3308 are primarily used to execute shader programs. In at least one embodiment, shader processor 3302 can process various shader programs and dispatch execution threads associated with shader programs via a thread dispatcher 3304. In at least one embodiment, thread dispatcher 3304 includes logic to arbitrate thread initiation requests from graphics and media pipelines and instantiate requested threads on one or more execution units in execution units 3307 and/or 3308. For example, in at least one embodiment, a geometry pipeline can dispatch vertex, tessellation, or geometry shaders to thread execution logic for processing. In at least one embodiment, thread dispatcher 3304 can also process runtime thread spawning requests from executing shader programs.

[0525] In at least one embodiment, execution units 3307 and/or 3308 support an instruction set that includes native support for many standard 3D graphics shader instructions, such that shader programs from graphics libraries (e.g., Direct 3D and OpenGL) are executed with a minimal translation. In at least one embodiment, execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, and/or vertex shaders), pixel processing (e.g., pixel shaders, fragment shaders) and general-purpose processing (e.g., compute and media shaders). In at least one embodiment, each of execution units 3307 and/or 3308, which include one or more arithmetic logic units (ALUs), is capable of multi-issue single instruction multiple data (SIMD) execution and multi-threaded operation enables an efficient execution environment despite higher latency memory accesses. In at least one embodiment, each hardware thread within each execution unit has a dedicated high-bandwidth register file and associated independent thread-state. In at least one embodiment, execution is multi-issue per clock to pipelines capable of integer, single and double precision floating point operations, SIMD branch capability, logical operations, transcendental operations, and other miscellaneous operations. In at least one embodiment, while waiting for data from memory or one of shared functions, dependency logic within execution units 3307 and/or 3308 causes a waiting thread to sleep until requested data has been returned. In at least one embodiment, while an awaiting thread is sleeping, hardware resources may be devoted to processing other threads. For example, in at least one embodiment, during a delay associated with a vertex shader operation, an execution unit can perform operations for a pixel shader, fragment shader, or another type of shader program, including a different vertex shader.

[0526] In at least one embodiment, each execution unit in execution units 3307 and/or 3308 operates on arrays of data elements. In at least one embodiment, a number of data elements is an “execution size,” or number of channels for an instruction. In at least one embodiment, 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 3307 and/or 3308 support integer and floating-point data types.

[0527] 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.

[0528] In at least one embodiment, one or more execution units can be combined into a fused execution unit 3309A-3309N having thread control logic (3311A-3311N) that is common to fused EUs such as execution unit 3307A fused with execution unit 3308A into fused execution unit 3309A. In at least one embodiment, multiple EUs can be fused into an EU group. In at least one embodiment, each EU in a fused EU group can be configured to execute a separate SIMD hardware thread, with a number of EUs in a fused EU group possibly varying according to various embodiments. In at least one embodiment, various SIMD widths can be performed per-EU, including but not limited to SIMD8, SIMD16, and SIMD32. In at least one embodiment, each fused graphics execution unit 3309A-3309N includes at least two execution units. For example, in at least one embodiment, fused execution unit 3309A includes a first EU 3307A, second EU 3308A, and thread control logic 3311A that is common to first EU 3307A and second EU 3308A. In at least one embodiment, thread control logic 3311A controls threads executed on

fused graphics execution unit 3309A, allowing each EU within fused execution units 3309A-3309N to execute using a common instruction pointer register.

[0529] In at least one embodiment, one or more internal instruction caches (e.g., 3306) are included in thread execution logic 3300 to cache thread instructions for execution units. In at least one embodiment, one or more data caches (e.g., 3312) are included to cache thread data during thread execution. In at least one embodiment, sampler 3310 is included to provide texture sampling for 3D operations and media sampling for media operations. In at least one embodiment, sampler 3310 includes specialized texture or media sampling functionality to process texture or media data during sampling process before providing sampled data to an execution unit.

[0530] During execution, in at least one embodiment, graphics and media pipelines send thread initiation requests to thread execution logic 3300 via thread spawning and dispatch logic. In at least one embodiment, once a group of geometric objects has been processed and rasterized into pixel data, pixel processor logic (e.g., pixel shader logic, fragment shader logic, etc.) within shader processor 3302 is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In at least one embodiment, a pixel shader or a fragment shader calculates values of various vertex attributes that are to be interpolated across a rasterized object. In at least one embodiment, pixel processor logic within shader processor 3302 then executes an application programming interface (API)-supplied pixel or fragment shader program. In at least one embodiment, to execute a shader program, shader processor 3302 dispatches threads to an execution unit (e.g., 3308A) via thread dispatcher 3304. In at least one embodiment, shader processor 3302 uses texture sampling logic in sampler 3310 to access texture data in texture maps stored in memory. In at least one embodiment, arithmetic operations on texture data and input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

[0531] In at least one embodiment, data port 3314 provides a memory access mechanism for thread execution logic 3300 to output processed data to memory for further processing on a graphics processor output pipeline. In at least one embodiment, data port 3314 includes or couples to one or more cache memories (e.g., data cache 3312) to cache data for memory access via a data port.

[0532] As illustrated in FIG. 33B, in at least one embodiment, a graphics execution unit 3308 can include an instruction fetch unit 3337, a general register file array (GRF) 3324, an architectural register file array (ARF) 3326, a thread arbiter 3322, a send unit 3330, a branch unit 3332, a set of SIMD floating point units (FPUs) 3334, and a set of dedicated integer SIMD ALUs 3335. In at least one embodiment, GRF 3324 and ARF 3326 includes a set of general register files and architecture register files associated with each simultaneous hardware thread that may be active in graphics execution unit 3308. In at least one embodiment, per thread architectural state is maintained in ARF 3326, while data used during thread execution is stored in GRF 3324. In at least one embodiment, execution state of each thread, including instruction pointers for each thread, can be held in thread-specific registers in ARF 3326.

[0533] In at least one embodiment, graphics execution unit 3308 has an architecture that is a combination of Simultaneous Multi-Threading (SMT) and fine-grained Interleaved Multi-Threading (IMT). In at least one embodiment, architecture has a modular configuration that can be fine-tuned at design time based on a target number of simultaneous threads and number of registers per execution unit, where execution unit resources are divided across logic used to execute multiple simultaneous threads.

[0534] In at least one embodiment, graphics execution unit 3308 can co-issue multiple instructions, which may each be different instructions. In at least one embodiment, thread arbiter 3322 of graphics execution unit thread 3308 can dispatch instructions to one of send unit 3330, branch unit 3332, or SIMD FPU(s) 3334 for execution. In at least one embodiment, each execution thread can access 128 general-purpose registers within GRF 3324, where each register can store 32 bytes, accessible as a SIMD 8-element vector of 32-bit data elements. In at least one embodiment, each execution unit thread has access to 4 kilobytes within GRF 3324, although embodiments are not so limited, and greater or fewer register resources may be provided in other embodiments. In at least one embodiment, up to seven threads can execute simultaneously, although a number of threads per execution unit can also vary according to embodiments. In at least one embodiment, in which seven threads may access 4 kilobytes, GRF 3324 can store a total of 28 kilobytes. In at least one embodiment, flexible addressing modes can permit registers to be addressed together to build effectively wider registers or to represent strided rectangular block data structures.

[0535] In at least one embodiment, memory operations, sampler operations, and other longer-latency system communications are dispatched via “send” instructions that are executed by message passing to send unit 3330. In at least one embodiment, branch instructions are dispatched to branch unit 3332 to facilitate SIMD divergence and eventual convergence.

[0536] In at least one embodiment, graphics execution unit 3308 includes one or more SIMD floating point units (FPU(s)) 3334 to perform floating-point operations. In at least one embodiment, FPU(s) 3334 also support integer computation. In at least one embodiment, FPU(s) 3334 can SIMD execute up to M number of 32-bit floating-point (or integer) operations, or SIMD execute up to 2M 16-bit integer or 16-bit floating-point operations. In at least one embodiment, at least one FPU provides extended math capability to support high-throughput transcendental math functions and double precision 64-bit floating-point. In at least one embodiment, a set of 8-bit integer SIMD ALUs 3335 are also present, and may be specifically optimized to perform operations associated with machine learning computations.

[0537] In at least one embodiment, arrays of multiple instances of graphics execution unit 3308 can be instantiated in a graphics sub-core grouping (e.g., a sub-slice). In at least one embodiment, execution unit 3308 can execute instructions across a plurality of execution channels. In at least one embodiment, each thread executed on graphics execution unit 3308 is executed on a different channel.

[0538] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, portions or all of logic 915 may be incorporated into thread execution logic 3300. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIGS. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs thread of execution logic 3300 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0539] In at least one embodiment, at least one component shown or described with respect to FIGS. 33A-33B is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, execution logic 3300 performs one or more operations

related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0540] FIG. 34 illustrates a parallel processing unit (“PPU”) 3400, according to at least one embodiment. In at least one embodiment, PPU 3400 is configured with machine-readable code that, if executed by PPU 3400, causes PPU 3400 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, PPU 3400 is a multi-threaded processor that is implemented on one or more integrated circuit devices 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, PPU 3400 includes one or more graphics cores 2000. 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 3400. In at least one embodiment, PPU 3400 is a graphics processing unit (“GPU”) configured to implement a graphics rendering pipeline for processing three-dimensional (“3D”) graphics data in order to generate two-dimensional (“2D”) image data for display on a display device such as a liquid crystal display (“LCD”) device. In at least one embodiment, PPU 3400 is utilized to perform computations such as linear algebra operations and machine-learning operations. FIG. 34 illustrates an example parallel processor for illustrative purposes only and should be construed as a non-limiting example of processor architectures contemplated within scope of this disclosure and that any suitable processor may be employed to supplement and/or substitute for same.

[0541] In at least one embodiment, one or more PPUs 3400 are configured to accelerate High Performance Computing (“HPC”), data center, and machine learning applications. In at least one embodiment, PPU 3400 is configured to accelerate deep learning systems and applications including following non-limiting examples: autonomous vehicle platforms, deep learning, high-accuracy speech, image, text recognition systems, intelligent video analytics, molecular simulations, drug discovery, disease diagnosis, weather forecasting, big data analytics, astronomy, molecular dynamics simulation, financial modeling, robotics, factory automation, real-time language translation, online search optimizations, and personalized user recommendations, and more.

[0542] In at least one embodiment, PPU 3400 includes, without limitation, an Input/Output (“I/O”) unit 3406, a front-end unit 3410, a scheduler (sequencer) unit 3412, a work distribution unit 3414, a hub 3416, a crossbar (“XBar”) 3420, one or more general processing clusters (“GPCs”) 3418, and one or more partition units (“memory partition units”) 3422. In at least one embodiment, PPU 3400 is connected to a host processor or other PPUs 3400 via one or more high-speed GPU interconnects (“GPU interconnects”) 3408. In at least one embodiment, PPU 3400 is connected to a host processor or other peripheral devices via a system bus 3402. In at least one embodiment, PPU 3400 is connected to a local memory comprising one or more memory devices (“memory”) 3404. In at least one embodiment, memory devices 3404 include, without limitation, one or more dynamic random access memory (“DRAM”) devices. In at least one embodiment, one or more DRAM devices are configured and/or configurable as high-bandwidth memory (“HBM”) subsystems, with multiple DRAM dies stacked within each device.

[0543] In at least one embodiment, high-speed GPU interconnect 3408 may refer to a wire-based multi-lane communications link that is used by systems to scale and include one or more PPUs 3400 combined with one or more central processing units (“CPUs”), supports cache coherence between PPUs 3400 and CPUs, and CPU mastering. In at least one embodiment, data and/or commands are transmitted by high-speed GPU interconnect 3408 through hub 3416 to/from other units of PPU 3400 such as one or more copy engines, video encoders, video decoders, power management units, and other components which may not be explicitly illustrated in FIG. 34.

[0544] In at least one embodiment, I/O unit 3406 is configured to transmit and receive communications (e.g., commands, data) from a host processor (not illustrated in FIG. 34) over system bus 3402. In at least one embodiment, I/O unit 3406 communicates with host processor directly via system bus 3402 or through one or more intermediate devices such as a memory bridge. In at least one embodiment, I/O unit 3406 may communicate with one or more other processors, such as one or more of PPUs 3400 via system bus 3402. In at least one embodiment, I/O unit 3406 implements a Peripheral Component Interconnect Express (“PCIe”) interface for communications over a PCIe bus. In at least one embodiment, I/O unit 3406 implements interfaces for communicating with external devices.

[0545] In at least one embodiment, I/O unit 3406 decodes packets received via system bus 3402. In at least one embodiment, at least some packets represent commands configured to cause PPU 3400 to perform various operations. In at least one embodiment, I/O unit 3406 transmits decoded commands to various other units of PPU 3400 as specified by commands. In at least one embodiment, commands are transmitted to front-end unit 3410 and/or transmitted to hub 3416 or other units of PPU 3400 such as one or more copy engines, a video encoder, a video decoder, a power management unit, etc. (not explicitly illustrated in FIG. 34). In at least one embodiment, I/O unit 3406 is configured to route communications between and among various logical units of PPU 3400.

[0546] In at least one embodiment, a program executed by host processor encodes a command stream in a buffer that provides workloads to PPU 3400 for processing. In at least one embodiment, a workload comprises instructions and data to be processed by those instructions. In at least one embodiment, a buffer is a region in a memory that is accessible (e.g., read/write) by both a host processor and PPU 3400 — a host interface unit may be configured to access that buffer in a system memory connected to system bus 3402 via memory requests transmitted over system bus 3402 by I/O unit 3406. 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 3400 such that front-end unit 3410 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 3400.

[0547] In at least one embodiment, front-end unit 3410 is coupled to scheduler unit 3412 (which may be referred to as a sequencer unit, a thread sequencer, and/or an asynchronous compute engine) that configures various GPCs 3418 to process tasks defined by one or more command streams. In at least one embodiment, scheduler unit 3412 is configured to track state information related to various tasks managed by scheduler unit 3412 where state information may indicate which of GPCs 3418 a task is assigned to, whether task is active or inactive, a priority level associated with task, and so forth. In at least one embodiment, scheduler unit 3412 manages execution of a plurality of tasks on one or more of GPCs 3418.

[0548] In at least one embodiment, scheduler unit 3412 is coupled to work distribution unit 3414 that is configured to dispatch tasks for execution on GPCs 3418. In at least one

embodiment, work distribution unit 3414 tracks a number of scheduled tasks received from scheduler unit 3412 and work distribution unit 3414 manages a pending task pool and an active task pool for each of GPCs 3418. In at least one embodiment, pending task pool comprises a number of slots (e.g., 32 slots) that contain tasks assigned to be processed by a particular GPC 3418; an active task pool may comprise a number of slots (e.g., 4 slots) for tasks that are actively being processed by GPCs 3418 such that as one of GPCs 3418 completes execution of a task, that task is evicted from that active task pool for GPC 3418 and another task from a pending task pool is selected and scheduled for execution on GPC 3418. In at least one embodiment, if an active task is idle on GPC 3418, such as while waiting for a data dependency to be resolved, then that active task is evicted from GPC 3418 and returned to that pending task pool while another task in that pending task pool is selected and scheduled for execution on GPC 3418.

[0549] In at least one embodiment, work distribution unit 3414 communicates with one or more GPCs 3418 via XBar 3420. In at least one embodiment, XBar 3420 is an interconnect network that couples many of units of PPU 3400 to other units of PPU 3400 and can be configured to couple work distribution unit 3414 to a particular GPC 3418. In at least one embodiment, one or more other units of PPU 3400 may also be connected to XBar 3420 via hub 3416.

[0550] In at least one embodiment, tasks are managed by scheduler unit 3412 and dispatched to one of GPCs 3418 by work distribution unit 3414. In at least one embodiment, GPC 3418 is configured to process task and generate results. In at least one embodiment, results may be consumed by other tasks within GPC 3418, routed to a different GPC 3418 via XBar 3420, or stored in memory 3404. In at least one embodiment, results can be written to memory 3404 via partition units 3422, which implement a memory interface for reading and writing data to/from memory 3404. In at least one embodiment, results can be transmitted to another PPU or CPU via high-speed GPU interconnect 3408. In at least one embodiment, PPU 3400 includes, without limitation, a number U of partition units 3422 that is equal to a number of separate and distinct memory devices 3404 coupled to PPU 3400, as described in more detail herein in conjunction with FIG. 36.

[0551] In at least one embodiment, a host processor executes a driver kernel that implements an application programming interface (“API”) that enables one or more applications executing

on a host processor to schedule operations for execution on PPU 3400. In at least one embodiment, multiple compute applications are simultaneously executed by PPU 3400 and PPU 3400 provides isolation, quality of service (“QoS”), and independent address spaces for multiple compute applications. In at least one embodiment, an application generates instructions (e.g., in form of API calls) that cause a driver kernel to generate one or more tasks for execution by PPU 3400 and that driver kernel outputs tasks to one or more streams being processed by PPU 3400. In at least one embodiment, each task comprises one or more groups of related threads, which may be referred to as a warp, wavefront, and/or wave. In at least one embodiment, a warp, wavefront, and/or wave comprises a plurality of related threads (e.g., 32 threads) that can be executed in parallel. In at least one embodiment, cooperating threads can refer to a plurality of threads including instructions to perform task and that exchange data through shared memory. In at least one embodiment, threads and cooperating threads are described in more detail in conjunction with FIG. 36.

[0552] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to PPU 3400. In at least one embodiment, deep learning application processor is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by PPU 3400. In at least one embodiment, PPU 3400 may be used to perform one or more neural network use cases described herein.

[0553] In at least one embodiment, at least one component shown or described with respect to FIG. 34 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, PPU 3400 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0554] FIG. 35 illustrates a general processing cluster (“GPC”) 3500, according to at least one embodiment. In at least one embodiment, GPC 3500 is GPC 3418 of FIG. 34. In at least one

embodiment, each GPC 3500 includes, without limitation, a number of hardware units for processing tasks and each GPC 3500 includes, without limitation, a pipeline manager 3502, a pre-raster operations unit (“preROP”) 3504, a raster engine 3508, a work distribution crossbar (“WDX”) 3516, a memory management unit (“MMU”) 3518, one or more Data Processing Clusters (“DPCs”) 3506, and any suitable combination of parts.

[0555] In at least one embodiment, operation of GPC 3500 is controlled by pipeline manager 3502. In at least one embodiment, pipeline manager 3502 manages configuration of one or more DPCs 3506 for processing tasks allocated to GPC 3500. In at least one embodiment, pipeline manager 3502 configures at least one of one or more DPCs 3506 to implement at least a portion of a graphics rendering pipeline. In at least one embodiment, DPC 3506 is configured to execute a vertex shader program on a programmable streaming multi-processor (“SM”) 3514. In at least one embodiment, pipeline manager 3502 is configured to route packets received from a work distribution unit to appropriate logical units within GPC 3500, in at least one embodiment, and some packets may be routed to fixed function hardware units in preROP 3504 and/or raster engine 3508 while other packets may be routed to DPCs 3506 for processing by a primitive engine 3512 or SM 3514. In at least one embodiment, pipeline manager 3502 configures at least one of DPCs 3506 to implement a neural network model and/or a computing pipeline.

[0556] In at least one embodiment, preROP unit 3504 is configured, in at least one embodiment, to route data generated by raster engine 3508 and DPCs 3506 to a Raster Operations (“ROP”) unit in partition unit 3422, described in more detail above in conjunction with FIG. 34. In at least one embodiment, preROP unit 3504 is configured to perform optimizations for color blending, organize pixel data, perform address translations, and more. In at least one embodiment, raster engine 3508 includes, without limitation, a number of fixed function hardware units configured to perform various raster operations, in at least one embodiment, and raster engine 3508 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 3508 comprises fragments to be processed by any suitable entity, such as by a fragment shader implemented within DPC 3506.

[0557] In at least one embodiment, each DPC 3506 included in GPC 3500 comprises, without limitation, an M-Pipe Controller (“MPC”) 3510; primitive engine 3512; one or more SMs 3514; and any suitable combination thereof. In at least one embodiment, MPC 3510 controls operation of DPC 3506, routing packets received from pipeline manager 3502 to appropriate units in DPC 3506. In at least one embodiment, packets associated with a vertex are routed to primitive engine 3512, which is configured to fetch vertex attributes associated with a vertex from memory; in contrast, packets associated with a shader program may be transmitted to SM 3514.

[0558] In at least one embodiment, SM 3514 comprises, without limitation, a programmable streaming processor that is configured to process tasks represented by a number of threads. In at least one embodiment, SM 3514 is multi-threaded and configured to execute a plurality of threads (e.g., 32 threads) from a particular group of threads concurrently and implements a Single-Instruction, Multiple-Data (“SIMD”) architecture where each thread in a group of threads (e.g., a warp, wavefront, wave) is configured to process a different set of data based on same set of instructions. In at least one embodiment, all threads in group of threads execute a common set of instructions. In at least one embodiment, SM 3514 implements a Single-Instruction, Multiple Thread (“SIMT”) architecture wherein each thread in a group of threads is configured to process a different set of data based on that common set of instructions, but where individual threads in a group of threads are allowed to diverge during execution. In at least one embodiment, a program counter, call stack, and execution state is maintained for each warp (which may be referred to as wavefronts and/or waves), 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 3514 is described in more detail herein.

[0559] In at least one embodiment, MMU 3518 provides an interface between GPC 3500 and a memory partition unit (e.g., partition unit 3422 of FIG. 34) and MMU 3518 provides translation of virtual addresses into physical addresses, memory protection, and arbitration of memory requests. In at least one embodiment, MMU 3518 provides one or more translation lookaside buffers (“TLBs”) for performing translation of virtual addresses into physical addresses in memory.

[0560] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to GPC 3500. In at least one embodiment, GPC 3500 is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by GPC 3500. In at least one embodiment, GPC 3500 may be used to perform one or more neural network use cases described herein.

[0561] In at least one embodiment, at least one component shown or described with respect to FIG. 35 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, GPC 3500 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0562] FIG. 36 illustrates a memory partition unit 3600 of a parallel processing unit (“PPU”), in accordance with at least one embodiment. In at least one embodiment, memory partition unit 3600 includes, without limitation, a Raster Operations (“ROP”) unit 3602, a level two (“L2”) cache 3604, a memory interface 3606, and any suitable combination thereof. In at least one embodiment, memory interface 3606 is coupled to memory. In at least one embodiment, memory interface 3606 may implement 32, 64, 128, 1024-bit data buses, or like, for high-speed data transfer. In at least one embodiment, PPU incorporates U memory interfaces 3606 where U is a positive integer, with one memory interface 3606 per pair of partition units 3600, where each

pair of partition units 3600 is connected to a corresponding memory device. For example, in at least one embodiment, PPU may be connected to up to Y memory devices, such as high bandwidth memory stacks or graphics double-data-rate, version 5, synchronous dynamic random access memory (“GDDR5 SDRAM”).

[0563] In at least one embodiment, memory interface 3606 implements a high bandwidth memory second generation (“HBM2”) memory interface and Y equals half of U. In at least one embodiment, HBM2 memory stacks are located on a physical package with a PPU, providing substantial power and area savings compared with conventional GDDR5 SDRAM systems. In at least one embodiment, each HBM2 stack includes, without limitation, four memory dies with Y=4, with each HBM2 stack including two 128-bit channels per die for a total of 8 channels and a data bus width of 1024 bits. In at least one embodiment, that memory supports Single-Error Correcting Double-Error Detecting (“SECCDED”) Error Correction Code (“ECC”) to protect data. In at least one embodiment, ECC can provide higher reliability for compute applications that are sensitive to data corruption.

[0564] In at least one embodiment, PPU implements a multi-level memory hierarchy. In at least one embodiment, memory partition unit 3600 supports a unified memory to provide a single unified virtual address space for central processing unit (“CPU”) and PPU memory, enabling data sharing between virtual memory systems. In at least one embodiment frequency of accesses by a PPU to a memory located on other processors is traced to ensure that memory pages are moved to physical memory of PPU that is accessing pages more frequently. In at least one embodiment, high-speed GPU interconnect 3408 supports address translation services allowing PPU to directly access a CPU’s page tables and providing full access to CPU memory by a PPU.

[0565] In at least one embodiment, copy engines transfer data between multiple PPUs or between PPUs and CPUs. In at least one embodiment, copy engines can generate page faults for addresses that are not mapped into page tables and memory partition unit 3600 then services page faults, mapping addresses into page table, after which copy engine performs a transfer. In at least one embodiment, memory is pinned (i.e., non-pageable) for multiple copy engine operations between multiple processors, substantially reducing available memory. In at least one embodiment, with hardware page faulting, addresses can be passed to copy engines without regard as to whether memory pages are resident, and a copy process is transparent.

[0566] Data from memory 3404 of FIG. 34 or other system memory is fetched by memory partition unit 3600 and stored in L2 cache 3604, which is located on-chip and is shared between various GPCs, in accordance with at least one embodiment. Each memory partition unit 3600, in at least one embodiment, includes, without limitation, at least a portion of L2 cache associated with a corresponding memory device. In at least one embodiment, lower level caches are implemented in various units within GPCs. In at least one embodiment, each of SMs 3514 in FIG. 35 may implement a Level 1 (“L1”) cache wherein that L1 cache is private memory that is dedicated to a particular SM 3514 and data from L2 cache 3604 is fetched and stored in each L1 cache for processing in functional units of SMs 3514. In at least one embodiment, L2 cache 3604 is coupled to memory interface 3606 and XBar 3420 shown in FIG. 34.

[0567] ROP unit 3602 performs graphics raster operations related to pixel color, such as color compression, pixel blending, and more, in at least one embodiment. ROP unit 3602, in at least one embodiment, implements depth testing in conjunction with raster engine 3508, receiving a depth for a sample location associated with a pixel fragment from a culling engine of raster engine 3508. In at least one embodiment, depth is tested against a corresponding depth in a depth buffer for a sample location associated with a fragment. In at least one embodiment, if that fragment passes that depth test for that sample location, then ROP unit 3602 updates depth buffer and transmits a result of that depth test to raster engine 3508. It will be appreciated that a number of partition units 3600 may be different than a number of GPCs and, therefore, each ROP unit 3602 can, in at least one embodiment, be coupled to each GPC. In at least one embodiment, ROP unit 3602 tracks packets received from different GPCs and determines whether a result generated by ROP unit 3602 is to be routed to through XBar 3420.

[0568] FIG. 37 illustrates a streaming multi-processor (“SM”) 3700, according to at least one embodiment. In at least one embodiment, SM 3700 is SM of FIG. 35. In at least one embodiment, SM 3700 includes, without limitation, an instruction cache 3702, one or more scheduler units 3704 (which may be referred to as sequencer units), a register file 3708, one or more processing cores (“cores”) 3710, one or more special function units (“SFUs”) 3712, one or more load/store units (“LSUs”) 3714, an interconnect network 3716, a shared memory/level one (“L1”) cache 3718, and/or any suitable combination thereof. In at least one embodiment, LSUs

3714 perform load of store operations corresponding to loading/storing data (e.g., instructions) to perform an operation (e.g., perform an API, an API call).

[0569] 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 3700 (which may be referred to as CUs and/or slices). In at least one embodiment, scheduler unit 3704 (which may be referred to as a sequencer and/or asynchronous compute engine) receives tasks from a work distribution unit and manages instruction scheduling for one or more thread blocks assigned to SM 3700. In at least one embodiment, scheduler unit 3704 schedules thread blocks for execution as warps (which may be referred to as wavefronts and/or waves) of parallel threads, wherein each thread block is allocated at least one warp. In at least one embodiment, each warp executes threads. In at least one embodiment, scheduler unit 3704 manages a plurality of different thread blocks, allocating warps to different thread blocks and then dispatching instructions from plurality of different cooperative groups to various functional units (e.g., processing cores 3710, SFUs 3712, and LSUs 3714) during each clock cycle.

[0570] In at least one embodiment, Cooperative Groups (which may also be referred to as wavefronts and/or waves) may refer to a programming model for organizing groups of communicating threads that allows developers to express granularity at which threads are communicating, enabling expression of richer, more efficient parallel decompositions. In at least one embodiment, cooperative launch APIs support synchronization amongst thread blocks for execution of parallel algorithms. In at least one embodiment, applications of conventional programming models provide a single, simple construct for synchronizing cooperating threads: a barrier across all threads of a thread block (e.g., `syncthreads( )` function). However, in at least one embodiment, programmers may define groups of threads at smaller than thread block granularities and synchronize within defined groups to enable greater performance, design flexibility, and software reuse in form of collective group-wide function interfaces. In at least one embodiment, Cooperative Groups enables programmers to define groups of threads explicitly at sub-block (i.e., as small as a single thread) and multi-block granularities, and to perform collective operations such as synchronization on threads in a cooperative group. In at

least one embodiment, that programming model supports clean composition across software boundaries, so that libraries and utility functions can synchronize safely within their local context without having to make assumptions about convergence. In at least one embodiment, Cooperative Groups primitives enable new patterns of cooperative parallelism, including, without limitation, producer-consumer parallelism, opportunistic parallelism, and global synchronization across an entire grid of thread blocks.

[0571] In at least one embodiment, a dispatch unit 3706 is configured to transmit instructions to one or more functional units and scheduler unit 3704 and includes, without limitation, two dispatch units 3706 that enable two different instructions from a common warp to be dispatched during each clock cycle. In at least one embodiment, each scheduler unit 3704 includes a single dispatch unit 3706 or additional dispatch units 3706.

[0572] In at least one embodiment, each SM 3700 (which may be referred to as a CU and/or slice), in at least one embodiment, includes, without limitation, register file 3708 that provides a set of registers for functional units of SM 3700. In at least one embodiment, register file 3708 is divided between each functional unit such that each functional unit is allocated a dedicated portion of register file 3708. In at least one embodiment, register file 3708 is divided between different warps being executed by SM 3700 and register file 3708 provides temporary storage for operands connected to data paths of functional units. In at least one embodiment, each SM 3700 comprises, without limitation, a plurality of L processing cores 3710, where L is a positive integer. In at least one embodiment, SM 3700 includes, without limitation, a large number (e.g., 128 or more) of distinct processing cores 3710. In at least one embodiment, each processing core 3710 includes, without limitation, a fully-pipelined, single-precision, double-precision, and/or mixed precision processing unit that includes, without limitation, a floating point arithmetic logic unit and an integer arithmetic logic unit. In at least one embodiment, floating point arithmetic logic units implement IEEE 754-2008 standard for floating point arithmetic. In at least one embodiment, processing cores 3710 include, without limitation, 64 single-precision (32-bit) floating point cores, 64 integer cores, 32 double-precision (64-bit) floating point cores, and 8 tensor cores.

[0573] Tensor cores are configured to perform matrix operations in accordance with at least one embodiment. In at least one embodiment, one or more tensor cores are included in

processing cores 3710. In at least one embodiment, tensor cores are configured to perform deep learning matrix arithmetic, such as convolution operations for neural network training and inferencing. In at least one embodiment, each tensor core operates on a 4x4 matrix and performs a matrix multiply and accumulate operation, $D = A \times B + C$, where A, B, C, and D are 4x4 matrices.

[0574] 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 (which may be referred to as a wavefront and/or wave).

[0575] In at least one embodiment, each SM 3700 comprises, without limitation, M SFUs 3712 that perform special functions (e.g., attribute evaluation, reciprocal square root, and like). In at least one embodiment, SFUs 3712 include, without limitation, a tree traversal unit configured to traverse a hierarchical tree data structure. In at least one embodiment, SFUs 3712 include, without limitation, a texture unit configured to perform texture map filtering operations. In at least one embodiment, texture units are configured to load texture maps (e.g., a 2D array of texels) from memory and sample texture maps to produce sampled texture values for use in shader programs executed by SM 3700. In at least one embodiment, texture maps are stored in shared memory/L1 cache 3718. In at least one embodiment, texture units implement texture operations such as filtering operations using mip-maps (e.g., texture maps of varying levels of detail), in accordance with at least one embodiment. In at least one embodiment, each SM 3700 includes, without limitation, two texture units.

[0576] Each SM 3700 comprises, without limitation, N LSUs 3714 that implement load and store operations between shared memory/L1 cache 3718 and register file 3708, in at least one embodiment. Interconnect network 3716 connects each functional unit to register file 3708 and LSU 3714 to register file 3708 and shared memory/L1 cache 3718 in at least one embodiment. In at least one embodiment, interconnect network 3716 is a crossbar that can be configured to connect any functional units to any registers in register file 3708 and connect LSUs 3714 to register file 3708 and memory locations in shared memory/L1 cache 3718.

[0577] In at least one embodiment, shared memory/L1 cache 3718 is an array of on-chip memory that allows for data storage and communication between SM 3700 and primitive engine and between threads in SM 3700, in at least one embodiment. In at least one embodiment, shared memory/L1 cache 3718 comprises, without limitation, 128 KB of storage capacity and is in a path from SM 3700 to a partition unit. In at least one embodiment, shared memory/L1 cache 3718, in at least one embodiment, is used to cache reads and writes. In at least one embodiment, one or more of shared memory/L1 cache 3718, L2 cache, and memory are backing stores.

[0578] 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 3718 enables shared memory/L1 cache 3718 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 3700 to execute program and perform calculations, shared memory/L1 cache 3718 to communicate between threads, and LSU 3714 to

read and write global memory through shared memory/L1 cache 3718 and memory partition unit. In at least one embodiment, when configured for general purpose parallel computation, SM 3700 writes commands that scheduler unit 3704 can use to launch new work on DPCs.

[0579] 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.

[0580] 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.

[0581] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to SM 3700. In at least one embodiment, SM 3700 is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by SM 3700. In at least one embodiment, SM 3700 may be used to perform one or more neural network use cases described herein.

[0582] In at least one embodiment, at least one component shown or described with respect to FIG. 36 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, memory partition unit 3600 stores data used to performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0583] Embodiments are disclosed related a virtualized computing platform for advanced computing, such as image inferencing and image processing in medical applications. Without limitation, embodiments may include radiography, magnetic resonance imaging (MRI), nuclear medicine, ultrasound, sonography, elastography, photoacoustic imaging, tomography, echocardiography, functional near-infrared spectroscopy, and magnetic particle imaging, or a combination thereof. In at least one embodiment, a virtualized computing platform and associated processes described herein may additionally or alternatively be used, without limitation, in forensic science analysis, sub-surface detection and imaging (e.g., oil exploration, archaeology, paleontology, etc.), topography, oceanography, geology, osteology, meteorology, intelligent area or object tracking and monitoring, sensor data processing (e.g., RADAR, SONAR, LIDAR, etc.), and/or genomics and gene sequencing.

[0584] With reference to FIG. 38, FIG. 38 is an example data flow diagram for a process 3800 of generating and deploying an image processing and inferencing pipeline, in accordance with at least one embodiment. In at least one embodiment, process 3800 may be deployed for use with imaging devices, processing devices, genomics devices, gene sequencing devices, radiology devices, and/or other device types at one or more facilities 3802, such as medical facilities, hospitals, healthcare institutes, clinics, research or diagnostic labs, etc. In at least one embodiment, process 3800 may be deployed to perform genomics analysis and inferencing on sequencing data. Examples of genomic analyses that may be performed using systems and processes described herein include, without limitation, variant calling, mutation detection, and gene expression quantification.

[0585] In at least one embodiment, process 3800 may be executed within a training system 3804 and/or a deployment system 3806. In at least one embodiment, training system 3804 may be used to perform training, deployment, and implementation of machine learning models (e.g., neural networks, object detection algorithms, computer vision algorithms, etc.) for use in deployment system 3806. In at least one embodiment, deployment system 3806 may be configured to offload processing and compute resources among a distributed computing environment to reduce infrastructure requirements at facility 3802. In at least one embodiment, deployment system 3806 may provide a streamlined platform for selecting, customizing, and implementing virtual instruments for use with imaging devices (e.g., MRI, CT Scan, X-Ray,

Ultrasound, etc.) or sequencing devices at facility 3802. In at least one embodiment, virtual instruments may include software-defined applications for performing one or more processing operations with respect to imaging data generated by imaging devices, sequencing devices, radiology devices, and/or other device types. In at least one embodiment, one or more applications in a pipeline may use or call upon services (e.g., inference, visualization, compute, AI, etc.) of deployment system 3806 during execution of applications.

[0586] 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 3802 using data 3808 (such as imaging data) generated at facility 3802 (and stored on one or more picture archiving and communication system (PACS) servers at facility 3802), may be trained using imaging or sequencing data 3808 from another facility or facilities (e.g., a different hospital, lab, clinic, etc.), or a combination thereof. In at least one embodiment, training system 3804 may be used to provide applications, services, and/or other resources for generating working, deployable machine learning models for deployment system 3806.

[0587] In at least one embodiment, a model registry 3824 may be backed by object storage that may support versioning and object metadata. In at least one embodiment, object storage may be accessible through, for example, a cloud storage (e.g., a cloud 3926 of FIG. 39) compatible application programming interface (API) from within a cloud platform. In at least one embodiment, machine learning models within model registry 3824 may be uploaded, listed, 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.

[0588] In at least one embodiment, a training pipeline 3904 (FIG. 39) may include a scenario where facility 3802 is training their own machine learning model, or has an existing machine learning model that needs to be optimized or updated. In at least one embodiment, imaging data 3808 generated by imaging device(s), sequencing devices, and/or other device types may be received. In at least one embodiment, once imaging data 3808 is received, AI-assisted annotation 3810 may be used to aid in generating annotations corresponding to imaging data

3808 to be used as ground truth data for a machine learning model. In at least one embodiment, AI-assisted annotation 3810 may include one or more machine learning models (e.g., convolutional neural networks (CNNs)) that may be trained to generate annotations corresponding to certain types of imaging data 3808 (e.g., from certain devices) and/or certain types of anomalies in imaging data 3808. In at least one embodiment, AI-assisted annotations 3810 may then be used directly, or may be adjusted or fine-tuned using an annotation tool (e.g., by a researcher, a clinician, a doctor, a scientist, etc.), to generate ground truth data. In at least one embodiment, in some examples, labeled clinic data 3812 (e.g., annotations provided by a clinician, doctor, scientist, technician, etc.) may be used as ground truth data for training a machine learning model. In at least one embodiment, AI-assisted annotations 3810, labeled clinic data 3812, or a combination thereof may be used as ground truth data for training a machine learning model. In at least one embodiment, a trained machine learning model may be referred to as an output model 3816, and may be used by deployment system 3806, as described herein.

[0589] In at least one embodiment, training pipeline 3904 (FIG. 39) may include a scenario where facility 3802 needs a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3806, but facility 3802 may not currently have such a machine learning model (or may not have a model that is optimized, efficient, or effective for such purposes). In at least one embodiment, an existing machine learning model may be selected from model registry 3824. In at least one embodiment, model registry 3824 may include machine learning models trained to perform a variety of different inference tasks on imaging data. In at least one embodiment, machine learning models in model registry 3824 may have been trained on imaging data from different facilities than facility 3802 (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 3824. In at least one embodiment, a machine learning model may then be retrained, or

updated, at any number of other facilities, and a retrained or updated model may be made available in model registry 3824. In at least one embodiment, a machine learning model may then be selected from model registry 3824 – and referred to as output model 3816 – and may be used in deployment system 3806 to perform one or more processing tasks for one or more applications of a deployment system.

[0590] In at least one embodiment, training pipeline 3904 (FIG. 39) may be used in a scenario that includes facility 3802 requiring a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3806, but facility 3802 may not currently have such a machine learning model (or may not have a model that is optimized, efficient, or effective for such purposes). In at least one embodiment, a machine learning model selected from model registry 3824 might not be fine-tuned or optimized for imaging data 3808 generated at facility 3802 because of differences in populations, genetic variations, robustness of training data used to train a machine learning model, diversity in anomalies of training data, and/or other issues with training data. In at least one embodiment, AI-assisted annotation 3810 may be used to aid in generating annotations corresponding to imaging data 3808 to be used as ground truth data for retraining or updating a machine learning model. In at least one embodiment, labeled clinic data 3812 (e.g., annotations provided by a clinician, doctor, scientist, etc.) may be used as ground truth data for training a machine learning model. In at least one embodiment, retraining or updating a machine learning model may be referred to as model training 3814. In at least one embodiment, model training 3814 – e.g., AI-assisted annotations 3810, labeled clinic data 3812, or a combination thereof – may be used as ground truth data for retraining or updating a machine learning model.

[0591] In at least one embodiment, deployment system 3806 may include software 3818, services 3820, hardware 3822, and/or other components, features, and functionality. In at least one embodiment, deployment system 3806 may include a software “stack,” such that software 3818 may be built on top of services 3820 and may use services 3820 to perform some or all of processing tasks, and services 3820 and software 3818 may be built on top of hardware 3822 and use hardware 3822 to execute processing, storage, and/or other compute tasks of deployment system 3806.

[0592] In at least one embodiment, software 3818 may include any number of different containers, where each container may execute an instantiation of an application. In at least one embodiment, each application may perform one or more processing tasks in an advanced processing and inferencing pipeline (e.g., inferencing, object detection, feature detection, segmentation, image enhancement, calibration, etc.). In at least one embodiment, for each type of imaging device (e.g., CT, MRI, X-Ray, ultrasound, sonography, echocardiography, etc.), sequencing device, radiology device, genomics device, etc., there may be any number of containers that may perform a data processing task with respect to imaging data 3808 (or other data types, such as those described herein) generated by a device. In at least one embodiment, an advanced processing and inferencing pipeline may be defined based on selections of different containers that are desired or required for processing imaging data 3808, in addition to containers that receive and configure imaging data for use by each container and/or for use by facility 3802 after processing through a pipeline (e.g., to convert outputs back to a usable data type, such as digital imaging and communications in medicine (DICOM) data, radiology information system (RIS) data, clinical information system (CIS) data, remote procedure call (RPC) data, data substantially compliant with a representation state transfer (REST) interface, data substantially compliant with a file-based interface, and/or raw data, for storage and display at facility 3802). In at least one embodiment, a combination of containers within software 3818 (e.g., that make up a pipeline) may be referred to as a virtual instrument (as described in more detail herein), and a virtual instrument may leverage services 3820 and hardware 3822 to execute some or all processing tasks of applications instantiated in containers.

[0593] In at least one embodiment, a data processing pipeline may receive input data (e.g., imaging data 3808) in a DICOM, RIS, CIS, REST compliant, RPC, raw, and/or other format in response to an inference request (e.g., a request from a user of deployment system 3806, such as a clinician, a doctor, a radiologist, etc.). In at least one embodiment, input data may be representative of one or more images, video, and/or other data representations generated by one or more imaging devices, sequencing devices, radiology devices, genomics devices, and/or other device types. 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 3816 of training system 3804.

[0594] In at least one embodiment, tasks of data processing pipeline may be encapsulated in a container(s) that each represent a discrete, fully functional instantiation of an application and virtualized computing environment that is able to reference machine learning models. In at least one embodiment, containers or applications may be published into a private (e.g., limited access) area of a container registry (described in more detail herein), and trained or deployed models may be stored in model registry 3824 and associated with one or more applications. In at least one embodiment, images of applications (e.g., container images) may be available in a container registry, and once selected by a user from a container registry for deployment in a pipeline, an image may be used to generate a container for an instantiation of an application for use by a user's system.

[0595] In at least one embodiment, developers (e.g., software developers, clinicians, doctors, etc.) may develop, publish, and store applications (e.g., as containers) for performing image processing and/or inferencing on supplied data. In at least one embodiment, development, publishing, and/or storing may be performed using a software development kit (SDK) associated with a system (e.g., to ensure that an application and/or container developed is compliant with or compatible with a system). In at least one embodiment, an application that is developed may be tested locally (e.g., at a first facility, on data from a first facility) with an SDK which may support at least some of services 3820 as a system (e.g., system 3900 of FIG. 39). In at least one embodiment, because DICOM objects may contain anywhere from one to hundreds of images or other data types, and due to a variation in data, a developer may be responsible for managing (e.g., setting constructs for, building pre-processing into an application, etc.) extraction and preparation of incoming DICOM data. In at least one embodiment, once validated by system 3900 (e.g., for accuracy, safety, patient privacy, etc.), an application may be available in a container registry for selection and/or implementation by a user (e.g., a hospital, clinic, lab, healthcare provider, etc.) to perform one or more processing tasks with respect to data at a facility (e.g., a second facility) of a user.

[0596] In at least one embodiment, developers may then share applications or containers through a network for access and use by users of a system (e.g., system 3900 of FIG. 39). In at least one embodiment, completed and validated applications or containers may be stored in a container registry and associated machine learning models may be stored in model registry 3824. In at least one embodiment, a requesting entity (e.g., a user at a medical facility) – who provides an inference or image processing request – may browse a container registry and/or model registry 3824 for an application, container, dataset, machine learning model, etc., select a desired combination of elements for inclusion in data processing pipeline, and submit an imaging processing request. In at least one embodiment, a request may include input data (and associated patient data, in some examples) that is necessary to perform a request, and/or may include a selection of application(s) and/or machine learning models to be executed in processing a request. In at least one embodiment, a request may then be passed to one or more components of deployment system 3806 (e.g., a cloud) to perform processing of data processing pipeline. In at least one embodiment, processing by deployment system 3806 may include referencing selected elements (e.g., applications, containers, models, etc.) from a container registry and/or model registry 3824. In at least one embodiment, once results are generated by a pipeline, results may be returned to a user for reference (e.g., for viewing in a viewing application suite executing on a local, on-premises workstation or terminal). In at least one embodiment, a radiologist may receive results from a data processing pipeline including any number of application and/or containers, where results may include anomaly detection in X-rays, CT scans, MRIs, etc.

[0597] In at least one embodiment, to aid in processing or execution of applications or containers in pipelines, services 3820 may be leveraged. In at least one embodiment, services 3820 may include compute services, artificial intelligence (AI) services, visualization services, and/or other service types. In at least one embodiment, services 3820 may provide functionality that is common to one or more applications in software 3818, so functionality may be abstracted to a service that may be called upon or leveraged by applications. In at least one embodiment, functionality provided by services 3820 may run dynamically and more efficiently, while also scaling well by allowing applications to process data in parallel (e.g., using a parallel computing platform 3930 (FIG. 39)). In at least one embodiment, rather than each application that shares a same functionality offered by a service 3820 being required to have a respective instance of service 3820, service 3820 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. In at least one embodiment, a data augmentation service may further be included that may provide GPU accelerated data (e.g., DICOM, RIS, CIS, REST compliant, RPC, raw, etc.) extraction, resizing, scaling, and/or other augmentation. In at least one embodiment, a visualization service may be used that may add image rendering effects – such as ray-tracing, rasterization, denoising, sharpening, etc. – to add realism to two-dimensional (2D) and/or three-dimensional (3D) models. In at least one embodiment, virtual instrument services may be included that provide for beam-forming, segmentation, inferencing, imaging, and/or support for other applications within pipelines of virtual instruments.

[0598] In at least one embodiment, where a service 3820 includes an AI service (e.g., an inference service), one or more machine learning models associated with an application for anomaly detection (e.g., tumors, growth abnormalities, scarring, etc.) may be executed by calling upon (e.g., as an API call) an inference service (e.g., an inference server) to execute machine learning model(s), or processing thereof, as part of application execution. In at least one embodiment, where another application includes one or more machine learning models for segmentation tasks, an application may call upon an inference service to execute machine learning models for performing one or more of processing operations associated with segmentation tasks. In at least one embodiment, software 3818 implementing advanced processing and inferencing pipeline that includes segmentation application and anomaly detection application may be streamlined because each application may call upon a same inference service to perform one or more inferencing tasks.

[0599] In at least one embodiment, hardware 3822 may include GPUs, CPUs, graphics cards, an AI/deep learning system (e.g., an AI supercomputer, such as NVIDIA's DGX supercomputer system), a cloud platform, or a combination thereof. In at least one embodiment, different types of hardware 3822 may be used to provide efficient, purpose-built support for software 3818 and services 3820 in deployment system 3806. In at least one embodiment, use of GPU processing may be implemented for processing locally (e.g., at facility 3802), within an AI/deep learning system, in a cloud system, and/or in other processing components of deployment system 3806 to

improve efficiency, accuracy, and efficacy of image processing, image reconstruction, segmentation, MRI exams, stroke or heart attack detection (e.g., in real-time), image quality in rendering, etc. In at least one embodiment, a facility may include imaging devices, genomics devices, sequencing devices, and/or other device types on-premises that may leverage GPUs to generate imaging data representative of a subject's anatomy.

[0600] In at least one embodiment, software 3818 and/or services 3820 may be optimized for GPU processing with respect to deep learning, machine learning, and/or high-performance computing, as non-limiting examples. In at least one embodiment, at least some of computing environment of deployment system 3806 and/or training system 3804 may be executed in a datacenter one or more supercomputers or high performance computing systems, with GPU optimized software (e.g., hardware and software combination of NVIDIA's DGX system). In at least one embodiment, datacenters may be compliant with provisions of HIPAA, such that receipt, processing, and transmission of imaging data and/or other patient data is securely handled with respect to privacy of patient data. In at least one embodiment, hardware 3822 may include any number of GPUs that may be called upon to perform processing of data in parallel, as described herein. In at least one embodiment, cloud platform may further include GPU processing for GPU-optimized execution of deep learning tasks, machine learning tasks, or other computing tasks. In at least one embodiment, cloud platform (e.g., NVIDIA's NGC) may be executed using an AI/deep learning 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.

[0601] In at least one embodiment, at least one component shown or described with respect to FIGS. 37-38 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, streaming multiprocessor 3700 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0602] FIG. 39 is a system diagram for an example system 3900 for generating and deploying an imaging deployment pipeline, in accordance with at least one embodiment. In at least one embodiment, system 3900 may be used to implement process 3800 of FIG. 38 and/or other processes including advanced processing and inferencing pipelines. In at least one embodiment, system 3900 may include training system 3804 and deployment system 3806. In at least one embodiment, training system 3804 and deployment system 3806 may be implemented using software 3818, services 3820, and/or hardware 3822, as described herein.

[0603] In at least one embodiment, system 3900 (e.g., training system 3804 and/or deployment system 3806) may be implemented in a cloud computing environment (e.g., using cloud 3926). In at least one embodiment, system 3900 may be implemented locally with respect to a healthcare services facility, or as a combination of both cloud and local computing resources. In at least one embodiment, in embodiments where cloud computing is implemented, patient data may be separated from, or unprocessed by, by one or more components of system 3900 that would render processing non-compliant with HIPAA and/or other data handling and privacy regulations or laws. In at least one embodiment, access to APIs in cloud 3926 may be restricted to authorized users through enacted security measures or protocols. In at least one embodiment, a security protocol may include web tokens that may be signed by an authentication (e.g., AuthN, AuthZ, Gluecon, etc.) service and may carry appropriate authorization. In at least one embodiment, APIs of virtual instruments (described herein), or other instantiations of system 3900, may be restricted to a set of public IPs that have been vetted or authorized for interaction.

[0604] In at least one embodiment, various components of system 3900 may communicate between and among one another using any of a variety of different network types, including but not limited to local area networks (LANs) and/or wide area networks (WANs) via wired and/or wireless communication protocols. In at least one embodiment, communication between facilities and components of system 3900 (e.g., for transmitting inference requests, for receiving results of inference requests, etc.) may be communicated over a data bus or data busses, wireless data protocols (Wi-Fi), wired data protocols (e.g., Ethernet), etc.

[0605] In at least one embodiment, training system 3804 may execute training pipelines 3904, similar to those described herein with respect to FIG. 38. In at least one embodiment, where one or more machine learning models are to be used in deployment pipelines 3910 by deployment

system 3806, training pipelines 3904 may be used to train or retrain one or more (e.g., pre-trained) models, and/or implement one or more of pre-trained models 3906 (e.g., without a need for retraining or updating). In at least one embodiment, as a result of training pipelines 3904, output model(s) 3816 may be generated. In at least one embodiment, training pipelines 3904 may include any number of processing steps, such as but not limited to imaging data (or other input data) conversion or adaption (e.g., using DICOM adapter 3902A to convert DICOM images to another format suitable for processing by respective machine learning models, such as Neuroimaging Informatics Technology Initiative (NIfTI) format), AI-assisted annotation 3810, labeling or annotating of imaging data 3808 to generate labeled clinic data 3812, model selection from a model registry, model training 3814, training, retraining, or updating models, and/or other processing steps. In at least one embodiment, for different machine learning models used by deployment system 3806, different training pipelines 3904 may be used. In at least one embodiment, training pipeline 3904 similar to a first example described with respect to FIG. 38 may be used for a first machine learning model, training pipeline 3904 similar to a second example described with respect to FIG. 38 may be used for a second machine learning model, and training pipeline 3904 similar to a third example described with respect to FIG. 38 may be used for a third machine learning model. In at least one embodiment, any combination of tasks within training system 3804 may be used depending on what is required for each respective machine learning model. In at least one embodiment, one or more of machine learning models may already be trained and ready for deployment so machine learning models may not undergo any processing by training system 3804, and may be implemented by deployment system 3806.

[0606] In at least one embodiment, output model(s) 3816 and/or pre-trained model(s) 3906 may include any types of machine learning models depending on implementation or embodiment. In at least one embodiment, and without limitation, machine learning models used by system 3900 may include machine learning model(s) using linear regression, logistic regression, decision trees, support vector machines (SVM), Naïve Bayes, k-nearest neighbor (Knn), K means clustering, random forest, dimensionality reduction algorithms, gradient boosting algorithms, neural networks (e.g., auto-encoders, convolutional, recurrent, perceptrons, Long/Short Term Memory (LSTM), Hopfield, Boltzmann, deep belief, deconvolutional, generative adversarial, liquid state machine, etc.), and/or other types of machine learning models.

[0607] In at least one embodiment, training pipelines 3904 may include AI-assisted annotation, as described in more detail herein with respect to at least FIG. 42B. In at least one embodiment, labeled clinic data 3812 (e.g., traditional annotation) may be generated by any number of techniques. In at least one embodiment, labels or other annotations may be generated within a drawing program (e.g., an annotation program), a computer aided design (CAD) program, a labeling program, another type of program suitable for generating annotations or labels for ground truth, and/or may be hand drawn, in some examples. In at least one embodiment, ground truth data may be synthetically produced (e.g., generated from computer models or renderings), real produced (e.g., designed and produced from real-world data), machine-automated (e.g., using feature analysis and learning to extract features from data and then generate labels), human annotated (e.g., labeler, or annotation expert, defines location of labels), and/or a combination thereof. In at least one embodiment, for each instance of imaging data 3808 (or other data type used by machine learning models), there may be corresponding ground truth data generated by training system 3804. In at least one embodiment, AI-assisted annotation may be performed as part of deployment pipelines 3910; either in addition to, or in lieu of AI-assisted annotation included in training pipelines 3904. In at least one embodiment, system 3900 may include a multi-layer platform that may include a software layer (e.g., software 3818) of diagnostic applications (or other application types) that may perform one or more medical imaging and diagnostic functions. In at least one embodiment, system 3900 may be communicatively coupled to (e.g., via encrypted links) PACS server networks of one or more facilities. In at least one embodiment, system 3900 may be configured to access and referenced data (e.g., DICOM data, RIS data, raw data, CIS data, REST compliant data, RPC data, raw data, etc.) from PACS servers (e.g., via a DICOM adapter 3902, or another data type adapter such as RIS, CIS, REST compliant, RPC, raw, etc.) to perform operations, such as training machine learning models, deploying machine learning models, image processing, inferencing, and/or other operations.

[0608] In at least one embodiment, a software layer may be implemented as a secure, encrypted, and/or authenticated API through which applications or containers may be invoked (e.g., called) from an external environment(s) (e.g., facility 3802). In at least one embodiment, applications may then call or execute one or more services 3820 for performing compute, AI, or visualization tasks associated with respective applications, and software 3818 and/or services

3820 may leverage hardware 3822 to perform processing tasks in an effective and efficient manner.

[0609] In at least one embodiment, deployment system 3806 may execute deployment pipelines 3910. In at least one embodiment, deployment pipelines 3910 may include any number of applications that may be sequentially, non-sequentially, or otherwise applied to imaging data (and/or other data types) generated by imaging devices, sequencing devices, genomics devices, etc. – including AI-assisted annotation, as described above. In at least one embodiment, as described herein, a deployment pipeline 3910 for an individual device may be referred to as a virtual instrument for a device (e.g., a virtual ultrasound instrument, a virtual CT scan instrument, a virtual sequencing instrument, etc.). In at least one embodiment, for a single device, there may be more than one deployment pipeline 3910 depending on information desired from data generated by a device. In at least one embodiment, where detections of anomalies are desired from an MRI machine, there may be a first deployment pipeline 3910, and where image enhancement is desired from output of an MRI machine, there may be a second deployment pipeline 3910.

[0610] In at least one embodiment, applications available for deployment pipelines 3910 may include any application that may be used for performing processing tasks on imaging data or other data from devices. In at least one embodiment, different applications may be responsible for image enhancement, segmentation, reconstruction, anomaly detection, object detection, feature detection, treatment planning, dosimetry, beam planning (or other radiation treatment procedures), and/or other analysis, image processing, or inferencing tasks. In at least one embodiment, deployment system 3806 may define constructs for each of applications, such that users of deployment system 3806 (e.g., medical facilities, labs, clinics, etc.) may understand constructs and adapt applications for implementation within their respective facility. In at least one embodiment, an application for image reconstruction may be selected for inclusion in deployment pipeline 3910, but data type generated by an imaging device may be different from a data type used within an application. In at least one embodiment, DICOM adapter 3902B (and/or a DICOM reader) or another data type adapter or reader (e.g., RIS, CIS, REST compliant, RPC, raw, etc.) may be used within deployment pipeline 3910 to convert data to a form useable by an application within deployment system 3806. In at least one embodiment, access to

DICOM, RIS, CIS, REST compliant, RPC, raw, and/or other data type libraries may be accumulated and pre-processed, including decoding, extracting, and/or performing any convolutions, color corrections, sharpness, gamma, and/or other augmentations to data. In at least one embodiment, DICOM, RIS, CIS, REST compliant, RPC, and/or raw data may be unordered and a pre-pass may be executed to organize or sort collected data. In at least one embodiment, because various applications may share common image operations, in some embodiments, a data augmentation library (e.g., as one of services 3820) may be used to accelerate these operations. In at least one embodiment, to avoid bottlenecks of conventional processing approaches that rely on CPU processing, parallel computing platform 3930 may be used for GPU acceleration of these processing tasks.

[0611] In at least one embodiment, an image reconstruction application may include a processing task that includes use of a machine learning model. In at least one embodiment, a user may desire to use their own machine learning model, or to select a machine learning model from model registry 3824. In at least one embodiment, a user may implement their own machine learning model or select a machine learning model for inclusion in an application for performing a processing task. In at least one embodiment, applications may be selectable and customizable, and by defining constructs of applications, deployment and implementation of applications for a particular user are presented as a more seamless user experience. In at least one embodiment, by leveraging other features of system 3900 – such as services 3820 and hardware 3822 – deployment pipelines 3910 may be even more user friendly, provide for easier integration, and produce more accurate, efficient, and timely results.

[0612] In at least one embodiment, deployment system 3806 may include a user interface 3914 (e.g., a graphical user interface, a web interface, etc.) that may be used to select applications for inclusion in deployment pipeline(s) 3910, arrange applications, modify or change applications or parameters or constructs thereof, use and interact with deployment pipeline(s) 3910 during set-up and/or deployment, and/or to otherwise interact with deployment system 3806. In at least one embodiment, although not illustrated with respect to training system 3804, user interface 3914 (or a different user interface) may be used for selecting models for use in deployment system 3806, for selecting models for training, or retraining, in training system 3804, and/or for otherwise interacting with training system 3804.

[0613] In at least one embodiment, pipeline manager 3912 may be used, in addition to an application orchestration system 3928, to manage interaction between applications or containers of deployment pipeline(s) 3910 and services 3820 and/or hardware 3822. In at least one embodiment, pipeline manager 3912 may be configured to facilitate interactions from application to application, from application to service 3820, and/or from application or service to hardware 3822. In at least one embodiment, although illustrated as included in software 3818, this is not intended to be limiting, and in some examples (e.g., as illustrated in FIG. 40) pipeline manager 3912 may be included in services 3820. In at least one embodiment, application orchestration system 3928 (e.g., Kubernetes, DOCKER, etc.) may include a container orchestration system that may group applications into containers as logical units for coordination, management, scaling, and deployment. In at least one embodiment, by associating applications from deployment pipeline(s) 3910 (e.g., a reconstruction application, a segmentation application, etc.) with individual containers, each application may execute in a self-contained environment (e.g., at a kernel level) to increase speed and efficiency.

[0614] 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 3912 and application orchestration system 3928. In at least one embodiment, so long as an expected input and/or output of each container or application is known by a system (e.g., based on constructs of applications or containers), application orchestration system 3928 and/or pipeline manager 3912 may facilitate communication among and between, and sharing of resources among and between, each of applications or containers. In at least one embodiment, because one or more of applications or containers in deployment pipeline(s) 3910 may share same services and resources, application orchestration system 3928 may orchestrate, load balance, and determine sharing of services or resources between and among various applications or containers. In at least one embodiment, a scheduler may be used to track resource requirements of applications or containers, current usage or planned usage of these resources, and resource

availability. In at least one embodiment, a scheduler may thus allocate resources to different applications and distribute resources between and among applications in view of requirements and availability of a system. In some examples, a scheduler (and/or other component of application orchestration system 3928 such as a sequencer and/or asynchronous compute engine) 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.

[0615] In at least one embodiment, services 3820 leveraged by and shared by applications or containers in deployment system 3806 may include compute services 3916, AI services 3918, visualization services 3920, and/or other service types. In at least one embodiment, applications may call (e.g., execute) one or more of services 3820 to perform processing operations for an application. In at least one embodiment, compute services 3916 may be leveraged by applications to perform super-computing or other high-performance computing (HPC) tasks. In at least one embodiment, compute service(s) 3916 may be leveraged to perform parallel processing (e.g., using a parallel computing platform 3930) for processing 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 3930 (e.g., NVIDIA's CUDA) may enable general purpose computing on GPUs (GPGPU) (e.g., GPUs 3922). In at least one embodiment, a software layer of parallel computing platform 3930 may provide access to virtual instruction sets and parallel computational elements of GPUs, for execution of compute kernels. In at least one embodiment, parallel computing platform 3930 may include memory and, in some embodiments, a memory may be shared between and among multiple containers, and/or between and among different processing tasks within a single container. In at least one embodiment, inter-process communication (IPC) calls may be generated for multiple containers and/or for multiple processes within a container to use same data from a shared segment of memory of parallel computing platform 3930 (e.g., where multiple different stages of an application or multiple applications are processing same information). In at least one embodiment, rather than making a copy of data and moving data to different locations in memory (e.g., a read/write operation), same data in same location of a memory may be used for any number of processing tasks (e.g., at a same time, at different times, etc.). In at least one embodiment, as data is used to generate new data as a result of processing, this information of a

new location of data may be stored and shared between various applications. In at least one embodiment, location of data and a location of updated or modified data may be part of a definition of how a payload is understood within containers.

[0616] In at least one embodiment, AI services 3918 may be leveraged to perform inferencing services for executing machine learning model(s) associated with applications (e.g., tasked with performing one or more processing tasks of an application). In at least one embodiment, AI services 3918 may leverage AI system 3924 to execute machine learning model(s) (e.g., neural networks, such as CNNs) for segmentation, reconstruction, object detection, feature detection, classification, and/or other inferencing tasks. In at least one embodiment, applications of deployment pipeline(s) 3910 may use one or more of output models 3816 from training system 3804 and/or other models of applications to perform inferencing on imaging data (e.g., DICOM data, RIS data, CIS data, REST compliant data, RPC data, raw data, etc.). In at least one embodiment, two or more examples of inferencing using application orchestration system 3928 (e.g., a scheduler, sequencer, and/or asynchronous compute engine) 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 3928 may distribute resources (e.g., services 3820 and/or hardware 3822) based on priority paths for different inferencing tasks of AI services 3918.

[0617] In at least one embodiment, shared storage may be mounted to AI services 3918 within system 3900. In at least one embodiment, shared storage may operate as a cache (or other storage device type) and may be used to process inference requests from applications. In at least one embodiment, when an inference request is submitted, a request may be received by a set of API instances of deployment system 3806, and one or more instances may be selected (e.g., for best fit, for load balancing, etc.) to process a request. In at least one embodiment, to process a request, a request may be entered into a database, a machine learning model may be located from model registry 3824 if not already in a cache, a validation step may ensure appropriate machine learning model is loaded into a cache (e.g., shared storage), and/or a copy of a model may be

saved to a cache. In at least one embodiment, a scheduler (e.g., of pipeline manager 3912) may be used to launch an application that is referenced in a request if an application is not already running or if there are not enough instances of an application. In at least one embodiment, if an inference server is not already launched to execute a model, an inference server may be launched. In at least one embodiment, any number of inference servers may be launched per model. In at least one embodiment, in a pull model, in which inference servers are clustered, models may be cached whenever load balancing is advantageous. In at least one embodiment, inference servers may be statically loaded in corresponding, distributed servers.

[0618] 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 inferencing 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.

[0619] 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 inferencing 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.

[0620] In at least one embodiment, transfer of requests between services 3820 and inference applications may be hidden behind a software development kit (SDK), and robust transport may be provided 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 3926, and an inference service may perform inferencing on a GPU.

[0621] In at least one embodiment, visualization services 3920 may be leveraged to generate visualizations for viewing outputs of applications and/or deployment pipeline(s) 3910. In at least one embodiment, GPUs 3922 may be leveraged by visualization services 3920 to generate visualizations. In at least one embodiment, rendering effects, such as ray-tracing, may be implemented by visualization services 3920 to generate higher quality visualizations. In at least one embodiment, visualizations may include, without limitation, 2D image renderings, 3D volume renderings, 3D volume reconstruction, 2D tomographic slices, virtual reality displays, augmented reality displays, etc. In at least one embodiment, virtualized environments may be used to generate a virtual interactive display or environment (e.g., a virtual environment) for interaction by users of a system (e.g., doctors, nurses, radiologists, etc.). In at least one embodiment, visualization services 3920 may include an internal visualizer, cinematics, and/or other rendering or image processing capabilities or functionality (e.g., ray tracing, rasterization, internal optics, etc.).

[0622] In at least one embodiment, hardware 3822 may include GPUs 3922, AI system 3924, cloud 3926, and/or any other hardware used for executing training system 3804 and/or deployment system 3806. In at least one embodiment, GPUs 3922 (e.g., NVIDIA's TESLA and/or QUADRO GPUs) may include any number of GPUs that may be used for executing processing tasks of compute services 3916, AI services 3918, visualization services 3920, other services, and/or any of features or functionality of software 3818. For example, with respect to AI services 3918, GPUs 3922 may be used to perform pre-processing on imaging data (or other data types used by machine learning models), post-processing on outputs of machine learning models, and/or to perform inferencing (e.g., to execute machine learning models). In at least one embodiment, cloud 3926, AI system 3924, and/or other components of system 3900 may use GPUs 3922. In at least one embodiment, cloud 3926 may include a GPU-optimized platform for deep learning tasks. In at least one embodiment, AI system 3924 may use GPUs, and cloud 3926 – or at least a portion tasked with deep learning or inferencing – may be executed using one or more AI systems 3924. As such, although hardware 3822 is illustrated as discrete components, this is not intended to be limiting, and any components of hardware 3822 may be combined with, or leveraged by, any other components of hardware 3822.

[0623] In at least one embodiment, AI system 3924 may include a purpose-built computing system (e.g., a super-computer or an HPC) configured for inferencing, deep learning, machine learning, and/or other artificial intelligence tasks. In at least one embodiment, AI system 3924 (e.g., NVIDIA's DGX) may include GPU-optimized software (e.g., a software stack) that may be executed using a plurality of GPUs 3922, in addition to CPUs, RAM, storage, and/or other components, features, or functionality. In at least one embodiment, one or more AI systems 3924 may be implemented in cloud 3926 (e.g., in a data center) for performing some or all of AI-based processing tasks of system 3900.

[0624] In at least one embodiment, cloud 3926 may include a GPU-accelerated infrastructure (e.g., NVIDIA's NGC) that may provide a GPU-optimized platform for executing processing tasks of system 3900. In at least one embodiment, cloud 3926 may include an AI system(s) 3924 for performing one or more of AI-based tasks of system 3900 (e.g., as a hardware abstraction and scaling platform). In at least one embodiment, cloud 3926 may integrate with application orchestration system 3928 leveraging multiple GPUs to enable seamless scaling and load

balancing between and among applications and services 3820. In at least one embodiment, cloud 3926 may be tasked with executing at least some of services 3820 of system 3900, including compute services 3916, AI services 3918, and/or visualization services 3920, as described herein. In at least one embodiment, cloud 3926 may perform small and large batch inference (e.g., executing NVIDIA's TENSOR RT), provide an accelerated parallel computing API and platform 3930 (e.g., NVIDIA's CUDA), execute application orchestration system 3928 (e.g., KUBERNETES), provide a graphics rendering API and platform (e.g., for ray-tracing, 2D graphics, 3D graphics, and/or other rendering techniques to produce higher quality cinematics), and/or may provide other functionality for system 3900.

[0625] In at least one embodiment, in an effort to preserve patient confidentiality (e.g., where patient data or records are to be used off-premises), cloud 3926 may include a registry – such as a deep learning container registry. In at least one embodiment, a registry may store containers for instantiations of applications that may perform pre-processing, post-processing, or other processing tasks on patient data. In at least one embodiment, cloud 3926 may receive data that includes patient data as well as sensor data in containers, perform requested processing for just sensor data in those containers, and then forward a resultant output and/or visualizations to appropriate parties and/or devices (e.g., on-premises medical devices used for visualization or diagnoses), all without having to extract, store, or otherwise access patient data. In at least one embodiment, confidentiality of patient data is preserved in compliance with HIPAA and/or other data regulations.

[0626] In at least one embodiment, at least one component shown or described with respect to FIG. 39 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, AI system 3924 performs one or more operations related to inserting neural network masks into a trained, dense neural model and/or to creating an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0627] FIG. 40 includes an example illustration of a deployment pipeline 3910A for processing imaging data, in accordance with at least one embodiment. In at least one embodiment, system 3900 – and specifically deployment system 3806 – may be used to customize, update, and/or integrate deployment pipeline(s) 3910A into one or more production environments. In at least

one embodiment, deployment pipeline 3910A of FIG. 40 includes a non-limiting example of a deployment pipeline 3910A that may be custom defined by a particular user (or team of users) at a facility (e.g., at a hospital, clinic, lab, research environment, etc.). In at least one embodiment, to define deployment pipelines 3910A for a CT scanner 4002, a user may select – from a container registry, for example – one or more applications that perform specific functions or tasks with respect to imaging data generated by CT scanner 4002. In at least one embodiment, applications may be applied to deployment pipeline 3910A as containers that may leverage services 3820 and/or hardware 3822 of system 3900. In addition, deployment pipeline 3910A may include additional processing tasks or applications that may be implemented to prepare data for use by applications (e.g., DICOM adapter 3902B and DICOM reader 4006 may be used in deployment pipeline 3910A to prepare data for use by CT reconstruction 4008, organ segmentation 4010, etc.). In at least one embodiment, deployment pipeline 3910A may be customized or selected for consistent deployment, one time use, or for another frequency or interval. In at least one embodiment, a user may desire to have CT reconstruction 4008 and organ segmentation 4010 for several subjects over a specific interval, and thus may deploy pipeline 3910A for that period of time. In at least one embodiment, a user may select, for each request from system 3900, applications that a user wants to perform processing on that data for that request. In at least one embodiment, deployment pipeline 3910A may be adjusted at any interval and, because of adaptability and scalability of a container structure within system 3900, this may be a seamless process.

[0628] In at least one embodiment, deployment pipeline 3910A of FIG. 40 may include CT scanner 4002 generating imaging data of a patient or subject. In at least one embodiment, imaging data from CT scanner 4002 may be stored on a PACS server(s) 4004 associated with a facility housing CT scanner 4002. In at least one embodiment, PACS server(s) 4004 may include software and/or hardware components that may directly interface with imaging modalities (e.g., CT scanner 4002) at a facility. In at least one embodiment, DICOM adapter 3902B may enable sending and receipt of DICOM objects using DICOM protocols. In at least one embodiment, DICOM adapter 3902B may aid in preparation or configuration of DICOM data from PACS server(s) 4004 for use by deployment pipeline 3910A. In at least one embodiment, once DICOM data is processed through DICOM adapter 3902B, pipeline manager 3912 may route data through to deployment pipeline 3910A. In at least one embodiment,

DICOM reader 4006 may extract image files and any associated metadata from DICOM data (e.g., raw sinogram data, as illustrated in visualization 4016A). In at least one embodiment, working files that are extracted may be stored in a cache for faster processing by other applications in deployment pipeline 3910A. In at least one embodiment, once DICOM reader 4006 has finished extracting and/or storing data, a signal of completion may be communicated to pipeline manager 3912. In at least one embodiment, pipeline manager 3912 may then initiate or call upon one or more other applications or containers in deployment pipeline 3910A.

[0629] In at least one embodiment, CT reconstruction 4008 application and/or container may be executed once data (e.g., raw sinogram data) is available for processing by CT reconstruction 4008 application. In at least one embodiment, CT reconstruction 4008 may read raw sinogram data from a cache, reconstruct an image file out of raw sinogram data (e.g., as illustrated in visualization 4016B), and store resulting image file in a cache. In at least one embodiment, at completion of reconstruction, pipeline manager 3912 may be signaled that reconstruction task is complete. In at least one embodiment, once reconstruction is complete, and a reconstructed image file may be stored in a cache (or other storage device), organ segmentation 4010 application and/or container may be triggered by pipeline manager 3912. In at least one embodiment, organ segmentation 4010 application and/or container may read an image file from a cache, normalize or convert an image file to format suitable for inference (e.g., convert an image file to an input resolution of a machine learning model), and run inference against a normalized image. In at least one embodiment, to run inference on a normalized image, organ segmentation 4010 application and/or container may rely on services 3820, and pipeline manager 3912 and/or application orchestration system 3928 may facilitate use of services 3820 by organ segmentation 4010 application and/or container. In at least one embodiment, for example, organ segmentation 4010 application and/or container may leverage AI services 3918 to perform inferencing on a normalized image, and AI services 3918 may leverage hardware 3822 (e.g., AI system 3924) to execute AI services 3918. In at least one embodiment, a result of an inference may be a mask file (e.g., as illustrated in visualization 4016C) that may be stored in a cache (or other storage device).

[0630] 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 3912. In at least one embodiment, pipeline manager 3912 may then execute DICOM writer 4012 to read results from a cache (or other storage device), package results into a DICOM format (e.g., as DICOM output 4014) for use by users at a facility who generated a request. In at least one embodiment, DICOM output 4014 may then be transmitted to DICOM adapter 3902B to prepare DICOM output 4014 for storage on PACS server(s) 4004 (e.g., for viewing by a DICOM viewer at a facility). In at least one embodiment, in response to a request for reconstruction and segmentation, visualizations 4016B and 4016C may be generated and available to a user for diagnoses, research, and/or for other purposes.

[0631] Although illustrated as consecutive application in deployment pipeline 3910A, CT reconstruction 4008 and organ segmentation 4010 applications may be processed in parallel in at least one embodiment. In at least one embodiment, where applications do not have dependencies on one another, and data is available for each application (e.g., after DICOM reader 4006 extracts data), applications may be executed at a same time, substantially at a same time, or with some overlap. In at least one embodiment, where two or more applications require similar services 3820, a scheduler of system 3900 may be used to load balance and distribute compute or processing resources between and among various applications. In at least one embodiment, in some embodiments, parallel computing platform 3930 may be used to perform parallel processing for applications to decrease run-time of deployment pipeline 3910A to provide real-time results.

[0632] In at least one embodiment, and with reference to FIGS. 41A-41B, deployment system 3806 may be implemented as one or more virtual instruments to perform different functionalities – such as image processing, segmentation, enhancement, AI, visualization, and inferencing – with imaging devices (e.g., CT scanners, X-ray machines, MRI machines, etc.), sequencing devices, genomics devices, and/or other device types. In at least one embodiment, system 3900 may allow for creation and provision of virtual instruments that may include a software-defined deployment pipeline 3910 that may receive raw/unprocessed input data generated by a device(s) and output processed/reconstructed data. In at least one embodiment, deployment pipelines 3910 (e.g., 3910A and 3910B) that represent virtual instruments may implement intelligence into a pipeline, such as by leveraging machine learning models, to provide containerized inference support to a system. In at least one 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 3910 representing virtual instruments may be static (e.g., containers and/or applications may be set), while in other examples, container and/or applications for virtual instruments may be selected (e.g., on a per-request basis) from a pool of applications or resources (e.g., within a container registry).

[0633] In at least one embodiment, system 3900 may be instantiated or executed as one or more virtual instruments on-premise at a facility in, for example, a computing system deployed next to or otherwise in communication with a radiology machine, an imaging device, and/or another device type at a facility. In at least one embodiment, however, an on-premise installation may be instantiated or executed within a computing system of a device itself (e.g., a computing system integral to an imaging device), in a local datacenter (e.g., a datacenter on-premise), and/or in a cloud-environment (e.g., in cloud 3926). In at least one embodiment, deployment system 3806, operating as a virtual instrument, may be instantiated by a supercomputer or other HPC system in some examples. In at least one embodiment, on-premise installation may allow for high-bandwidth uses (via, for example, higher throughput local communication interfaces, such as RF over Ethernet) for real-time processing. In at least one embodiment, real-time or near real-time processing may be particularly useful where a virtual instrument supports an ultrasound device or other imaging modality where immediate visualizations are expected or required for accurate diagnoses and analyses. In at least one embodiment, a cloud-computing architecture may be capable of dynamic bursting to a cloud computing service provider, or other compute cluster, when local demand exceeds on-premise capacity or capability. In at least one embodiment, a cloud architecture, when implemented, may be tuned for training neural networks or other machine learning models, as described herein with respect to training system 3804. In at least one embodiment, with training pipelines in place, machine learning models may 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.

[0634] In at least one embodiment, a computing system may include some or all of hardware 3822 described herein, and hardware 3822 may be distributed in any of a number of ways including within a device, as part of a computing device coupled to and located proximate a

device, in a local datacenter at a facility, and/or in cloud 3926. In at least one embodiment, because deployment system 3806 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.

[0635] In at least one embodiment, at least one component shown or described with respect to FIG. 40 is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, organ segmentation 4010 uses one or more efficient neural models for deployment on a GPU hardware platform as described in conjunction with FIG. 6, and as otherwise described herein.

[0636] FIG. 41A includes an example data flow diagram of a virtual instrument supporting an ultrasound device, in accordance with at least one embodiment. In at least one embodiment, deployment pipeline 3910B may leverage one or more of services 3820 of system 3900. In at least one embodiment, deployment pipeline 3910B and services 3820 may leverage hardware 3822 of a system either locally or in cloud 3926. In at least one embodiment, although not illustrated, process 4100 may be facilitated by pipeline manager 3912, application orchestration system 3928, and/or parallel computing platform 3930.

[0637] In at least one embodiment, process 4100 may include receipt of imaging data from an ultrasound device 4102. In at least one embodiment, imaging data may be stored on PACS server(s) in a DICOM format (or other format, such as RIS, CIS, REST compliant, RPC, raw, etc.), and may be received by system 3900 for processing through deployment pipeline 3910 selected or customized as a virtual instrument (e.g., a virtual ultrasound) for ultrasound device 4102. In at least one embodiment, imaging data may be received directly from an imaging device (e.g., ultrasound device 4102) and processed by a virtual instrument. In at least one embodiment, a transducer or other signal converter communicatively coupled between an imaging device and a virtual instrument may convert signal data generated by an imaging device to image data that may be processed by a virtual instrument. In at least one embodiment, raw data and/or image data may be applied to DICOM reader 4006 to extract data for use by applications or containers of deployment pipeline 3910B. In at least one embodiment, DICOM

reader 4006 may leverage data augmentation library 4114 (e.g., NVIDIA's DALI) as a service 3820 (e.g., as one of compute service(s) 3916) for extracting, resizing, rescaling, and/or otherwise preparing data for use by applications or containers.

[0638] In at least one embodiment, once data is prepared, a reconstruction 4106 application and/or container may be executed to reconstruct data from ultrasound device 4102 into an image file. In at least one embodiment, after reconstruction 4106, or at a same time as reconstruction 4106, a detection 4108 application and/or container may be executed for anomaly detection, object detection, feature detection, and/or other detection tasks related to data. In at least one embodiment, an image file generated during reconstruction 4106 may be used during detection 4108 to identify anomalies, objects, features, etc. In at least one embodiment, detection 4108 application may leverage an inference engine 4116 (e.g., as one of AI service(s) 3918) to perform inferencing on data to generate detections. In at least one embodiment, one or more machine learning models (e.g., from training system 3804) may be executed or called by detection 4108 application.

[0639] In at least one embodiment, once reconstruction 4106 and/or detection 4108 is/are complete, data output from these application and/or containers may be used to generate visualizations 4110, such as visualization 4112 (e.g., a grayscale output) displayed on a workstation or display terminal. In at least one embodiment, visualization may allow a technician or other user to visualize results of deployment pipeline 3910B with respect to ultrasound device 4102. In at least one embodiment, visualization 4110 may be executed by leveraging a render component 4118 of system 3900 (e.g., one of visualization service(s) 3920). In at least one embodiment, render component 4118 may execute a 2D, OpenGL, or ray-tracing service to generate visualization 4112.

[0640] In at least one embodiment, at least one component shown or described with respect to FIG. 41A is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, inference engine 4115 uses one or more efficient neural models for deployment on a GPU hardware platform as described in conjunction with FIG. 6, and as otherwise described herein.

[0641] FIG. 41B includes an example data flow diagram of a virtual instrument supporting a CT scanner, in accordance with at least one embodiment. In at least one embodiment,

deployment pipeline 3910C may leverage one or more of services 3820 of system 3900. In at least one embodiment, deployment pipeline 3910C and services 3820 may leverage hardware 3822 of a system either locally or in cloud 3926. In at least one embodiment, although not illustrated, process 4120 may be facilitated by pipeline manager 3912, application orchestration system 3928, and/or parallel computing platform 3930.

[0642] In at least one embodiment, process 4120 may include CT scanner 4122 generating raw data that may be received by DICOM reader 4006 (e.g., directly, via a PACS server 4004, after processing, etc.). In at least one embodiment, a Virtual CT (instantiated by deployment pipeline 3910C) may include a first, real-time pipeline for monitoring a patient (e.g., patient movement detection AI 4126) and/or for adjusting or optimizing exposure of CT scanner 4122 (e.g., using exposure control AI 4124). In at least one embodiment, one or more of applications (e.g., 4124 and 4126) may leverage a service 3820, such as AI service(s) 3918. In at least one embodiment, outputs of exposure control AI 4124 application (or container) and/or patient movement detection AI 4126 application (or container) may be used as feedback to CT scanner 4122 and/or a technician for adjusting exposure (or other settings of CT scanner 4122) and/or informing a patient to move less.

[0643] In at least one embodiment, deployment pipeline 3910C may include a non-real-time pipeline for analyzing data generated by CT scanner 4122. In at least one embodiment, a second pipeline may include CT reconstruction 4008 application and/or container, a coarse detection AI 4128 application and/or container, a fine detection AI 4132 application and/or container (e.g., where certain results are detected by coarse detection AI 4128), a visualization 4130 application and/or container, and a DICOM writer 4012 (and/or other data type writer, such as RIS, CIS, REST compliant, RPC, raw, etc.) application and/or container. In at least one embodiment, raw data generated by CT scanner 4122 may be passed through pipelines of deployment pipeline 3910C (instantiated as a virtual CT instrument) to generate results. In at least one embodiment, results from DICOM writer 4012 may be transmitted for display and/or may be stored on PACS server(s) 4004 for later retrieval, analysis, or display by a technician, practitioner, or other user.

[0644] In at least one embodiment, at least one component shown or described with respect to FIG. 41B is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, exposure control AI 4124 uses one or more efficient neural

models for deployment on a GPU hardware platform as described in conjunction with FIG. 6, and as otherwise described herein.

[0645] FIG. 42A illustrates a data flow diagram for a process 4200 to train, retrain, or update a machine learning model, in accordance with at least one embodiment. In at least one embodiment, process 4200 may be executed using, as a non-limiting example, system 3900 of FIG. 39. In at least one embodiment, process 4200 may leverage services 3820 and/or hardware 3822 of system 3900, as described herein. In at least one embodiment, refined models 4212 generated by process 4200 may be executed by deployment system 3806 for one or more containerized applications in deployment pipelines 3910.

[0646] In at least one embodiment, model training 3814 may include retraining or updating an initial model 4204 (e.g., a pre-trained model) using new training data (e.g., new input data, such as customer dataset 4206, and/or new ground truth data associated with input data). In at least one embodiment, to retrain, or update, initial model 4204, output or loss layer(s) of initial model 4204 may be reset, or deleted, and/or replaced with an updated or new output or loss layer(s). In at least one embodiment, initial model 4204 may have previously fine-tuned parameters (e.g., weights and/or biases) that remain from prior training, so training or retraining 3814 may not take as long or require as much processing as training a model from scratch. In at least one embodiment, during model training 3814, by having reset or replaced output or loss layer(s) of initial model 4204, parameters may be updated and re-tuned for a new data set based on loss calculations associated with accuracy of output or loss layer(s) at generating predictions on new, customer dataset 4206 (e.g., image data 3808 of FIG. 38).

[0647] In at least one embodiment, pre-trained models 3906 may be stored in a data store, or registry (e.g., model registry 3824 of FIG. 38). In at least one embodiment, pre-trained models 3906 may have been trained, at least in part, at one or more facilities other than a facility executing process 4200. In at least one embodiment, to protect privacy and rights of patients, subjects, or clients of different facilities, pre-trained models 3906 may have been trained, on-premise, using customer or patient data generated on-premise. In at least one embodiment, pre-trained models 3906 may be trained using cloud 3926 and/or other hardware 3822, but confidential, privacy protected patient data may not be transferred to, used by, or accessible to any components of cloud 3926 (or other off premise hardware). In at least one embodiment,

where a pre-trained model 3906 is trained at using patient data from more than one facility, pre-trained model 3906 may have been individually trained for each facility prior to being trained on patient or customer data from another facility. In at least one embodiment, such as where a customer or patient data has been released of privacy concerns (e.g., by waiver, for experimental use, etc.), or where a customer or patient data is included in a public data set, a customer or patient data from any number of facilities may be used to train pre-trained model 3906 on-premise and/or off premise, such as in a datacenter or other cloud computing infrastructure.

[0648] In at least one embodiment, when selecting applications for use in deployment pipelines 3910, a user may also select machine learning models to be used for specific applications. In at least one embodiment, a user may not have a model for use, so a user may select a pre-trained model 3906 to use with an application. In at least one embodiment, pre-trained model 3906 may not be optimized for generating accurate results on customer dataset 4206 of a facility of a user (e.g., based on patient diversity, demographics, types of medical imaging devices used, etc.). In at least one embodiment, prior to deploying pre-trained model 3906 into deployment pipeline 3910 for use with an application(s), pre-trained model 3906 may be updated, retrained, and/or fine-tuned for use at a respective facility.

[0649] In at least one embodiment, a user may select pre-trained model 3906 that is to be updated, retrained, and/or fine-tuned, and pre-trained model 3906 may be referred to as initial model 4204 for training system 3804 within process 4200. In at least one embodiment, customer dataset 4206 (e.g., imaging data, genomics data, sequencing data, or other data types generated by devices at a facility) may be used to perform model training 3814 (which may include, without limitation, transfer learning) on initial model 4204 to generate refined model 4212. In at least one embodiment, ground truth data corresponding to customer dataset 4206 may be generated by training system 3804. In at least one embodiment, ground truth data may be generated, at least in part, by clinicians, scientists, doctors, practitioners, at a facility (e.g., as labeled clinic data 3812 of FIG. 38).

[0650] In at least one embodiment, AI-assisted annotation 3810 may be used in some examples to generate ground truth data. In at least one embodiment, AI-assisted annotation 3810 (e.g., implemented using an AI-assisted annotation SDK) may leverage machine learning models (e.g., neural networks) to generate suggested or predicted ground truth data for a customer dataset. In

at least one embodiment, user 4210 may use annotation tools within a user interface (a graphical user interface (GUI)) on computing device 4208.

[0651] In at least one embodiment, user 4210 may interact with a GUI via computing device 4208 to edit or fine-tune annotations or auto-annotations. In at least one embodiment, a polygon editing feature may be used to move vertices of a polygon to more accurate or fine-tuned locations.

[0652] In at least one embodiment, once customer dataset 4206 has associated ground truth data, ground truth data (e.g., from AI-assisted annotation, manual labeling, etc.) may be used during model training 3814 to generate refined model 4212. In at least one embodiment, customer dataset 4206 may be applied to initial model 4204 any number of times, and ground truth data may be used to update parameters of initial model 4204 until an acceptable level of accuracy is attained for refined model 4212. In at least one embodiment, once refined model 4212 is generated, refined model 4212 may be deployed within one or more deployment pipelines 3910 at a facility for performing one or more processing tasks with respect to medical imaging data.

[0653] In at least one embodiment, refined model 4212 may be uploaded to pre-trained models 3906 in model registry 3824 to be selected by another facility. In at least one embodiment, this process may be completed at any number of facilities such that refined model 4212 may be further refined on new datasets any number of times to generate a more universal model.

[0654] In at least one embodiment, at least one component shown or described with respect to FIG. 42A is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, model training system 4604 trains one or more neural network models that uses one or more neural network masks to, at least in part, create an efficient neural model for deployment on a GPU hardware platform as described in conjunction with FIGS. 5 and 6, and as otherwise described herein.

[0655] FIG. 42B is an example illustration of a client-server architecture 4232 to enhance annotation tools with pre-trained annotation models, in accordance with at least one embodiment. In at least one embodiment, AI-assisted annotation tools 4236 may be instantiated based on a client-server architecture 4232. In at least one embodiment, annotation tools 4236 in imaging

applications may aid radiologists, for example, identify organs and abnormalities. In at least one embodiment, imaging applications may include software tools that help user 4210 to identify, as a non-limiting example, a few extreme points on a particular organ of interest in raw images 4234 (e.g., in a 3D MRI or CT scan) and receive auto-annotated results for all 2D slices of a particular organ. In at least one embodiment, results may be stored in a data store as training data 4238 and used as (for example and without limitation) ground truth data for training. In at least one embodiment, when computing device 4208 sends extreme points for AI-assisted annotation 3810, a deep learning model, for example, may receive this data as input and return inference results of a segmented organ or abnormality. In at least one embodiment, pre-instantiated annotation tools, such as AI-Assisted Annotation Tool 4236 in FIG. 42B, may be enhanced by making API calls (e.g., API Call 4244) to a server, such as an Annotation Assistant Server 4240 that may include a set of pre-trained models 4242 stored in an annotation model registry, for example. In at least one embodiment, an annotation model registry may store pre-trained models 4242 (e.g., machine learning models, such as deep learning models) that are pre-trained to perform AI-assisted annotation on a particular organ or abnormality. In at least one embodiment, these models may be further updated by using training pipelines 3904. In at least one embodiment, pre-installed annotation tools may be improved over time as new labeled clinic data 3812 is added.

[0656] Logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, logic 915 may be used in AI-assisted annotation tool 4236 to infer annotations in medical images to identify organs and/or abnormalities. In at least one embodiment, logic 915 may be used in AI-assisted annotation tool 4236 to infer annotations 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.

[0657] In at least one embodiment, at least one component shown or described with respect to FIG. 42B is used to implement techniques and/or functions described in connection with FIGS. 1A-8B. In at least one embodiment, AI assisted annotation tool 4236 uses one or more efficient

neural models for deployment on a GPU hardware platform as described in conjunction with FIG. 6, and as otherwise described herein.

[0658] At least one embodiment of the disclosure can be described in view of the following clauses:

1. A processor comprising:
one or more circuits to select one or more masks to be used with one or more neural networks based, at least in part, on one or more hardware resources to use the one or more neural networks.
2. The processor of clause 1, wherein the one or more circuits are to select the one or more masks based, at least in part, on one or more metrics generated by performing variations of the one or more neural networks using the one or more hardware resources.
3. The processor of any of clauses 1-2, wherein the one or more circuits are to select the one or more masks to be used with the one or more neural networks to modify one or more data values of the one or more neural networks.
4. The processor of any of clauses 1-3, wherein the one or more circuits are to prune one or more layers of the one or more neural networks using the one or more masks.
5. The processor of any of clauses 1-4, wherein the one or more circuits are to select the one or more masks based, at least in part, on a list that correlates the one or more metrics with one or more optimization techniques and one or more parameters of the hardware resources.
6. The processor of any of clauses 1-5, wherein the one or more circuits are to select a mask from the selected one or more masks to be used with the one or more neural networks based, at least in part, on an identification of a hardware resource.
7. The processor of any of clauses 1-6, wherein the one or more circuits are to modify one or more parameters to be used with the one or more neural networks based, at least in part, on one or more dense versions of the one or more neural networks.

8. A system, comprising:

one or more processors to select one or more masks to be used with one or more neural networks based, at least in part, on one or more hardware resources to use the one or more neural networks.

9. The system of clause 8, wherein the one or more processors are to identify the one or more masks based, at least in part, on performance metrics generated by the one or more hardware resources performing the one or more neural networks.

10. The system of any of clauses 8-9, wherein the one or more processors are to quantize one or more values of the one or more neural networks based, at least in part, on the one or more masks.

11. The system of any of clauses 8-10, wherein the one or more processors are to generate one or more sparse neural networks using the selected one or more masks with the one or more neural networks.

12. The system of any of clauses 8-11, wherein the one or more processors are to select the one or more masks based, at least in part, on performance metrics generated by the one or more hardware resources performing the one or more neural networks that exceed one or more threshold performance metrics.

13. The system of any of clauses 8-12, wherein the one or more circuits are to select the one or more masks, based, at least in part, on one or more parameters of a hardware resource.

14. The system of any of clauses 8-13, wherein the one or more circuits are further to train the one or more neural networks using the one or more selected masks.

15. A method, comprising:

selecting one or more masks to be used with one or more neural networks based, at least in part, on one or more hardware resources to use the one or more neural networks.

16. The method of clause 15, wherein the one or more masks is identified based, at least in part, on performance metrics generated by using the one or more hardware resources

performing the one or more neural network models modified with a quantization of one or more values.

17. The method of any of clauses 15-16, wherein the one or more masks is to be used with the one or more neural networks to prune one or more portions of the one or more neural networks.

18. The method of any of clauses 15-17 further comprising generating one or more sparse neural networks using the one or more masks to reduce one or more layers of the one or more neural networks.

19. The method of any of clauses 15-18, wherein the one or more neural networks are to be performed by the one or more hardware resources to exceed one or more threshold performance metrics that indicate accuracy and one or more threshold values that indicate inferencing speed.

20. The method of any of clauses 15-19, wherein the one or more masks are one or more tensors that indicate which nonzero values of the one or more neural networks should be set to zero.

[0659] 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.

[0660] In at least one embodiment, referring back to FIG. 15, computer programs in form of machine-readable executable code or computer control logic algorithms are stored in main memory 1504 and/or secondary storage. Computer programs, if executed by one or more processors, enable system 1500 to perform various functions in accordance with at least one embodiment. In at least one embodiment, memory 1504, storage, and/or any other storage are possible examples of computer-readable media. In at least one embodiment, secondary storage

may refer to any suitable storage device or system such as a hard disk drive and/or a removable storage drive, representing a floppy disk drive, a magnetic tape drive, a compact disk drive, digital versatile disk (“DVD”) drive, recording device, universal serial bus (“USB”) flash memory, etc. In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of CPU 1502, parallel processing system 1512, an integrated circuit capable of at least a portion of capabilities of both CPU 1502, parallel processing system 1512, a chipset (e.g., a group of integrated circuits designed to work and sold as a unit for performing related functions, etc.), and/or any suitable combination of integrated circuit(s).

[0661] In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of a general computer system, a circuit board system, a game console system dedicated for entertainment purposes, an application-specific system, and more. In at least one embodiment, computer system 1500 may take form of a desktop 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. In at least one embodiment, a computer system 1500 comprises or refers to any devices in Figures 9A-42B

[0662] In at least one embodiment, parallel processing system 1512 includes, without limitation, a plurality of parallel processing units (“PPUs”) 1514 and associated memories 1516. In at least one embodiment, PPUs 1514 are connected to a host processor or other peripheral devices via an interconnect 1518 and a switch 1520 or multiplexer. In at least one embodiment, parallel processing system 1512 distributes computational tasks across PPUs 1514 which can be parallelizable — for example, as part of distribution of computational tasks across multiple graphics processing unit (“GPU”) thread blocks. In at least one embodiment, memory is shared and accessible (e.g., for read and/or write access) across some or all of PPUs 1514, although such shared memory may incur performance penalties relative to use of local memory and registers resident to a PPU 1514. In at least one embodiment, operation of PPUs 1514 is synchronized through use of a command such as `__syncthreads()`, wherein all threads in a block (e.g., executed across multiple PPUs 1514) to reach a certain point of execution of code before proceeding.

[0663] FIG. 43 is a system diagram illustrating system 4300 for interfacing with an application 4302 to process data, according to at least one embodiment. In at least one embodiment, application 4302 uses large language model (LLM) 4312 to generate output data 4320 based, at least in part, on input data 4310. In at least one embodiment, input data 4310 is a text prompt. In at least one embodiment, input data 4310 includes unstructured text. In at least one embodiment, input data 4310 includes a sequence of tokens. In at least one embodiment, a token is a portion of input data. In at least one embodiment, a token is a word. In at least one embodiment, a token is a character. In at least one embodiment, a token is a subword. In at least one embodiment, input data 4310 is formatted in Chat Markup Language (ChatML). In at least one embodiment, input data 4310 is an image. In at least one embodiment, input data 4310 is one or more video frames. In at least one embodiment, input data 4310 is any other expressive medium.

[0664] In at least one embodiment, large language model 4312 comprises a deep neural network. In at least one embodiment, a deep neural network is a neural network with two or more layers. In at least one embodiment, large language model 4312 comprises a transformer model. In at least one embodiment, large language model 4312 comprises a neural network configured to perform natural language processing. In at least one embodiment, large language model 4312 is configured to process one or more sequences of data. In at least one embodiment, large language model 4312 is configured to process text. In at least one embodiment, weights and biases of a large language model 4312 are configured to process text. In at least one embodiment, large language model 4312 is configured to determine patterns in data to perform one or more natural language processing tasks. In at least one embodiment, a natural language processing task comprises text generation. In at least one embodiment, a natural language processing task comprises question answering. In at least one embodiment, performing a natural language processing task results in output data 4320.

[0665] In at least one embodiment, a processor uses input data 4310 to query retrieval database 4314. In at least one embodiment, retrieval database 4314 is a key-value store. In at least one embodiment, retrieval database 4314 is a corpus used to train large language model 4312. In at least one embodiment, a processor uses retrieval database 4314 to provide large language model 4312 with updated information. In at least one embodiment, retrieval database 4314 comprises

data from an internet source. In at least one embodiment, large language model 4312 does not use retrieval database 4314 to perform inferencing.

[0666] In at least one embodiment, an encoder encodes input data 4310 into one or more feature vectors. In at least one embodiment, an encoder encodes input data 4310 into a sentence embedding vector. In at least one embodiment, a processor uses said sentencing embedding vector to perform a nearest neighbor search to generate one or more neighbors 4316. In at least one embodiment, one or more neighbors 4316 is value in retrieval database 4314 corresponding to a key comprising input data 4310. In at least one embodiment, one or more neighbors 4316 comprise text data. In at least one embodiment, encoder 4318 encodes one or more neighbors 4316. In at least one embodiment, encoder 4318 encodes one or more neighbors 4316 into a text embedding vector. In at least one embodiment, encoder 4318 encodes one or more neighbors 4316 into a sentence embedding vector. In at least one embodiment, large language model 4316 uses input data 4310 and data generated by encoder 4318 to generate output data 4320. In at least one embodiment, processor 4306 interfaces with application 4302 using large language model (LLM) application programming interface(s) (API(s)) 4304. In at least one embodiment, processor 4306 accesses large language model 4316 using large language model (LLM) application programming interface(s) (API(s)) 4304.

[0667] In at least one embodiment, output data 4320 comprise computer instructions. In at least one embodiment, output data 4320 comprise instructions written in CUDA programming language. In at least one embodiment, output data 4320 comprise instructions to be performed by processor 4306. In at least one embodiment, output data 4320 comprise instructions to control execution of one or more algorithm modules 4308. In at least one embodiment, one or more algorithm modules 4308 comprise, for example, one or more neural networks to perform pattern recognition. In at least one embodiment, one or more algorithm modules 4308 comprise, for example, one or more neural networks to perform frame generation. In at least one embodiment, one or more algorithm modules 4308 comprise, for example, one or more neural networks to generate a drive path. In at least one embodiment, one or more algorithm modules 4308 comprise, for example, one or more neural networks to generate a 5G signal. In at least one embodiment, processor 4306 interfaces with application 4302 using large language model (LLM) application programming interface(s) (API(s)) 4304. In at least one embodiment, processor 4306 may use

one or more parallel computing platforms and/or programming models (e.g., NVIDIA's CUDA model).

[0668] In at least one embodiment, aspects of systems and techniques described herein in relation to FIG. 43 are incorporated into aspects of preceding figure(s), including FIGS. 1A-8. For example, in at least one embodiment, an apparatus depicted in preceding figure(s) includes processor 4306. In at least one embodiment, processor 436 performs one or more operations of one or more modules described in conjunction with network training system 100 of FIG. 1.

[0669] For example, in at least one embodiment, system 4300 uses ChatGPT to write CUDA code. For example, in at least one embodiment, system 4300 uses ChatGPT to train an object classification neural network. For example, in at least one embodiment, system 4300 uses ChatGPT and a neural network to identify a driving path. For example, in at least one embodiment, system 4300 uses ChatGPT and a neural network to generate a 5G signal.

[0670] In at least one embodiment, one or more techniques described herein utilize a oneAPI programming model. In at least one embodiment, a oneAPI programming model refers to a programming model for interacting with various compute accelerator architectures. In at least one embodiment, oneAPI refers to an application programming interface (API) designed to interact with various compute accelerator architectures. In at least one embodiment, a oneAPI programming model utilizes a DPC++ programming language. In at least one embodiment, a DPC++ programming language refers to a high-level language for data parallel programming productivity. In at least one embodiment, a DPC++ programming language is based at least in part on C and/or C++ programming languages. In at least one embodiment, a oneAPI programming model is a programming model such as those developed by Intel Corporation of Santa Clara, CA.

[0671] In at least one embodiment, oneAPI and/or oneAPI programming model is utilized to interact with various accelerator, GPU, processor, and/or variations thereof, architectures. In at least one embodiment, oneAPI includes a set of libraries that implement various functionalities. In at least one embodiment, oneAPI includes at least a oneAPI DPC++ library, a oneAPI math kernel library, a oneAPI data analytics library, a oneAPI deep neural network library, a oneAPI collective communications library, a oneAPI threading building blocks library, a oneAPI video processing library, and/or variations thereof.

[0672] In at least one embodiment, a oneAPI DPC++ library, also referred to as oneDPL, is a library that implements algorithms and functions to accelerate DPC++ kernel programming. In at least one embodiment, oneDPL implements one or more standard template library (STL) functions. In at least one embodiment, oneDPL implements one or more parallel STL functions. In at least one embodiment, oneDPL provides a set of library classes and functions such as parallel algorithms, iterators, function object classes, range-based API, and/or variations thereof. In at least one embodiment, oneDPL implements one or more classes and/or functions of a C++ standard library. In at least one embodiment, oneDPL implements one or more random number generator functions.

[0673] In at least one embodiment, a oneAPI math kernel library, also referred to as oneMKL, is a library that implements various optimized and parallelized routines for various mathematical functions and/or operations. In at least one embodiment, oneMKL implements one or more basic linear algebra subprograms (BLAS) and/or linear algebra package (LAPACK) dense linear algebra routines. In at least one embodiment, oneMKL implements one or more sparse BLAS linear algebra routines. In at least one embodiment, oneMKL implements one or more random number generators (RNGs). In at least one embodiment, oneMKL implements one or more vector mathematics (VM) routines for mathematical operations on vectors. In at least one embodiment, oneMKL implements one or more Fast Fourier Transform (FFT) functions.

[0674] In at least one embodiment, a oneAPI data analytics library, also referred to as oneDAL, is a library that implements various data analysis applications and distributed computations. In at least one embodiment, oneDAL implements various algorithms for preprocessing, transformation, analysis, modeling, validation, and decision making for data analytics, in batch, online, and distributed processing modes of computation. In at least one embodiment, oneDAL implements various C++ and/or Java APIs and various connectors to one or more data sources. In at least one embodiment, oneDAL implements DPC++ API extensions to a traditional C++ interface and enables GPU usage for various algorithms.

[0675] In at least one embodiment, a oneAPI deep neural network library, also referred to as oneDNN, is a library that implements various deep learning functions. In at least one embodiment, oneDNN implements various neural network, machine learning, and deep learning functions, algorithms, and/or variations thereof.

[0676] In at least one embodiment, a oneAPI collective communications library, also referred to as oneCCL, is a library that implements various applications for deep learning and machine learning workloads. In at least one embodiment, oneCCL is built upon lower-level communication middleware, such as message passing interface (MPI) and libfabrics. In at least one embodiment, oneCCL enables a set of deep learning specific optimizations, such as prioritization, persistent operations, out of order executions, and/or variations thereof. In at least one embodiment, oneCCL implements various CPU and GPU functions.

[0677] In at least one embodiment, a oneAPI threading building blocks library, also referred to as oneTBB, is a library that implements various parallelized processes for various applications. In at least one embodiment, oneTBB is utilized for task-based, shared parallel programming on a host. In at least one embodiment, oneTBB implements generic parallel algorithms. In at least one embodiment, oneTBB implements concurrent containers. In at least one embodiment, oneTBB implements a scalable memory allocator. In at least one embodiment, oneTBB implements a work-stealing task scheduler. In at least one embodiment, oneTBB implements low-level synchronization primitives. In at least one embodiment, oneTBB is compiler-independent and usable on various processors, such as GPUs, PPU, CPUs, and/or variations thereof.

[0678] In at least one embodiment, a oneAPI video processing library, also referred to as oneVPL, is a library that is utilized for accelerating video processing in one or more applications. In at least one embodiment, oneVPL implements various video decoding, encoding, and processing functions. In at least one embodiment, oneVPL implements various functions for media pipelines on CPUs, GPUs, and other accelerators. In at least one embodiment, oneVPL implements device discovery and selection in media centric and video analytics workloads. In at least one embodiment, oneVPL implements API primitives for zero-copy buffer sharing.

[0679] In at least one embodiment, a oneAPI programming model utilizes a DPC++ programming language. In at least one embodiment, a DPC++ programming language is a programming language that includes, without limitation, functionally similar versions of CUDA mechanisms to define device code and distinguish between device code and host code. In at least one embodiment, a DPC++ programming language may include a subset of functionality of a CUDA programming language. In at least one embodiment, one or more CUDA programming

model operations are performed using a oneAPI programming model using a DPC++ programming language.

[0680] In at least one embodiment, any application programming interface (API) described herein is compiled into one or more instructions, operations, or any other signal by a compiler, interpreter, or other software tool. In at least one embodiment, compilation comprises generating one or more machine-executable instructions, operations, or other signals from source code. In at least one embodiment, an API compiled into one or more instructions, operations, or other signals, when performed, causes one or more processors such as graphics processors 3000, graphics cores 2000, parallel processor 2200, processor 2500, processor core 2500, or any other logic circuit further described herein to perform one or more computing operations.

[0681] It should be noted that, while example embodiments described herein may relate to a CUDA programming model, techniques described herein can be utilized with any suitable programming model, such as HIP, oneAPI, and/or variations thereof.

[0682] 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.

[0683] 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.

[0684] 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.”

[0685] 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.

[0686] In at least one embodiment, an arithmetic logic unit is a set of combinational logic circuitry that takes one or more inputs to produce a result. In at least one embodiment, an arithmetic logic unit is used by a processor to implement mathematical operation such as addition, subtraction, or multiplication. In at least one embodiment, an arithmetic logic unit is used to implement logical operations such as logical AND/OR or XOR. In at least one embodiment, an arithmetic logic unit is stateless, and made from physical switching components such as semiconductor transistors arranged to form logical gates. In at least one embodiment, an arithmetic logic unit may operate internally as a stateful logic circuit with an associated clock. In at least one embodiment, an arithmetic logic unit may be constructed as an asynchronous logic circuit with an internal state not maintained in an associated register set. In at least one embodiment, an arithmetic logic unit is used by a processor to combine operands stored in one or more registers of the processor and produce an output that can be stored by the processor in another register or a memory location.

[0687] In at least one embodiment, as a result of processing an instruction retrieved by the processor, the processor presents one or more inputs or operands to an arithmetic logic unit, causing the arithmetic logic unit to produce a result based at least in part on an instruction code

provided to inputs of the arithmetic logic unit. In at least one embodiment, the instruction codes provided by the processor to the ALU are based at least in part on the instruction executed by the processor. In at least one embodiment combinational logic in the ALU processes the inputs and produces an output which is placed on a bus within the processor. In at least one embodiment, the processor selects a destination register, memory location, output device, or output storage location on the output bus so that clocking the processor causes the results produced by the ALU to be sent to the desired location.

[0688] In the scope of this application, the term arithmetic logic unit, or ALU, is used to refer to any computational logic circuit that processes operands to produce a result. For example, in the present document, the term ALU can refer to a floating point unit, a DSP, a tensor core, a shader core, a coprocessor, or a CPU.

[0689] In at least one embodiment, one or more components of systems and/or processors disclosed above can communicate with one or more CPUs, ASICs, GPUs, FPGAs, or other hardware, circuitry, or integrated circuit components that include, e.g., an upscaler or upsampler to upscale an image, an image blender or image blender component to blend, mix, or add images together, a sampler to sample an image (e.g., as part of a DSP), a neural network circuit that is configured to perform an upscaler to upscale an image (e.g., from a low resolution image to a high resolution image), or other hardware to modify or generate an image, frame, or video to adjust its resolution, size, or pixels; one or more components of systems and/or processors disclosed above can use components described in this disclosure to perform methods, operations, or instructions that generate or modify an image.

[0690] 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.

[0691] 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.

[0692] 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.

[0693] 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.

[0694] 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.

[0695] 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.

[0696] 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.

[0697] 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.

[0698] 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 select one or more masks to be used with one or more neural networks based, at least in part, on one or more hardware resources to use the one or more neural networks.
2. The processor of claim 1, wherein the one or more circuits are to select the one or more masks based, at least in part, on one or more metrics generated by performing variations of the one or more neural networks using the one or more hardware resources.
3. The processor of claim 1, wherein the one or more circuits are to select the one or more masks to be used with the one or more neural networks to modify one or more data values of the one or more neural networks.
4. The processor of claim 1, wherein the one or more circuits are to prune one or more layers of the one or more neural networks using the one or more masks.
5. The processor of claim 1, wherein the one or more circuits are to select the one or more masks based, at least in part, on a list that correlates one or more metrics with one or more optimization techniques and one or more parameters of the one or more hardware resources.
6. The processor of claim 1, wherein the one or more circuits are to select a mask from the selected one or more masks to be used with the one or more neural networks based, at least in part, on an identification of a hardware resource.
7. The processor of claim 1, wherein the one or more circuits are to modify one or more parameters to be used with the one or more neural networks based, at least in part, on one or more dense versions of the one or more neural networks.
8. A system, comprising:
one or more processors to select one or more masks to be used with one or more neural networks based, at least in part, on one or more hardware resources to use the one or more neural networks.

9. The system of claim 8, wherein the one or more processors are to identify the one or more masks based, at least in part, on performance metrics generated by the one or more hardware resources performing the one or more neural networks.

10. The system of claim 8, wherein the one or more processors are to quantize one or more values of the one or more neural networks based, at least in part, on the one or more masks.

11. The system of claim 8, wherein the one or more processors are to generate one or more sparse neural networks using the selected one or more masks with the one or more neural networks.

12. The system of claim 8, wherein the one or more processors are to select the one or more masks based, at least in part, on performance metrics generated by the one or more hardware resources performing the one or more neural networks that exceed one or more threshold performance metrics.

13. The system of claim 8, wherein the one or more circuits are to select the one or more masks, based, at least in part, on one or more parameters of a hardware resource.

14. The system of claim 8, wherein the one or more circuits are further to train the one or more neural networks using the one or more selected masks.

15. A method, comprising:
selecting one or more masks to be used with one or more neural networks based, at least in part, on one or more hardware resources to use the one or more neural networks.

16. The method of claim 15, wherein the one or more masks is identified based, at least in part, on performance metrics generated by using the one or more hardware resources performing the one or more neural network models modified with a quantization of one or more values.

17. The method of claim 15, wherein the one or more masks is to be used with the one or more neural networks to prune one or more portions of the one or more neural networks.

18. The method of claim 15, further comprising generating one or more sparse neural networks using the one or more masks to reduce one or more layers of the one or more neural networks.

19. The method of claim 15, wherein the one or more neural networks are to be performed by the one or more hardware resources to exceed one or more threshold performance metrics that indicate accuracy and one or more threshold values that indicate inferencing speed.

20. The method of claim 15, wherein the one or more masks are one or more tensors that indicate which nonzero values of the one or more neural networks should be set to zero.

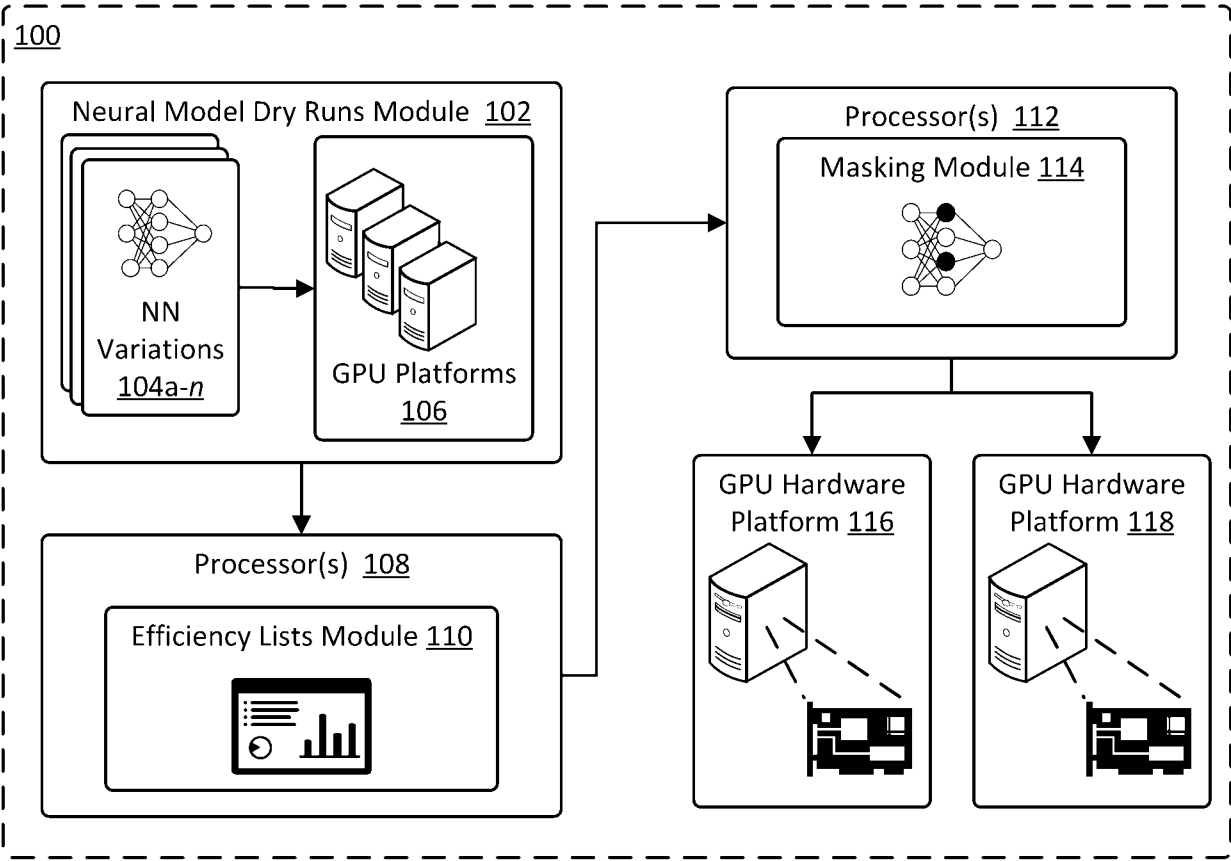


FIG. 1A

130

Layer No	Data Type	Sparse Ratio	Hardware	Speed Up
1	FP32	100%	A100	1.0x
1	FP16	50%	A100	1.2x
1	FP16	100%	A100	1.1x
1	INT8	50%	A100	1.6x
1	INT8	100%	A100	1.5x
2	FP32	100%	A100	1.0x
2	FP16	50%	A100	1.3x
2	FP16	100%	A100	1.15x
2	INT8	50%	A100	1.3x
2	INT8	100%	A100	1.2x

Layer No	Data Type	Sparse Ratio	Hardware	Speed Up
1	FP16	50%	A100, H100	1.2x
1	INT8	50%	A100	1.6x
1	INT8	50%	H100	1.7X
2	FP16	50%	A100, H100	1.3x
2	INT8	50%	A100	1.3x
2	INT8	50%	H100	1.4X

FIG. 1B

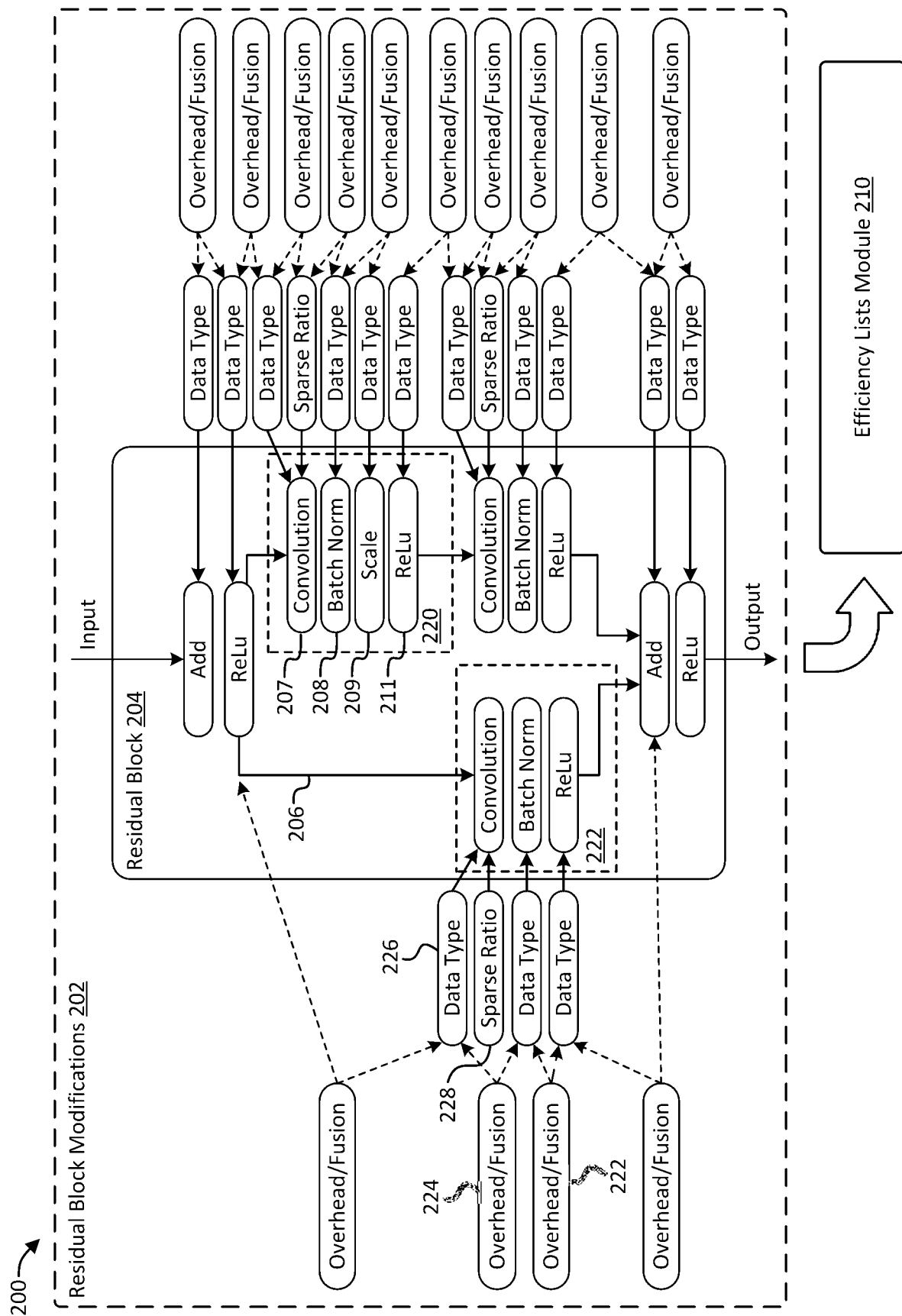


FIG. 2

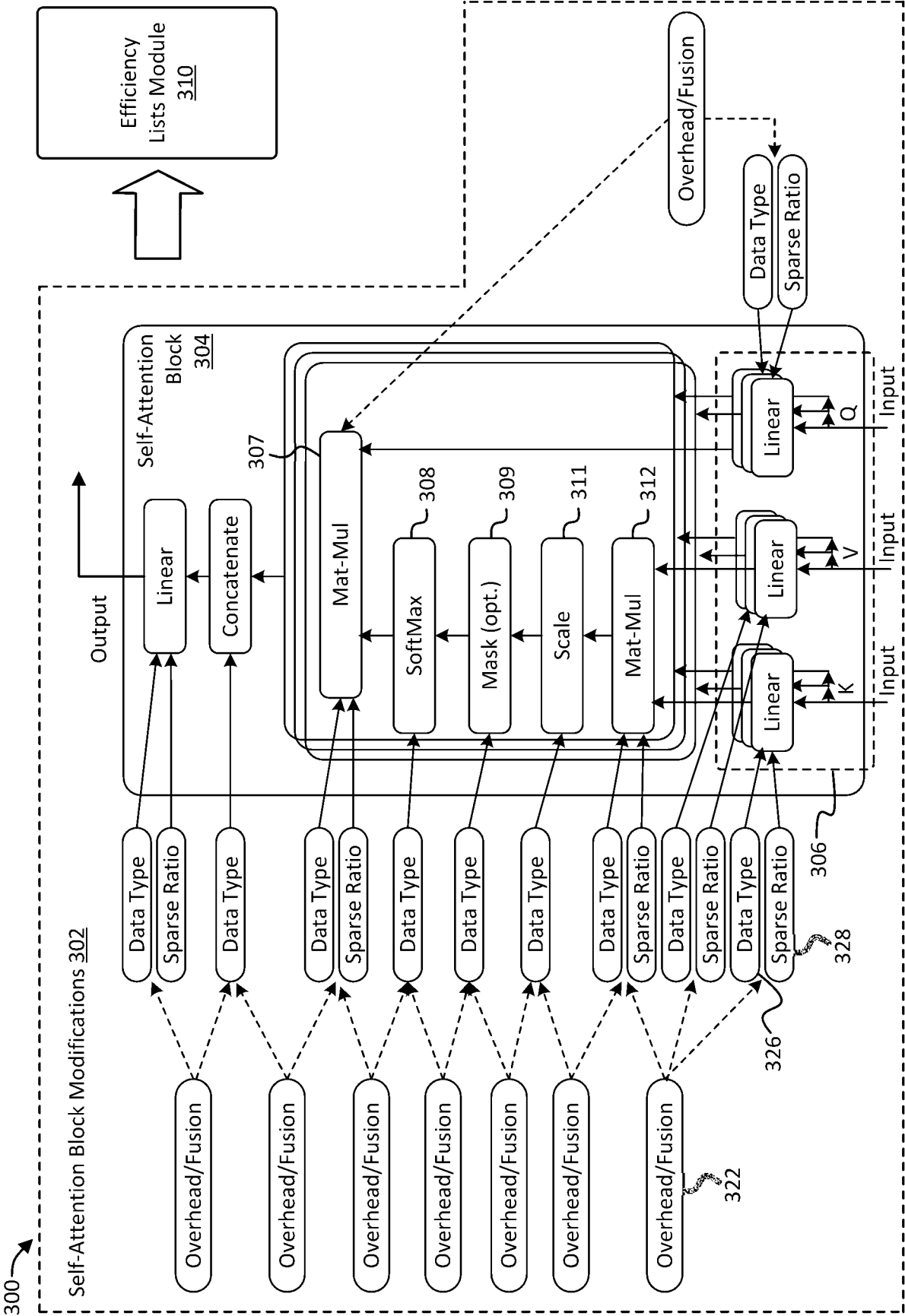


FIG. 3

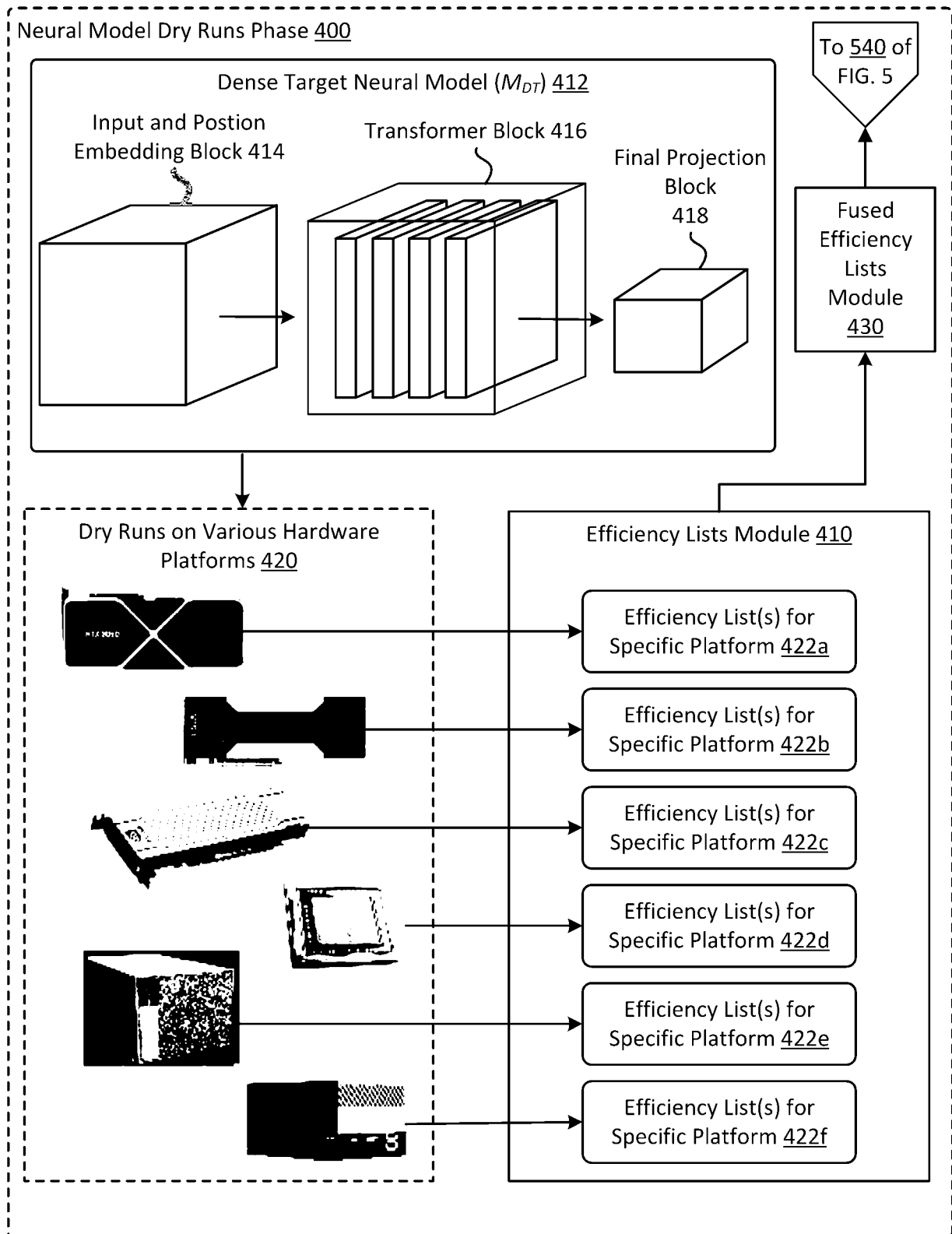


FIG. 4

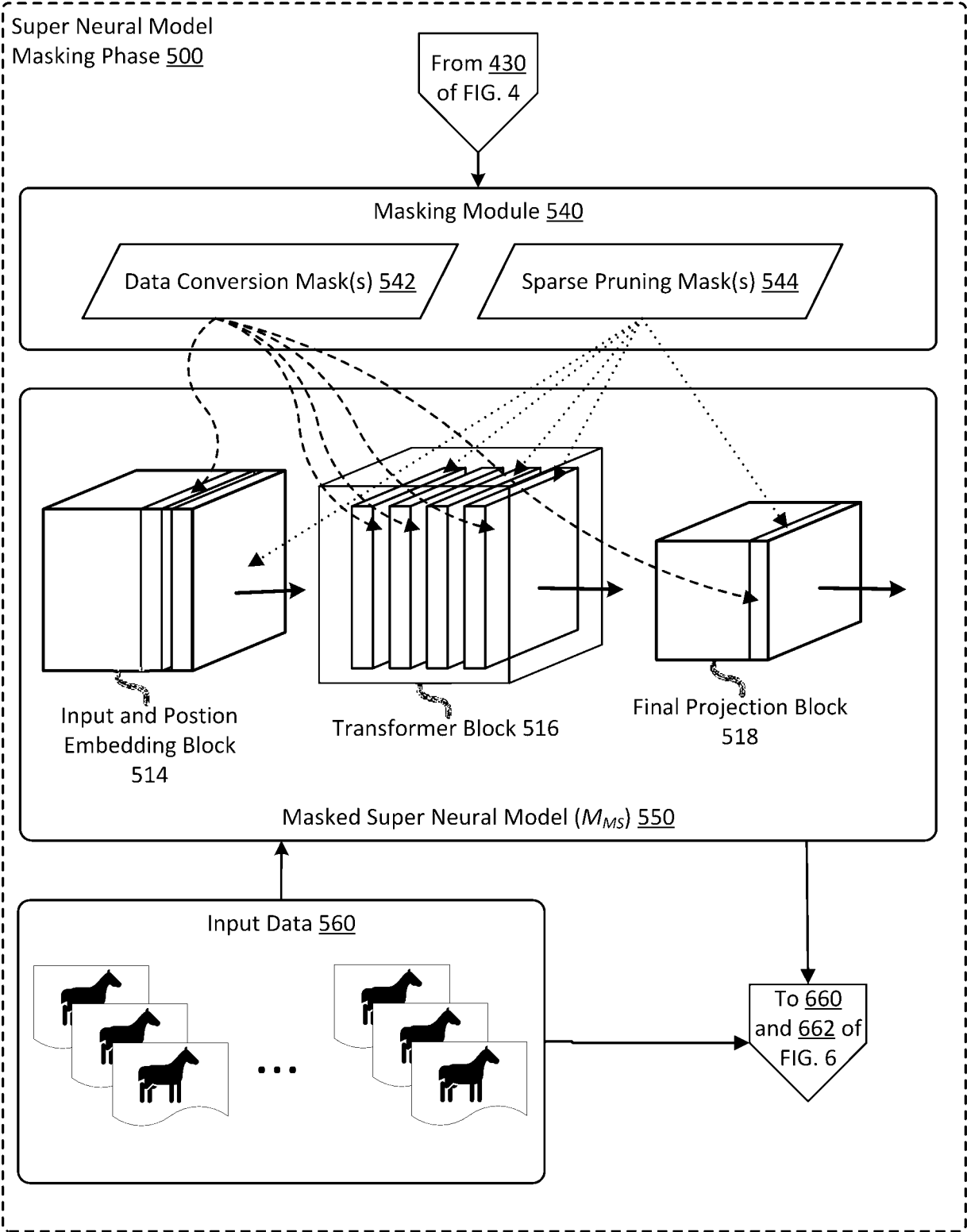


FIG. 5

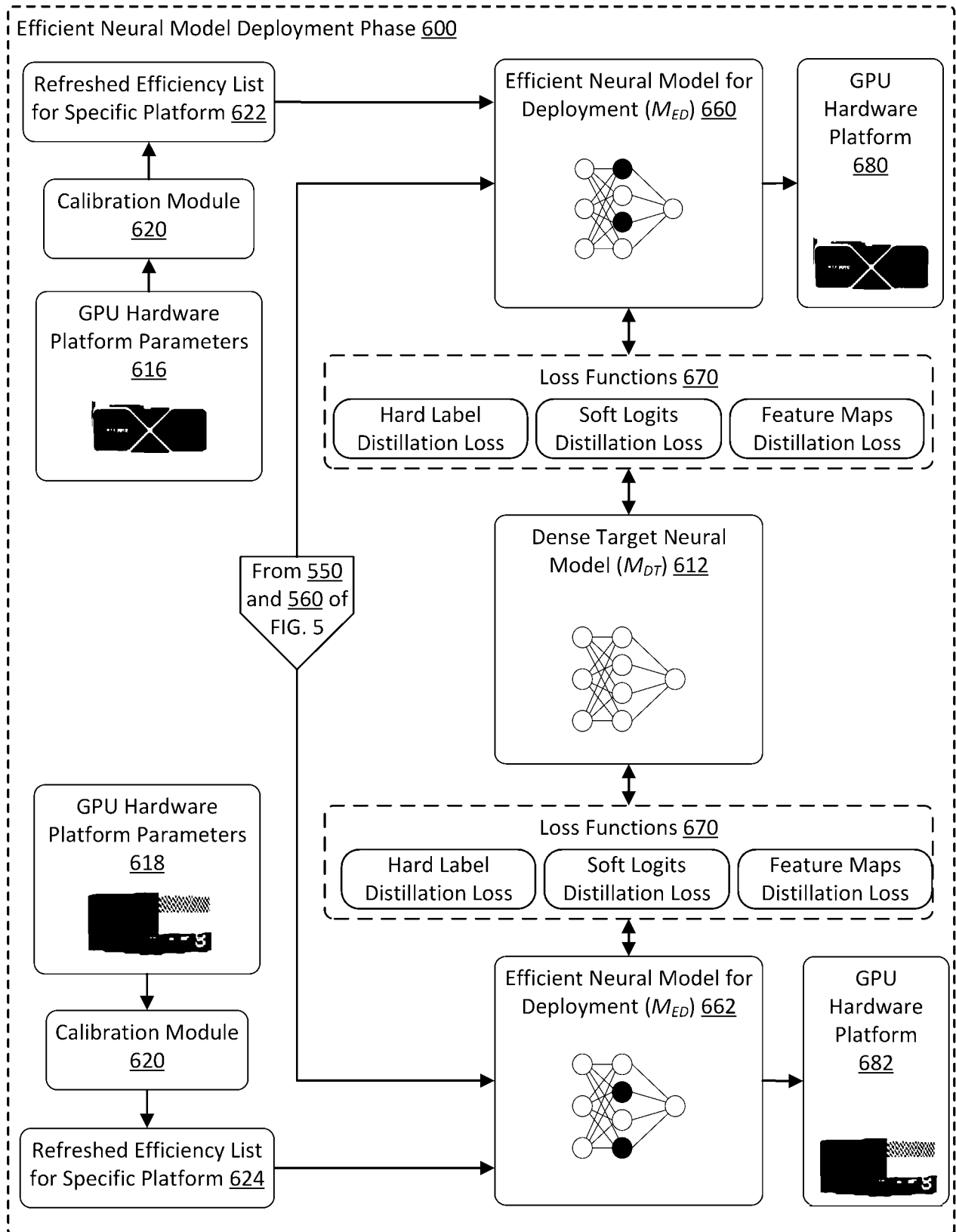
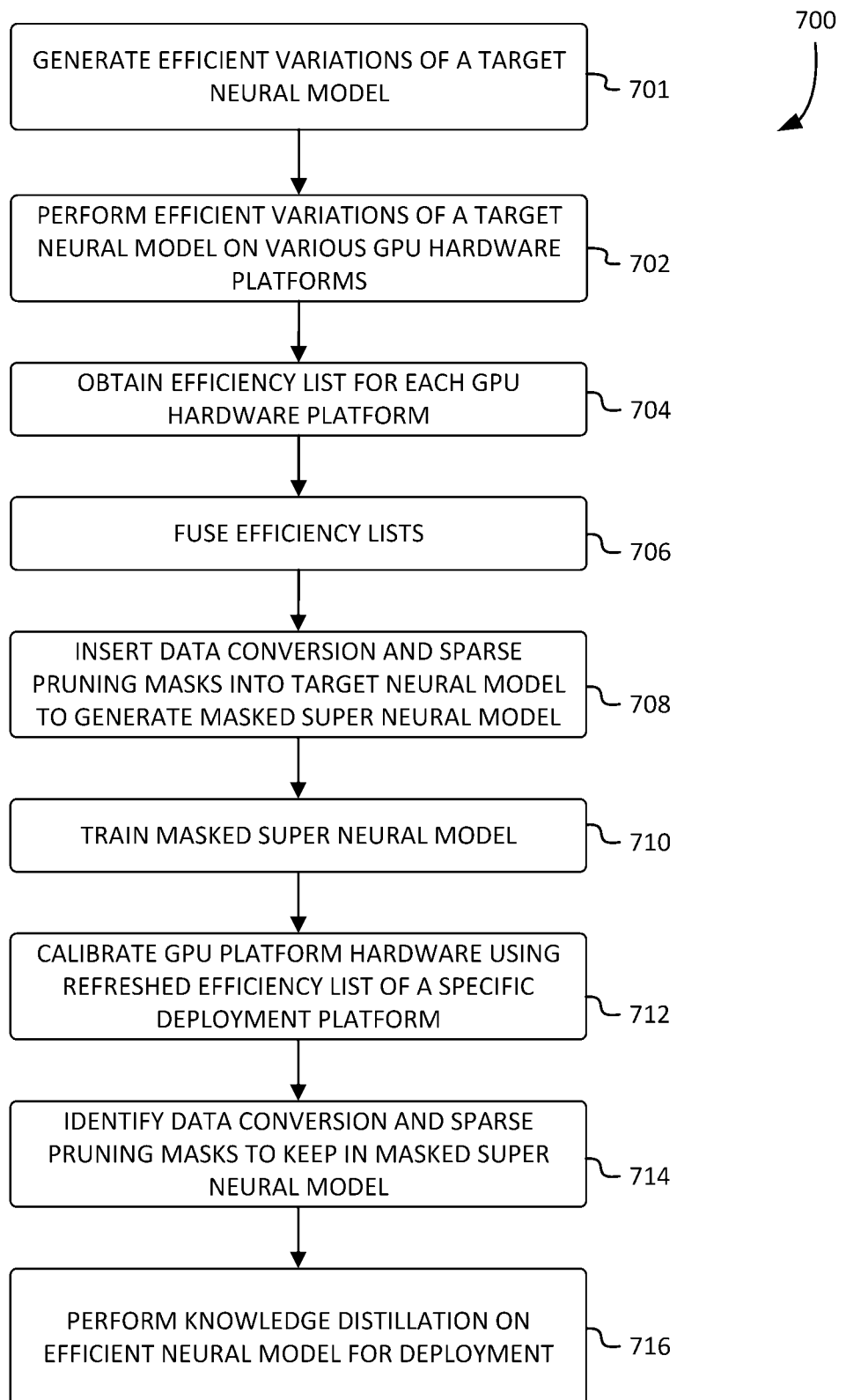
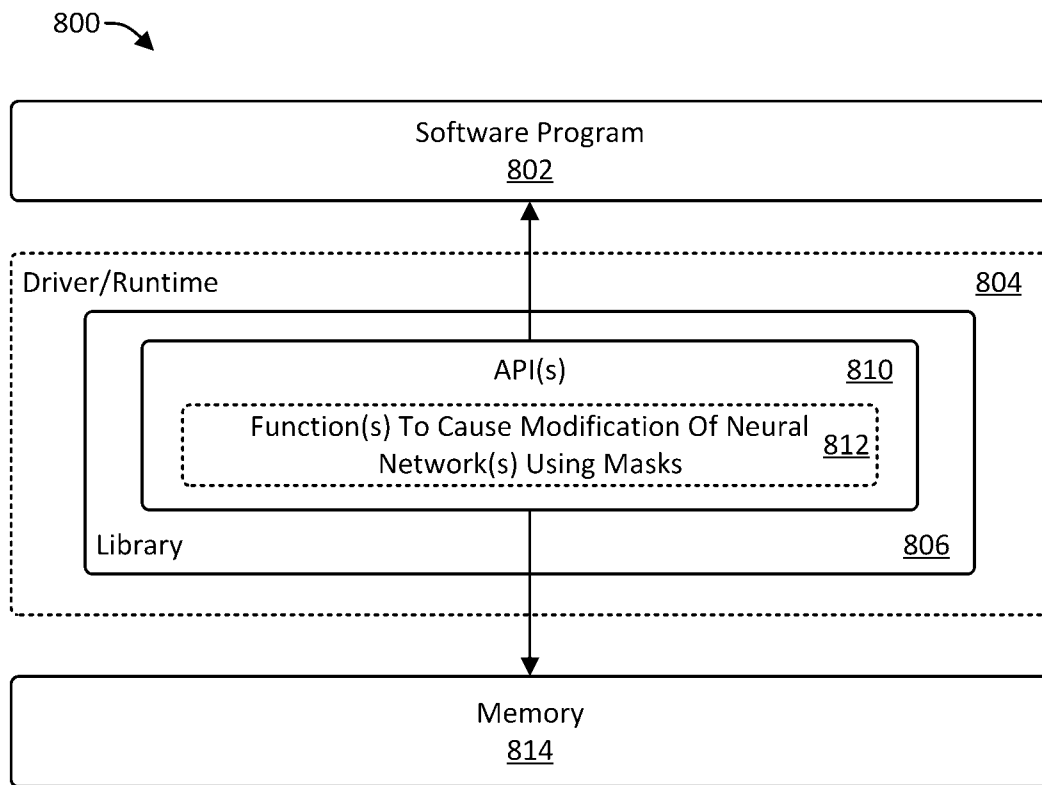
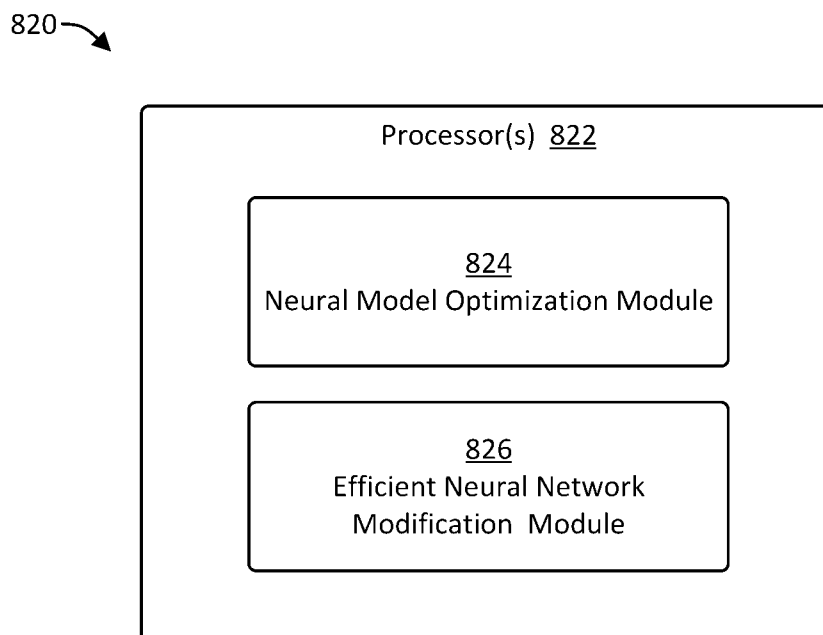
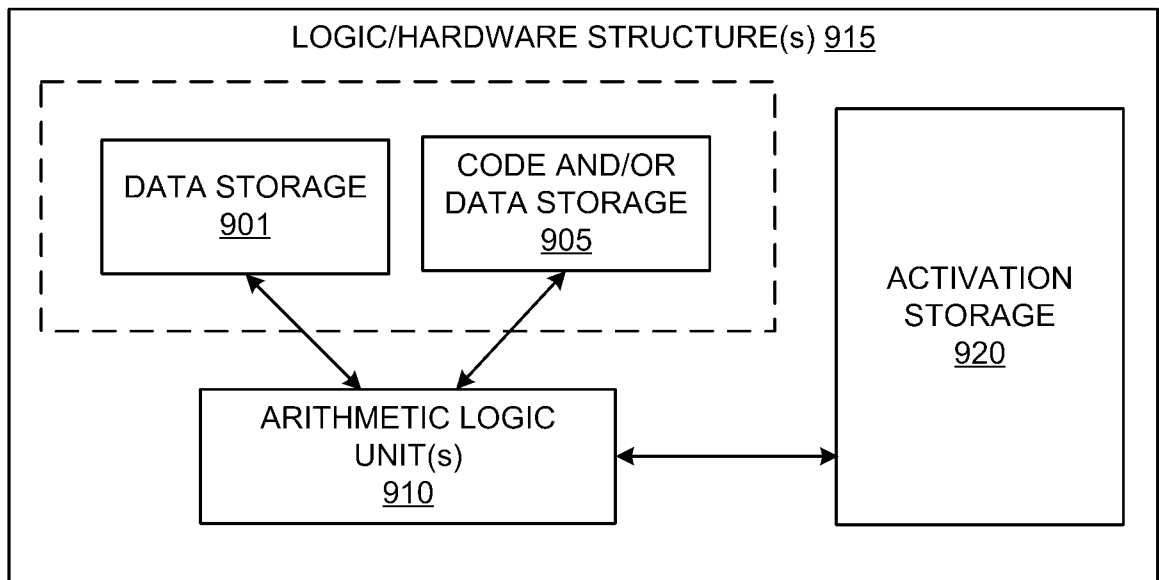
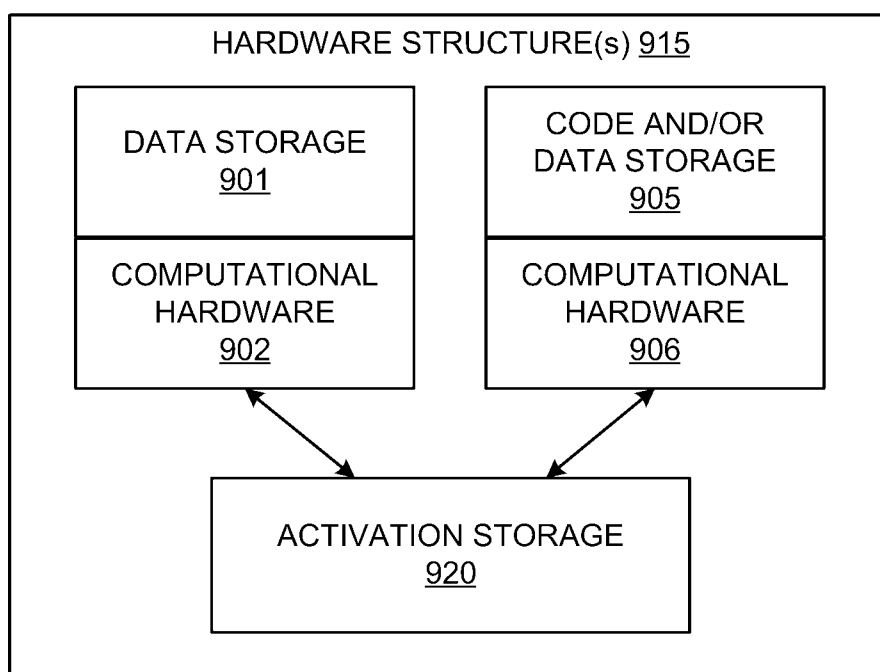


FIG. 6

**FIG. 7**

**FIG. 8A****FIG. 8B**

**FIG. 9A****FIG. 9B**

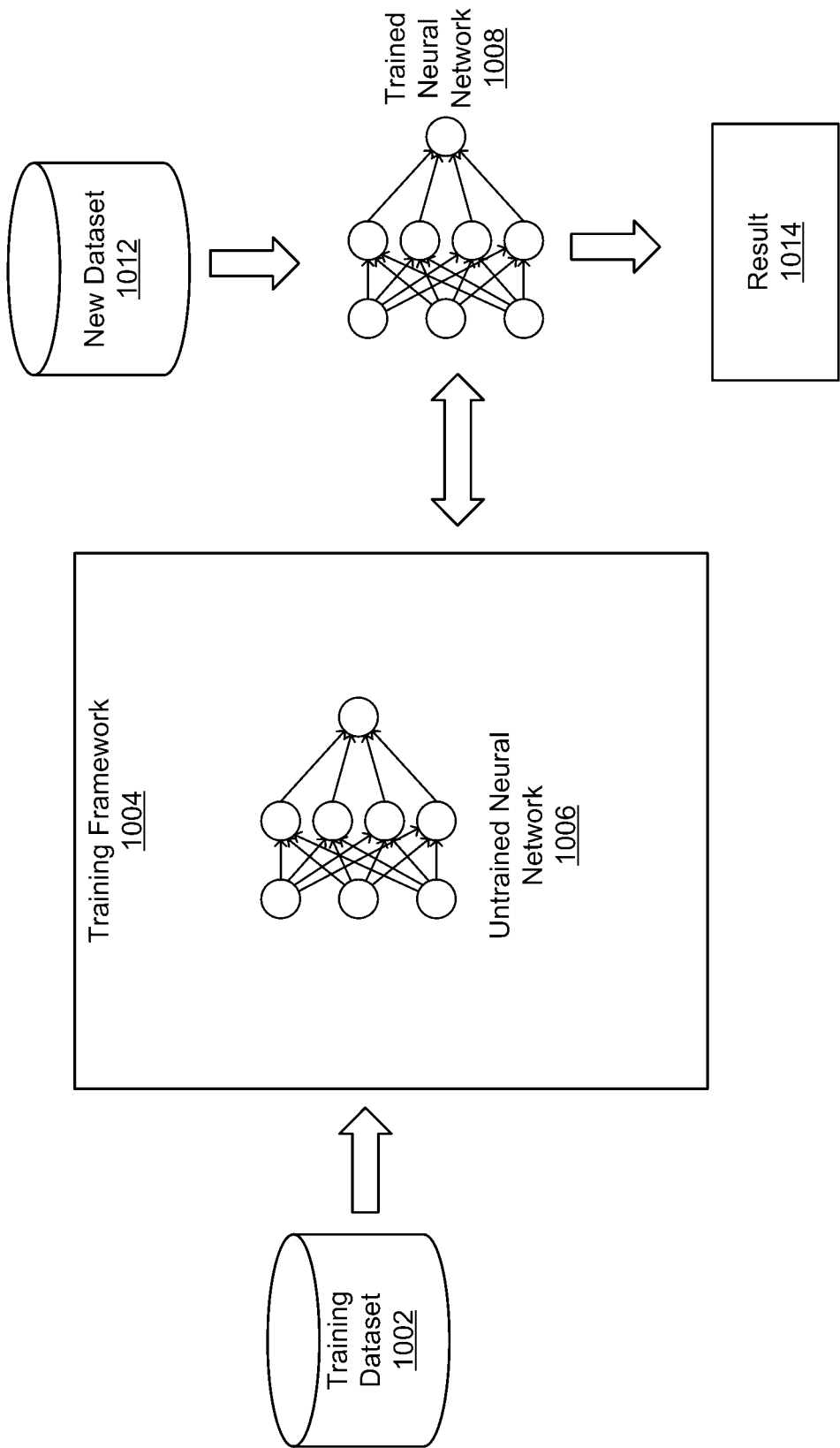


FIG. 10

DATA CENTER
1100

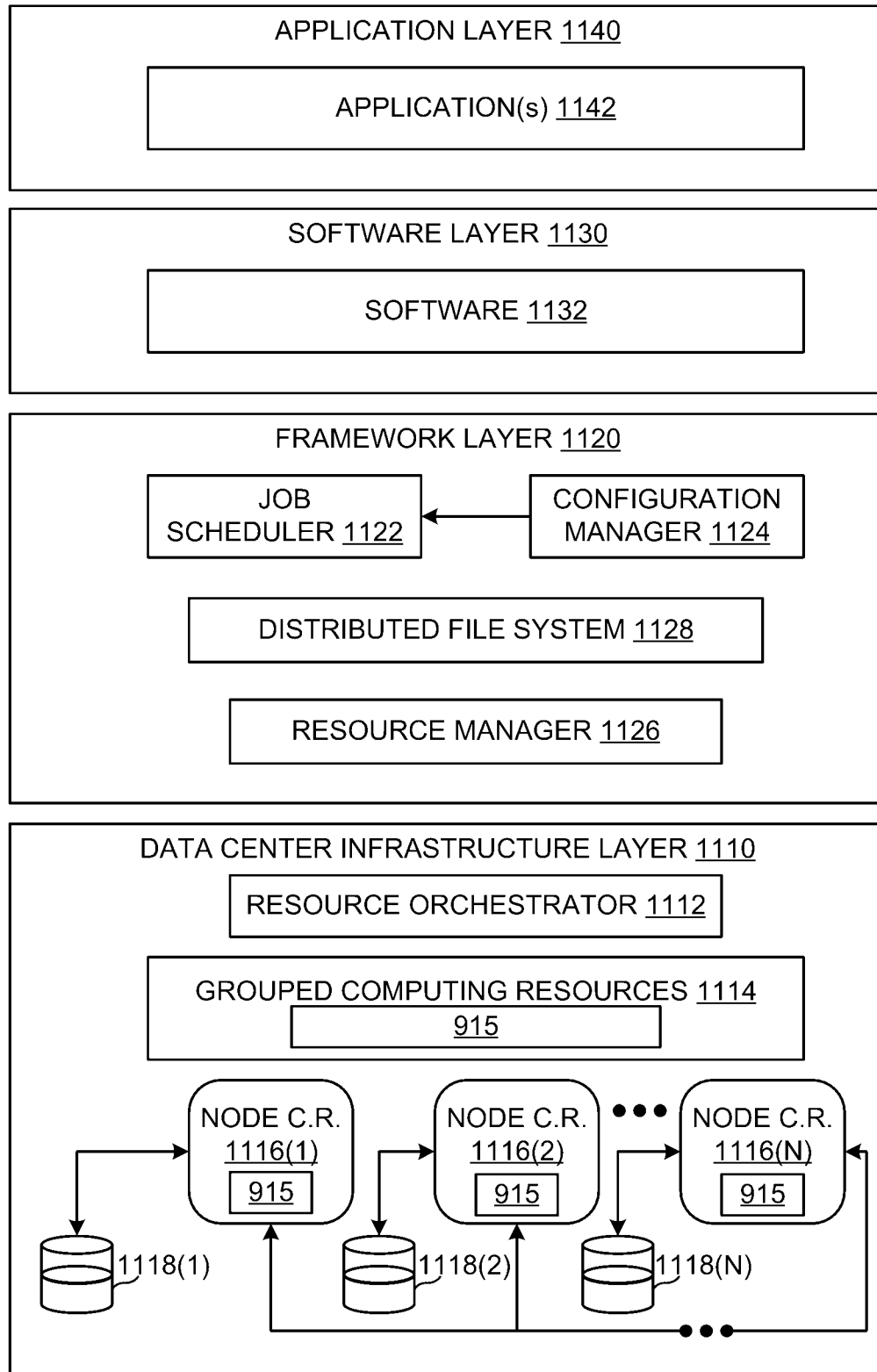


FIG. 11

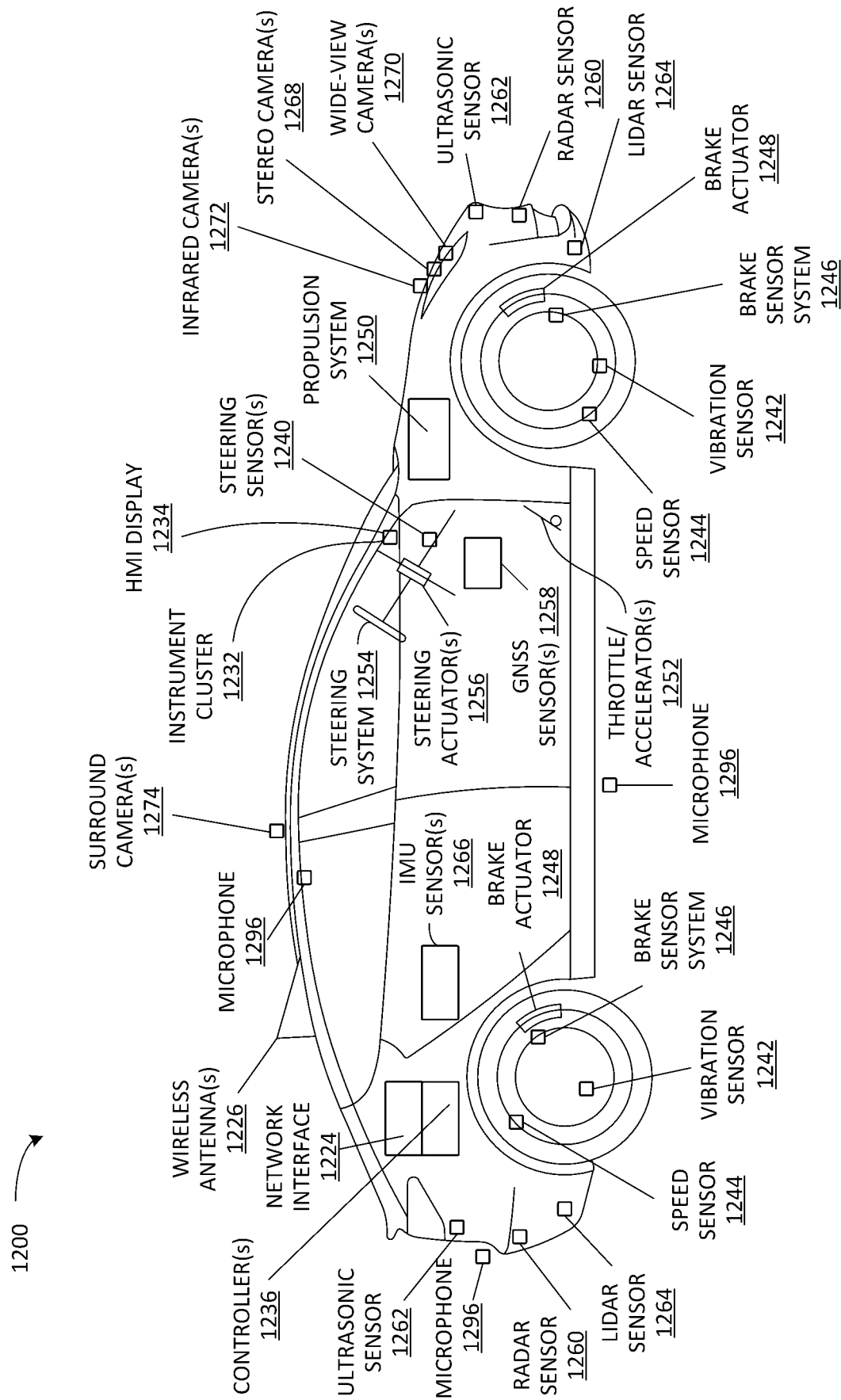


FIG. 12A

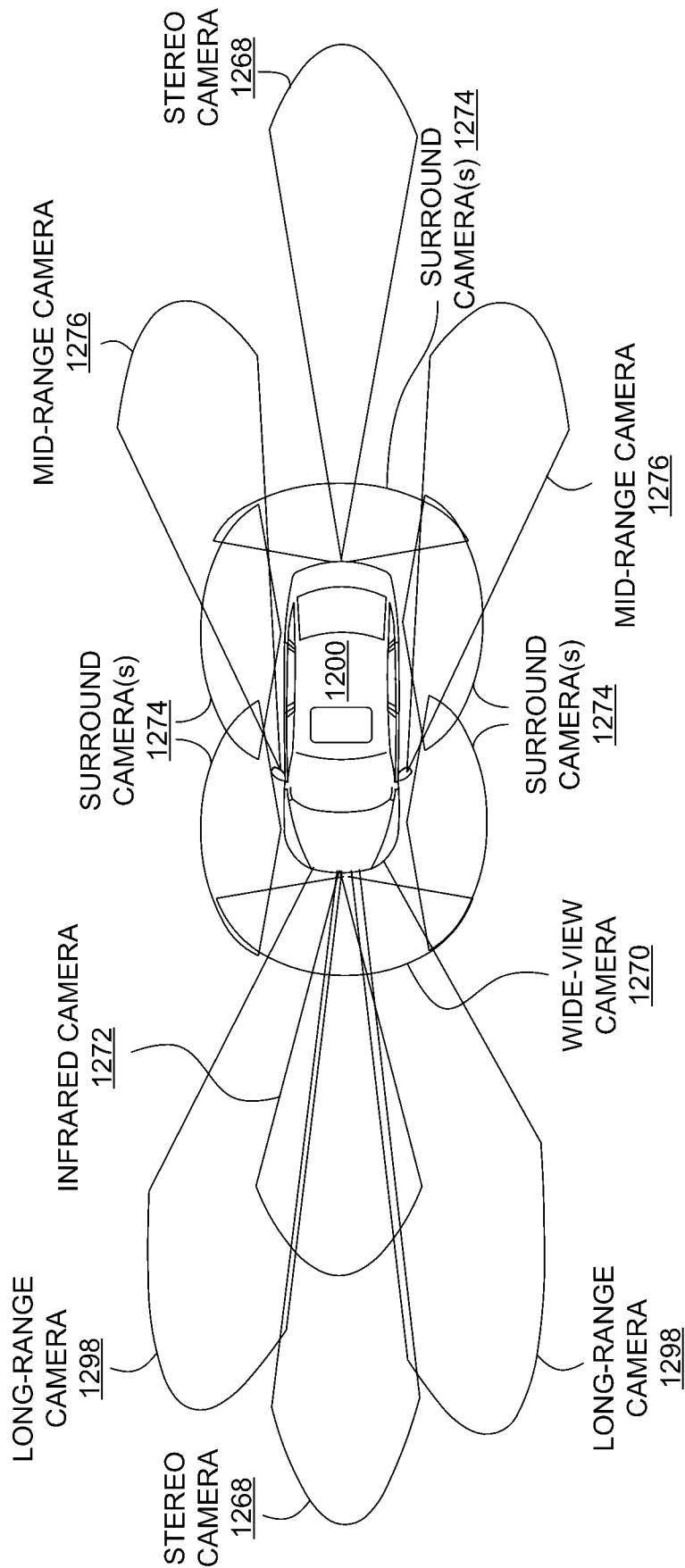


FIG. 12B

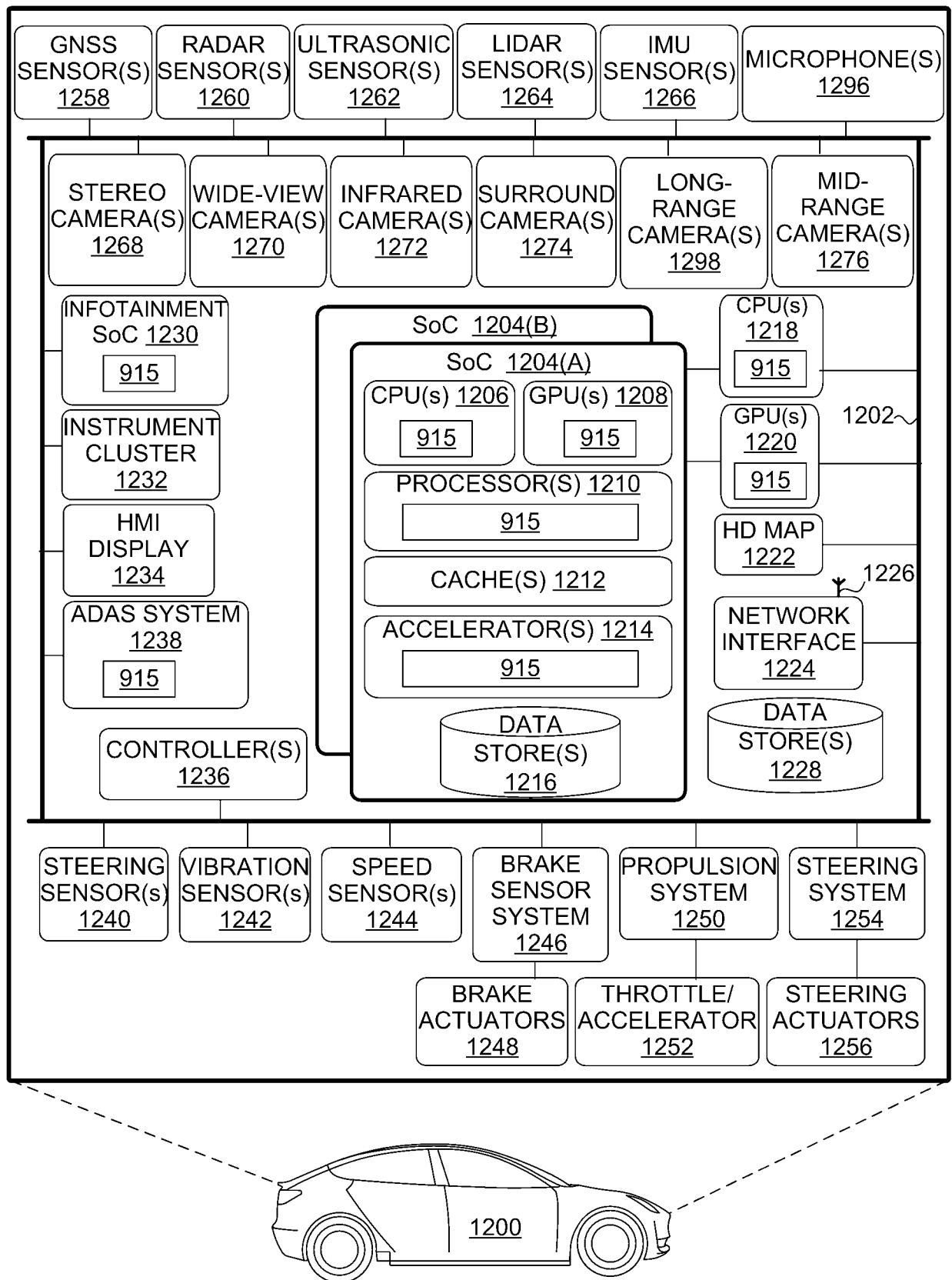
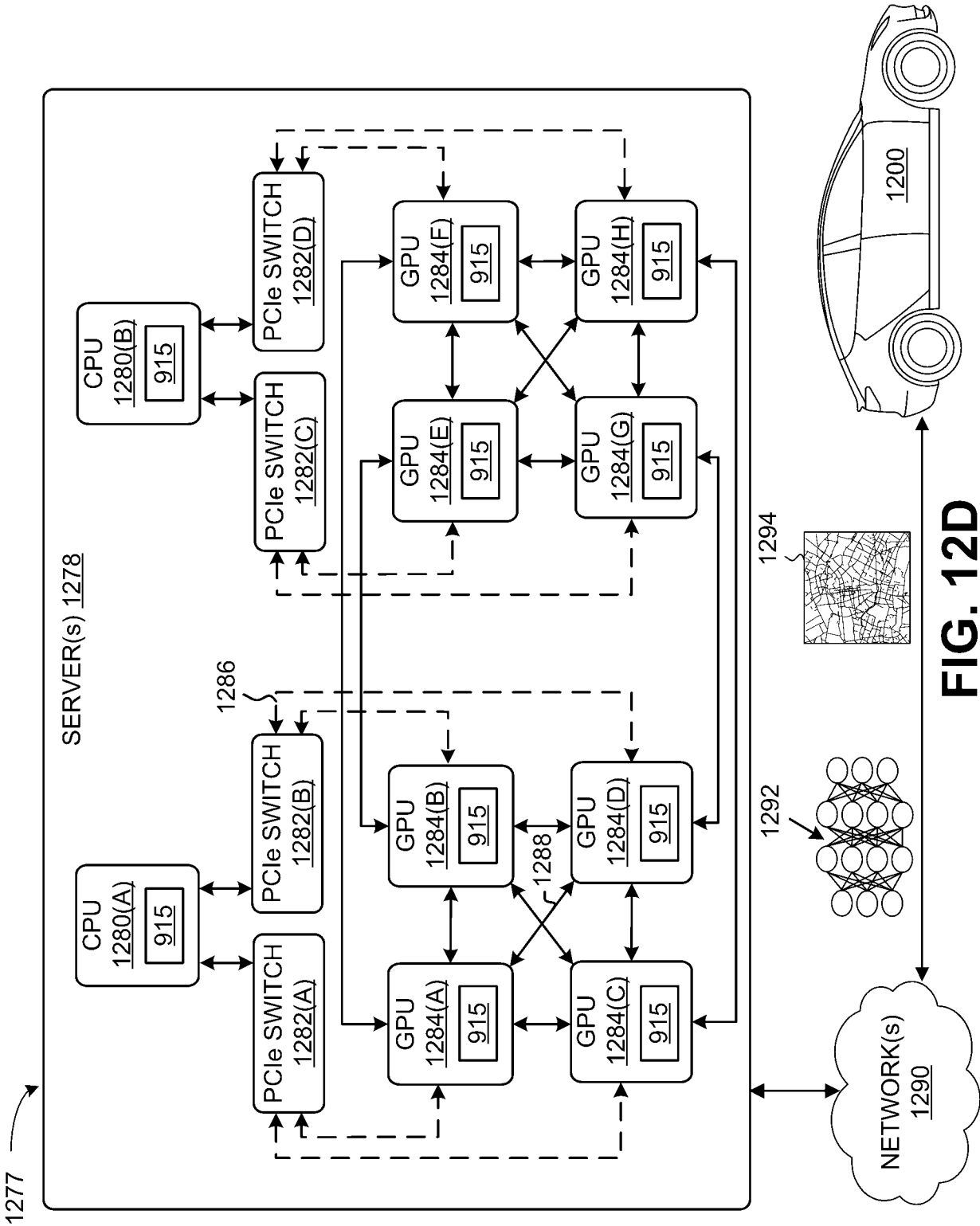
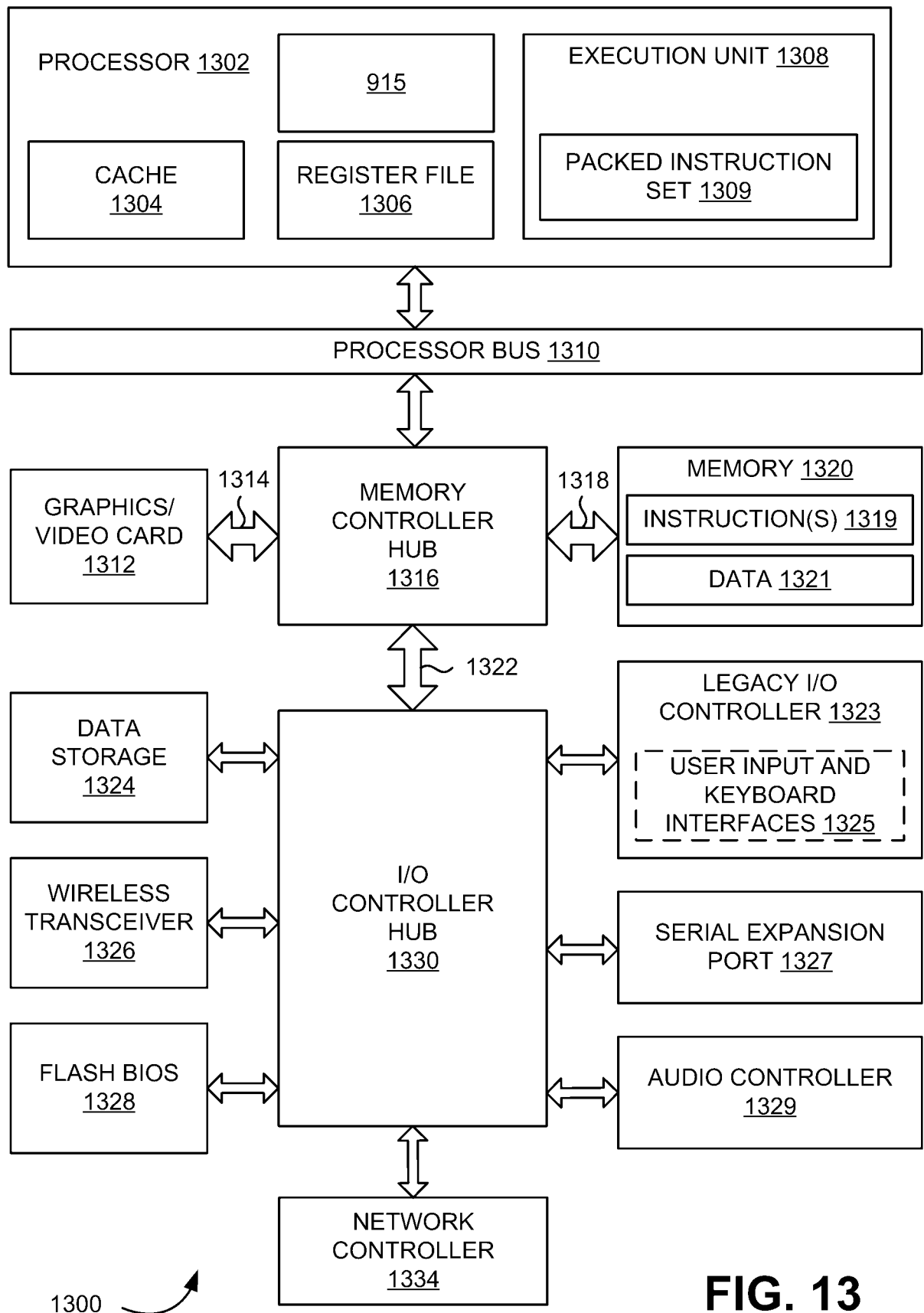


FIG. 12C



**FIG. 13**

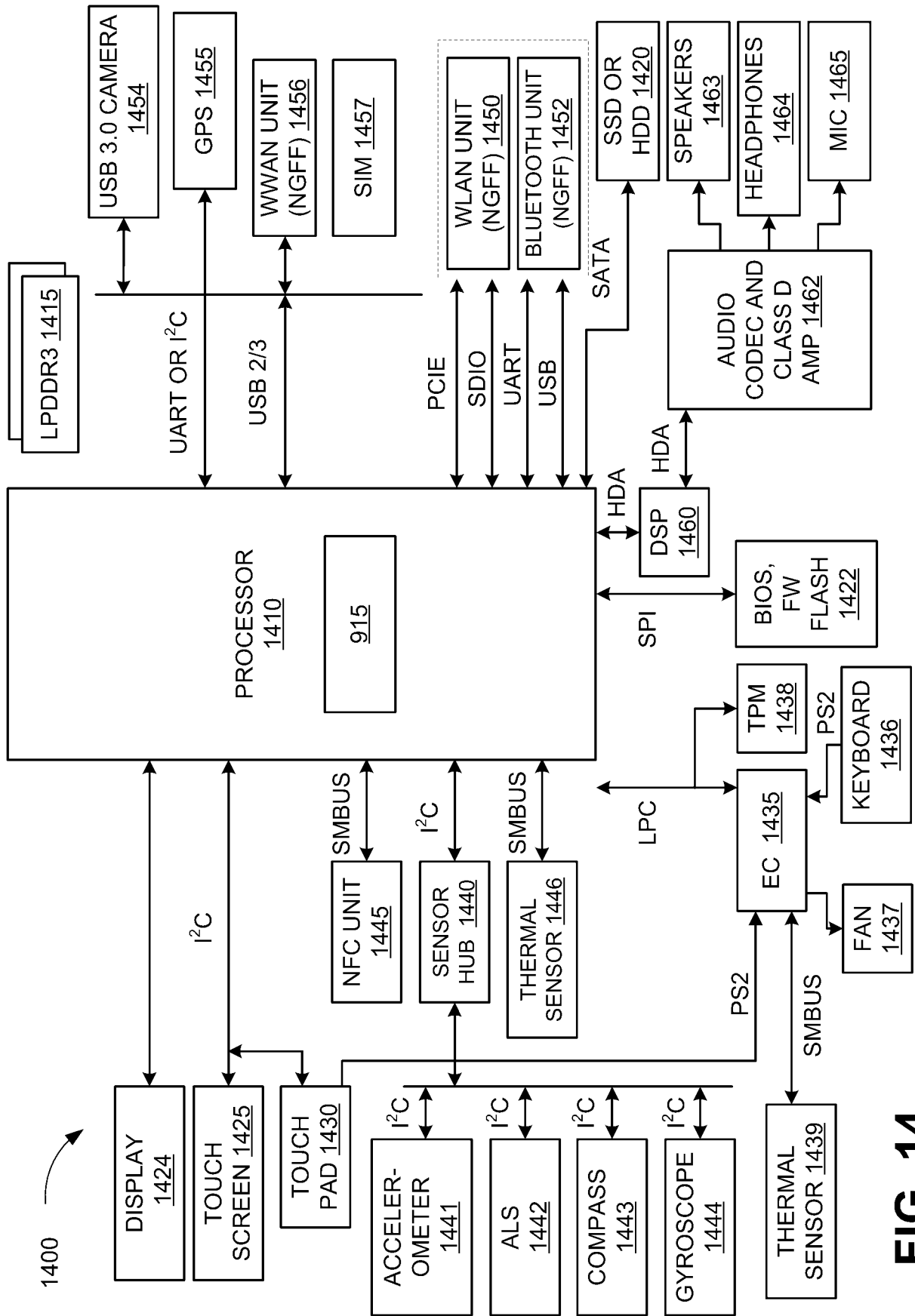
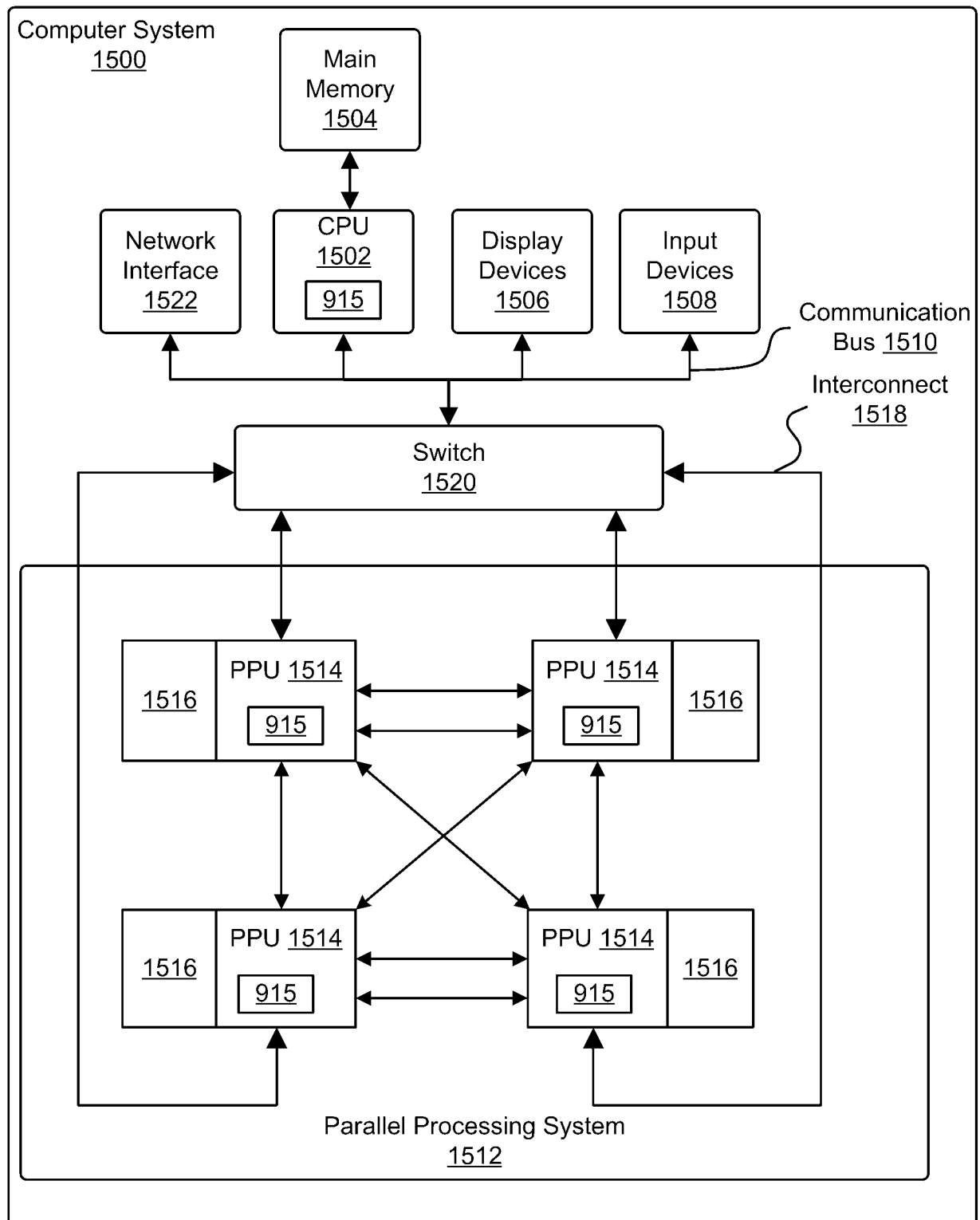


FIG. 14

**FIG. 15**

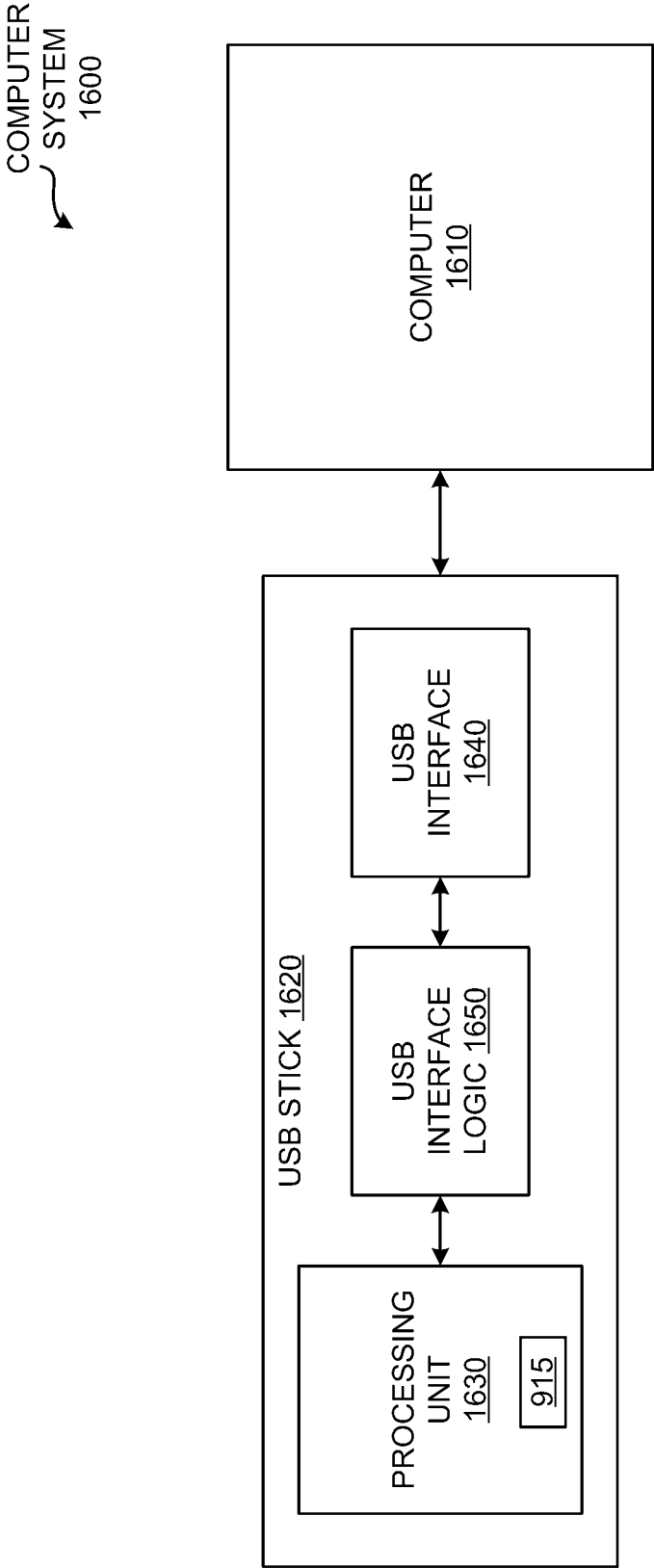


FIG. 16

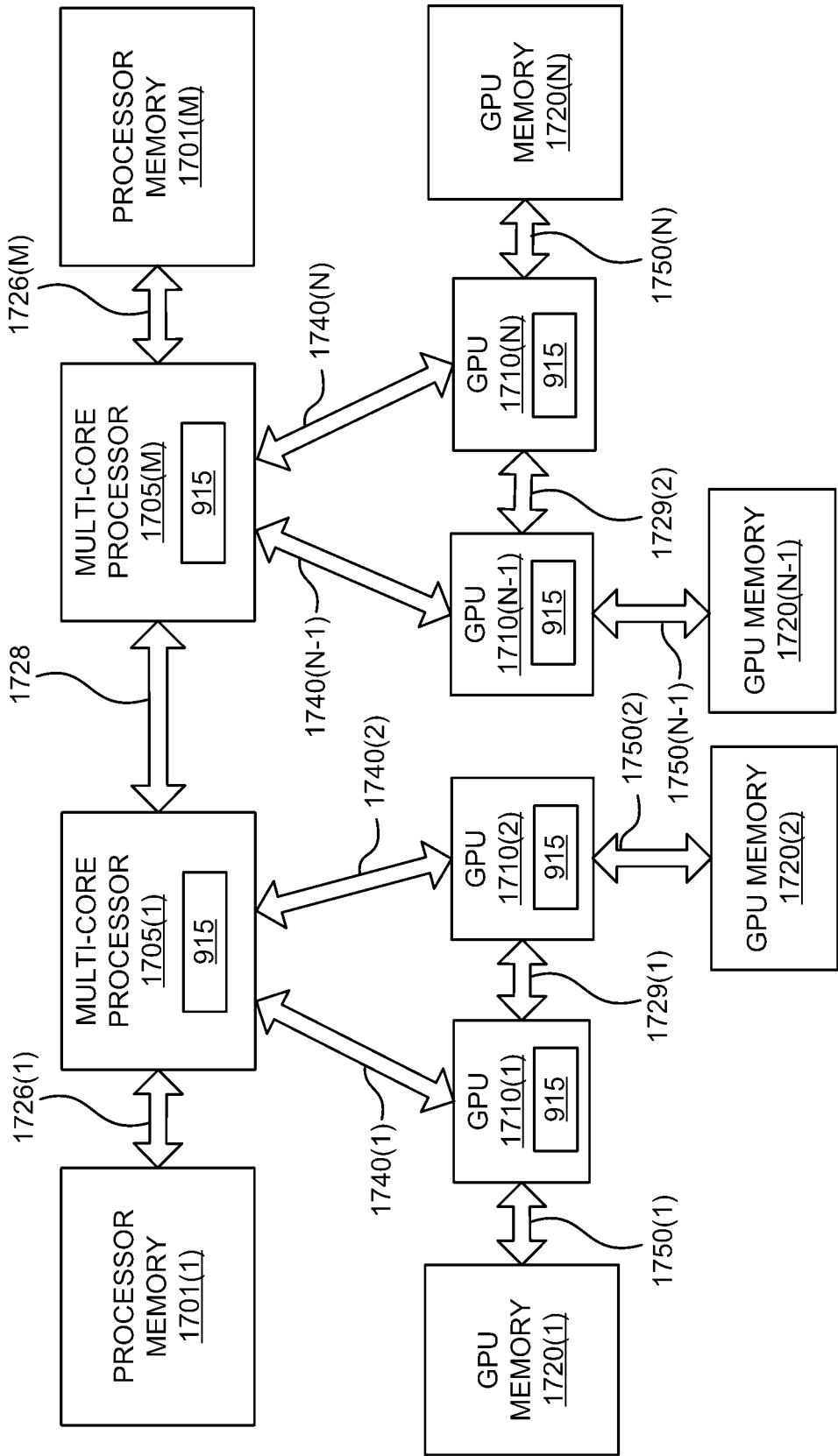


FIG. 17A

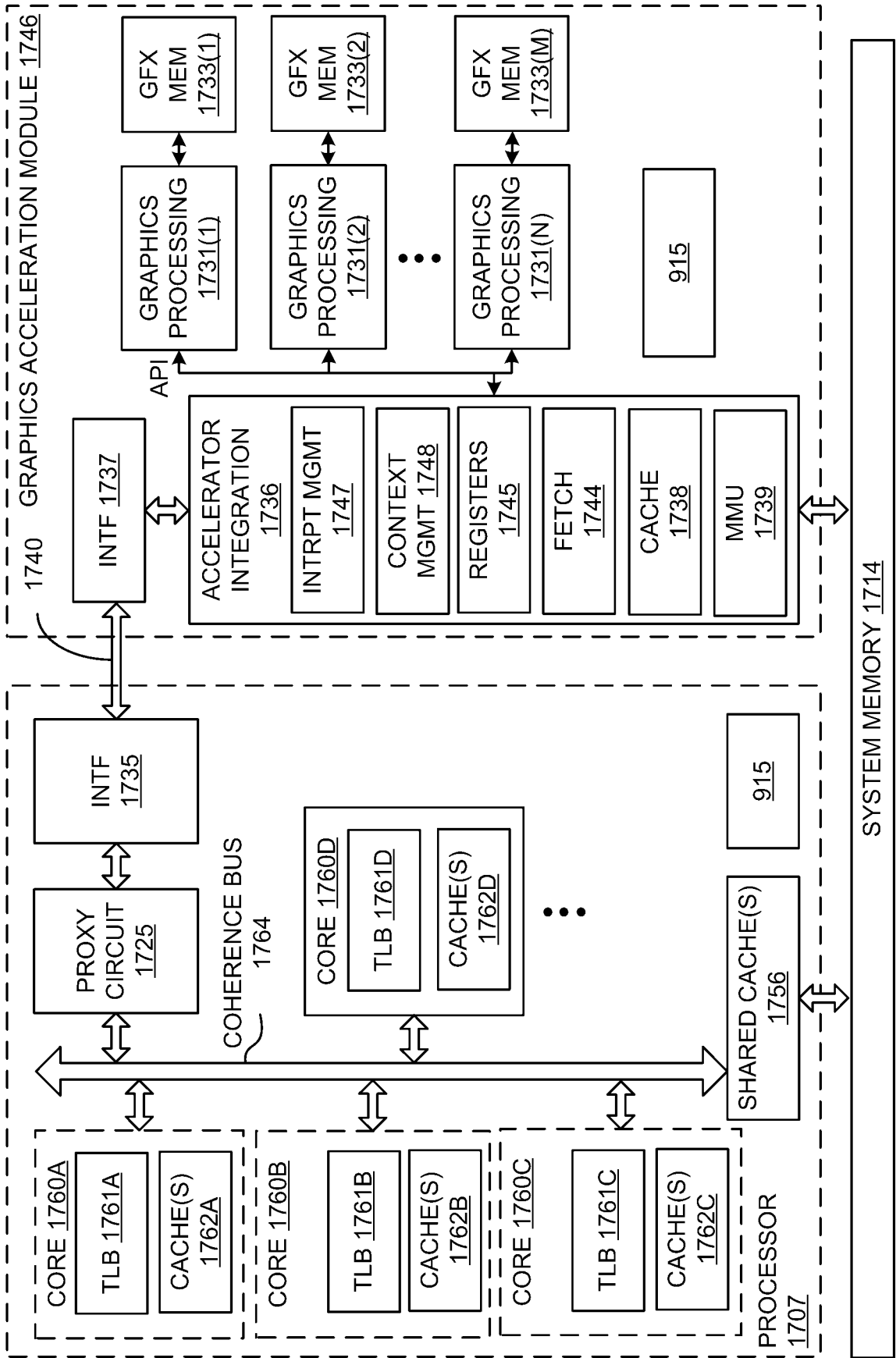


FIG. 17B

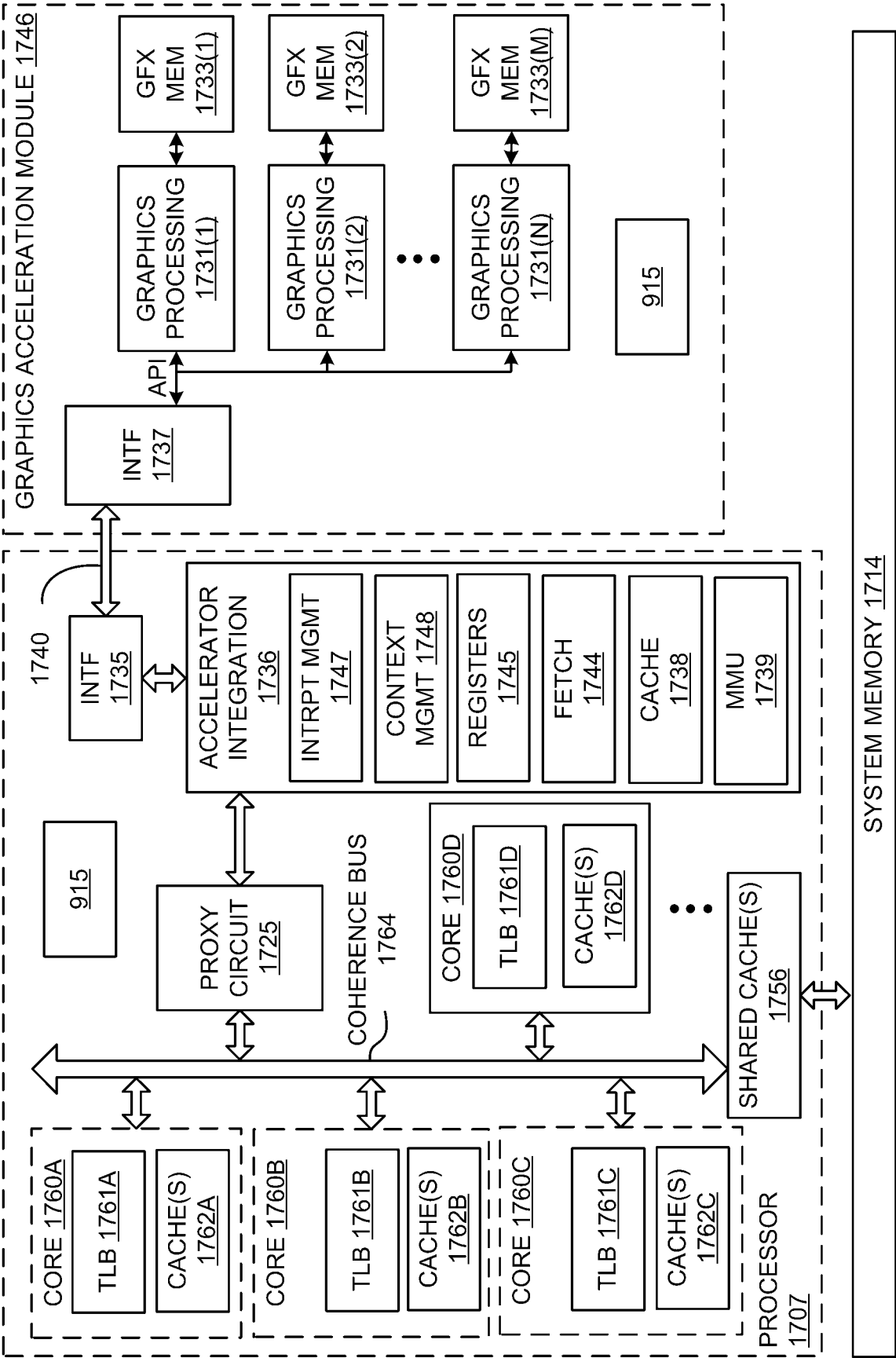
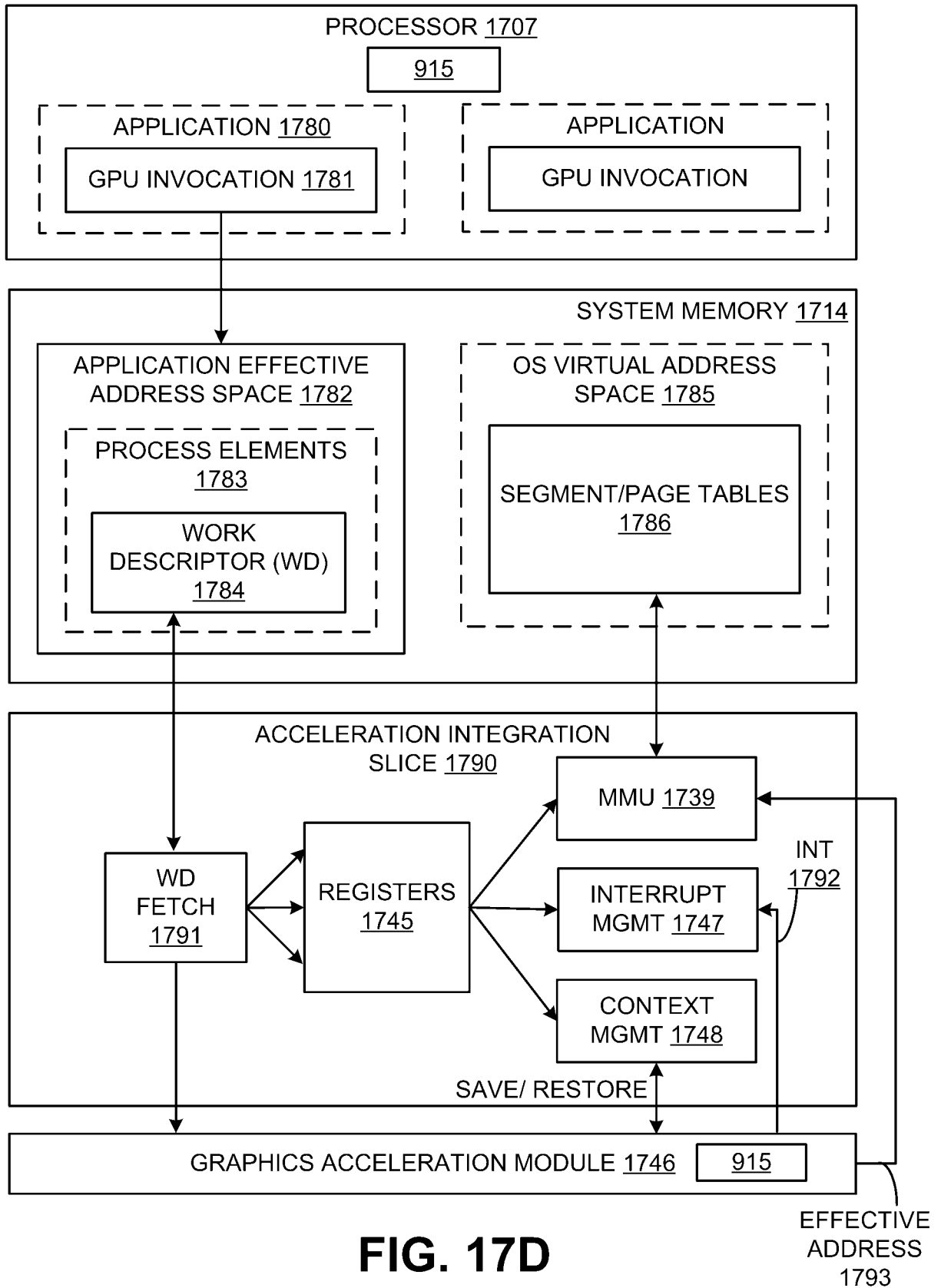


FIG. 17C



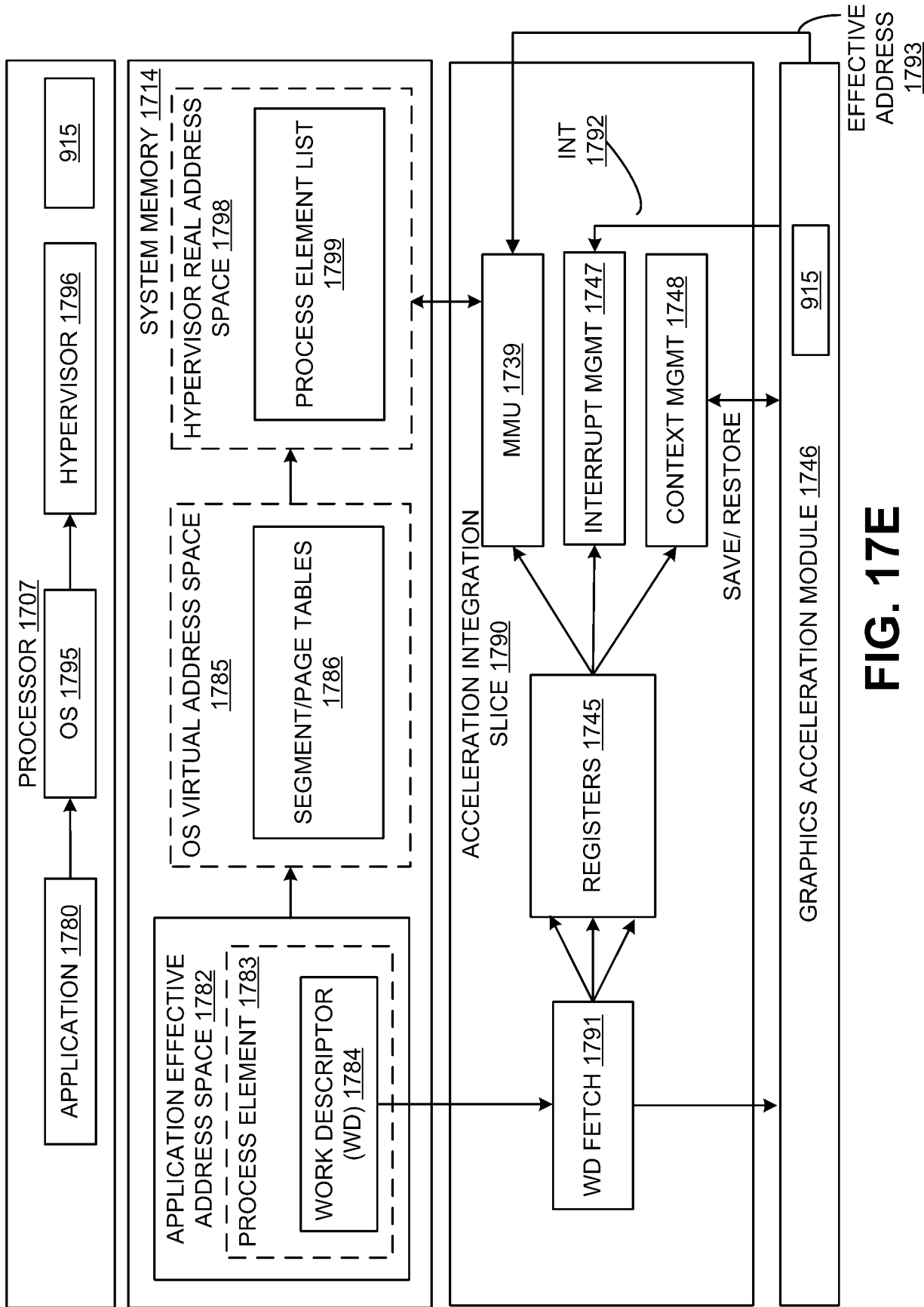


FIG. 17E

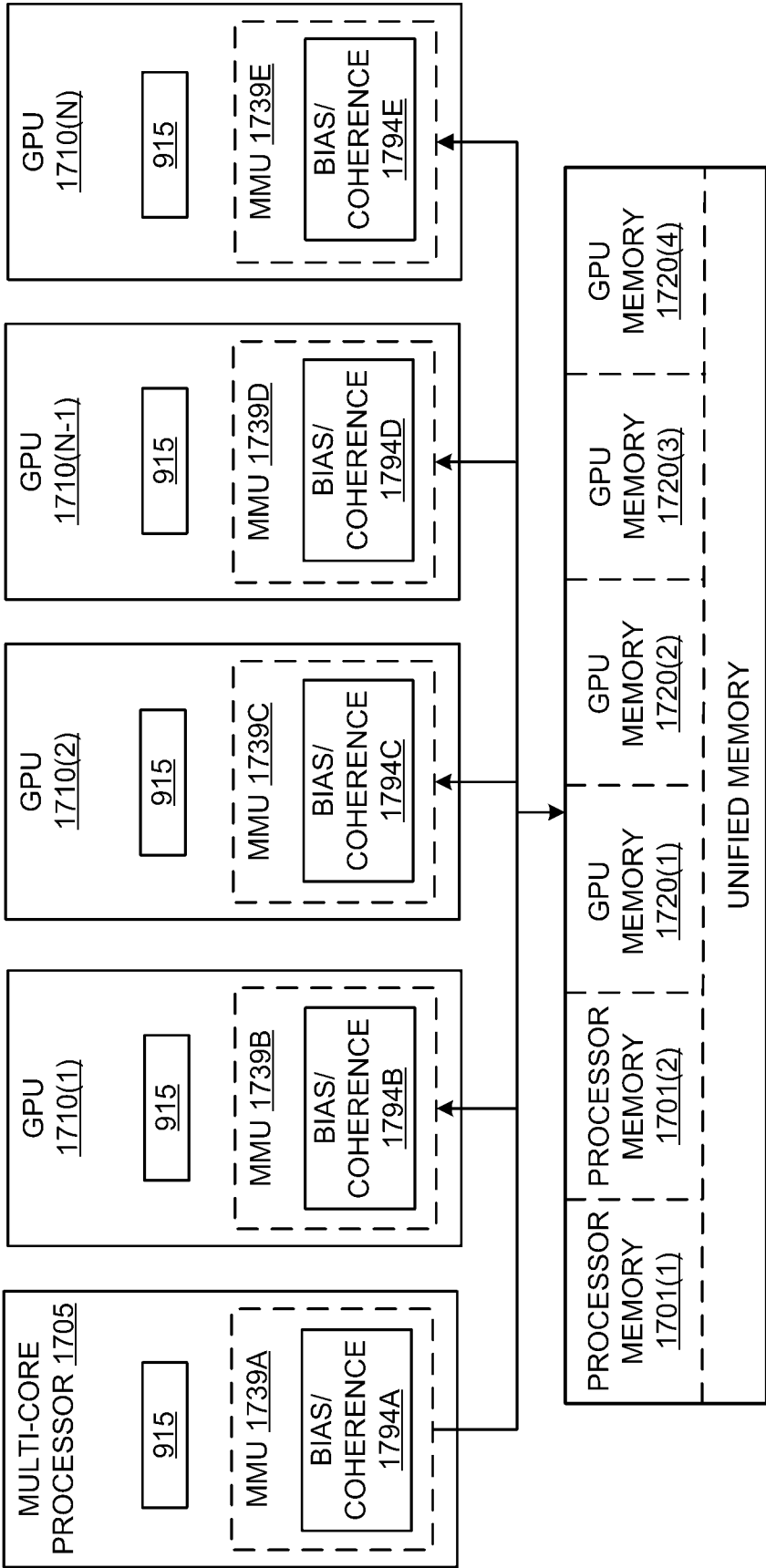


FIG. 17F

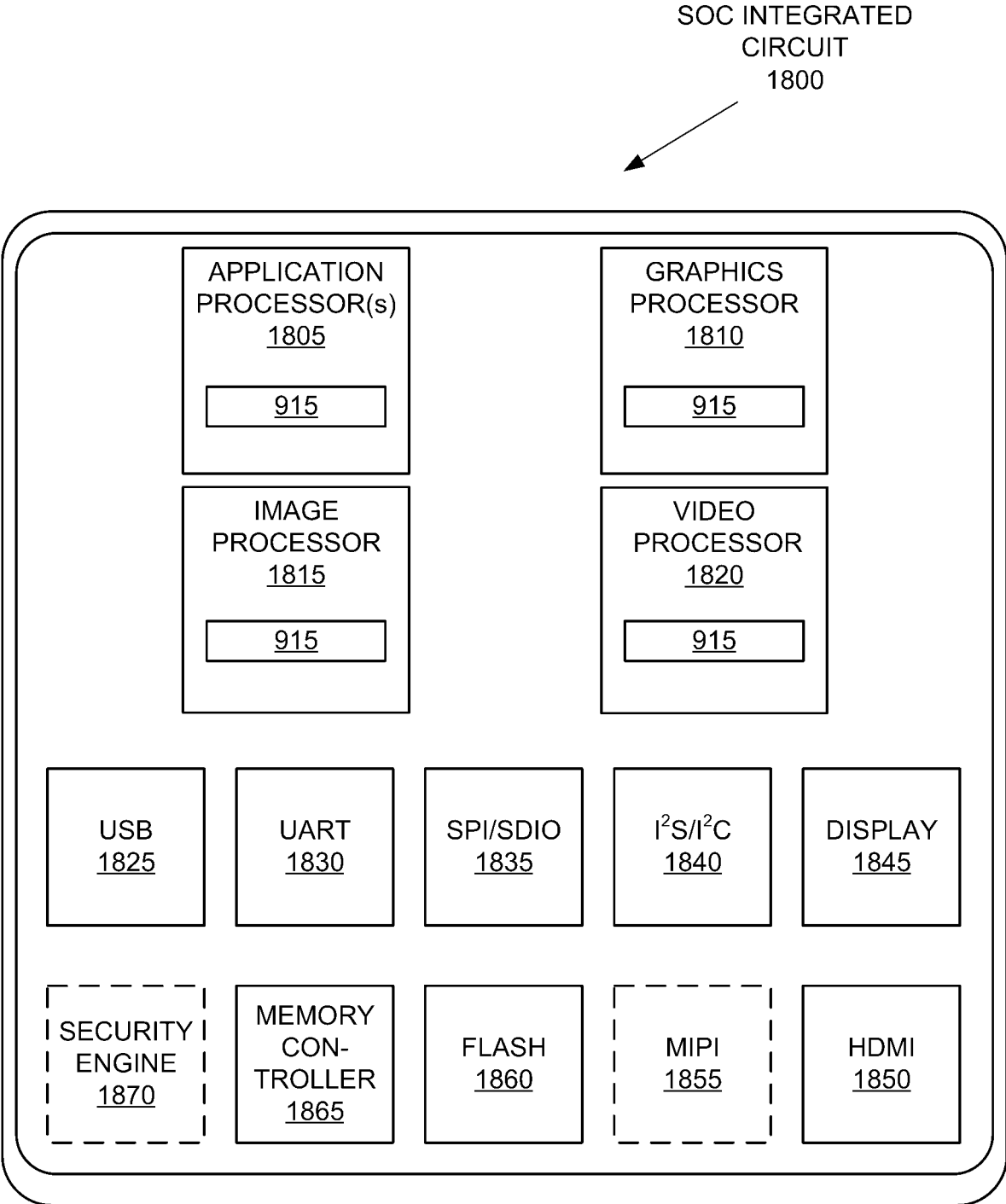


FIG. 18

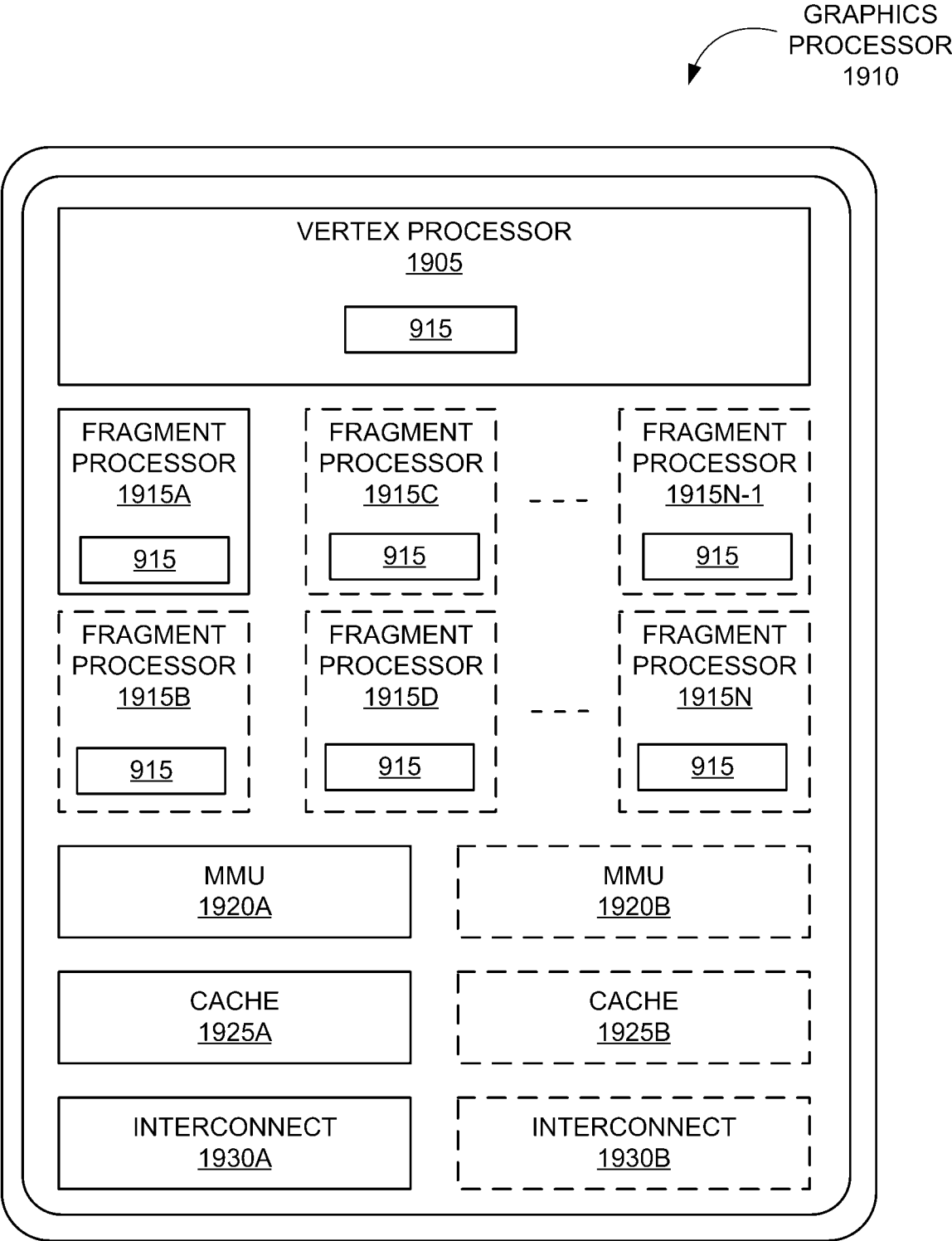


FIG. 19A

GRAPHICS
PROCESSOR
1940

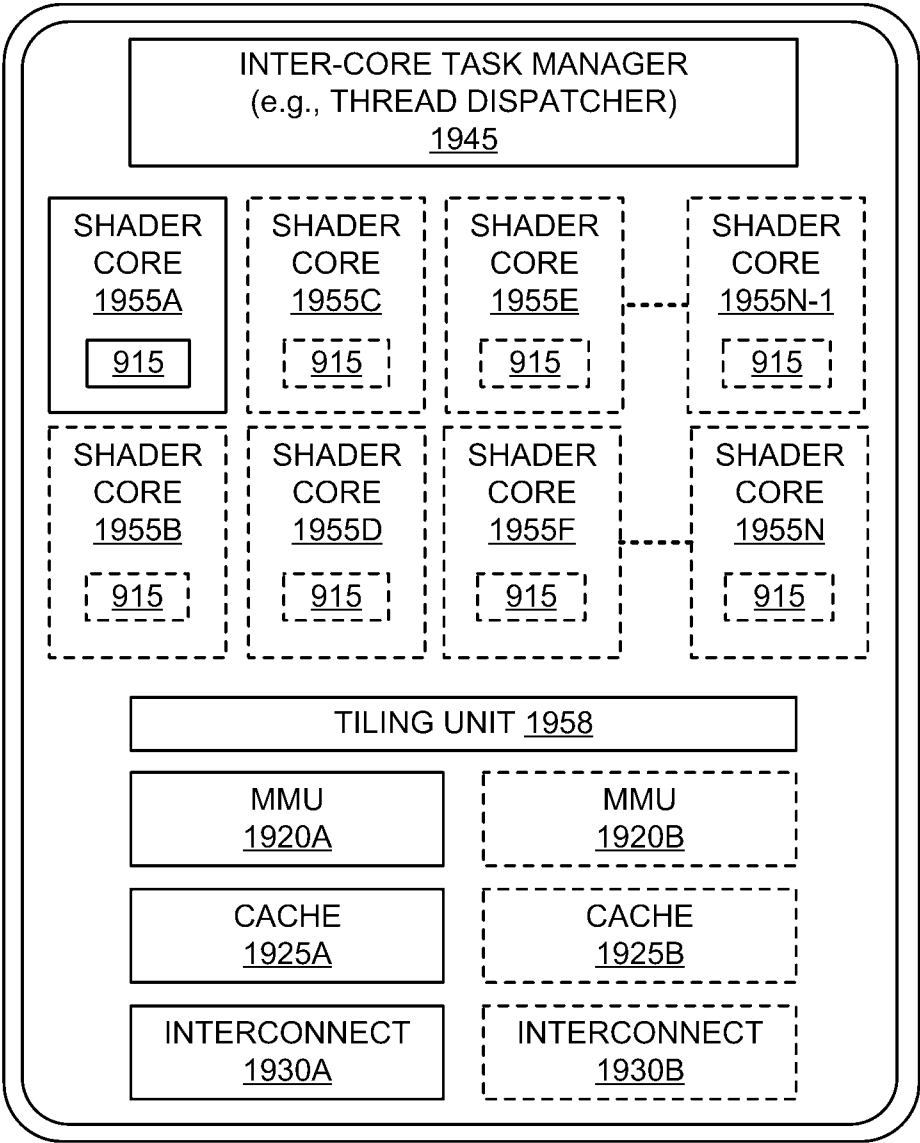
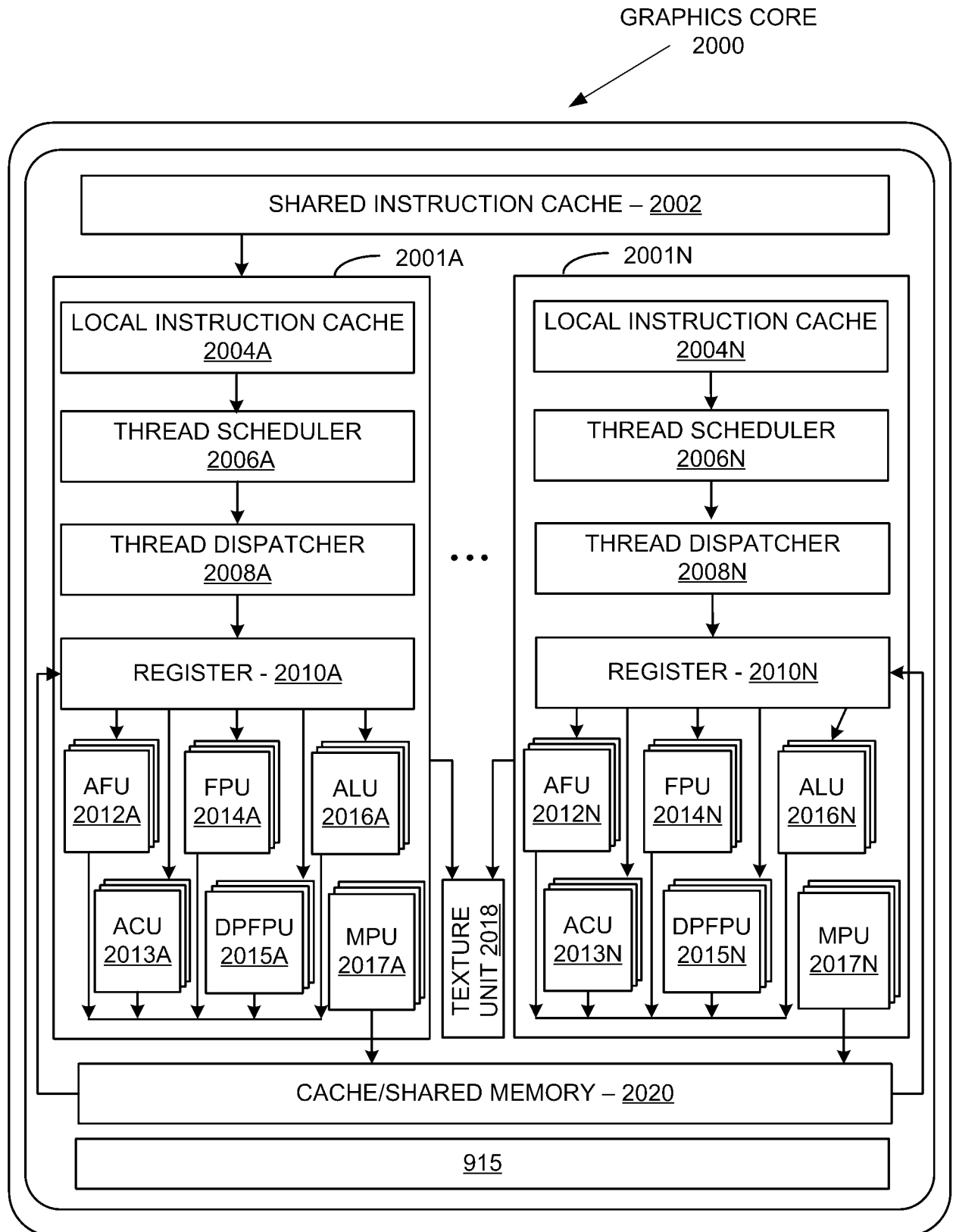


FIG. 19B

**FIG. 20A**

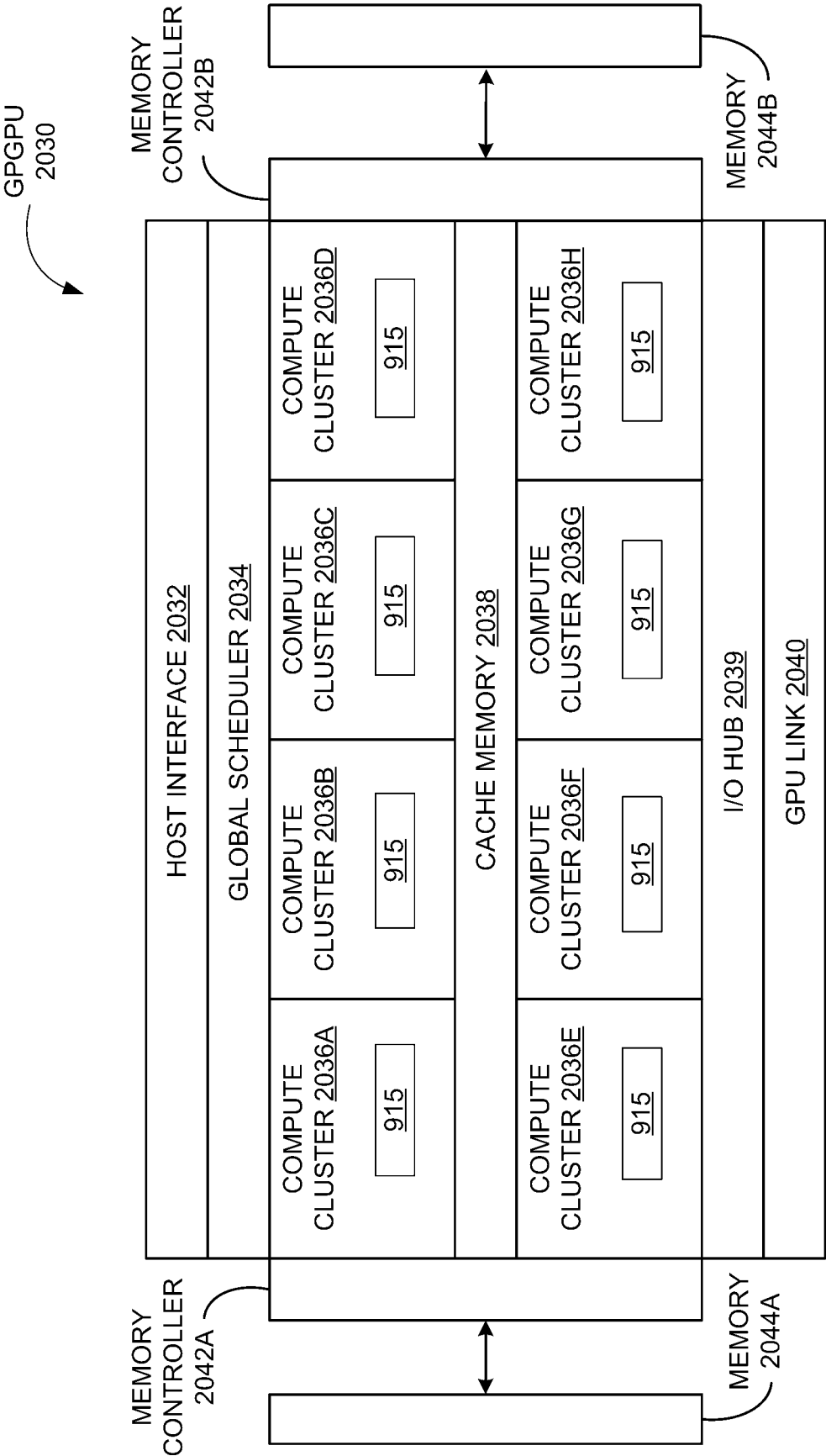
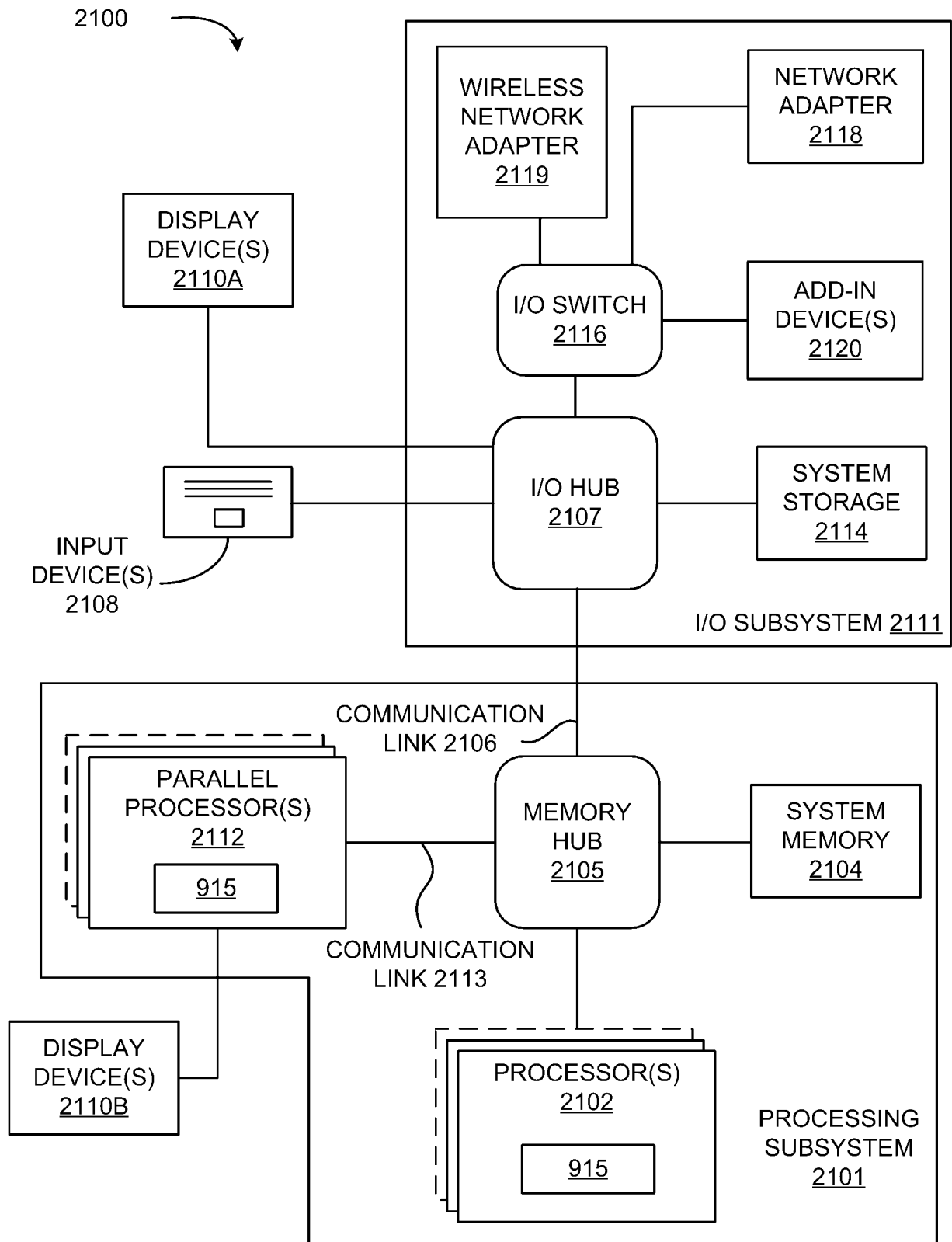
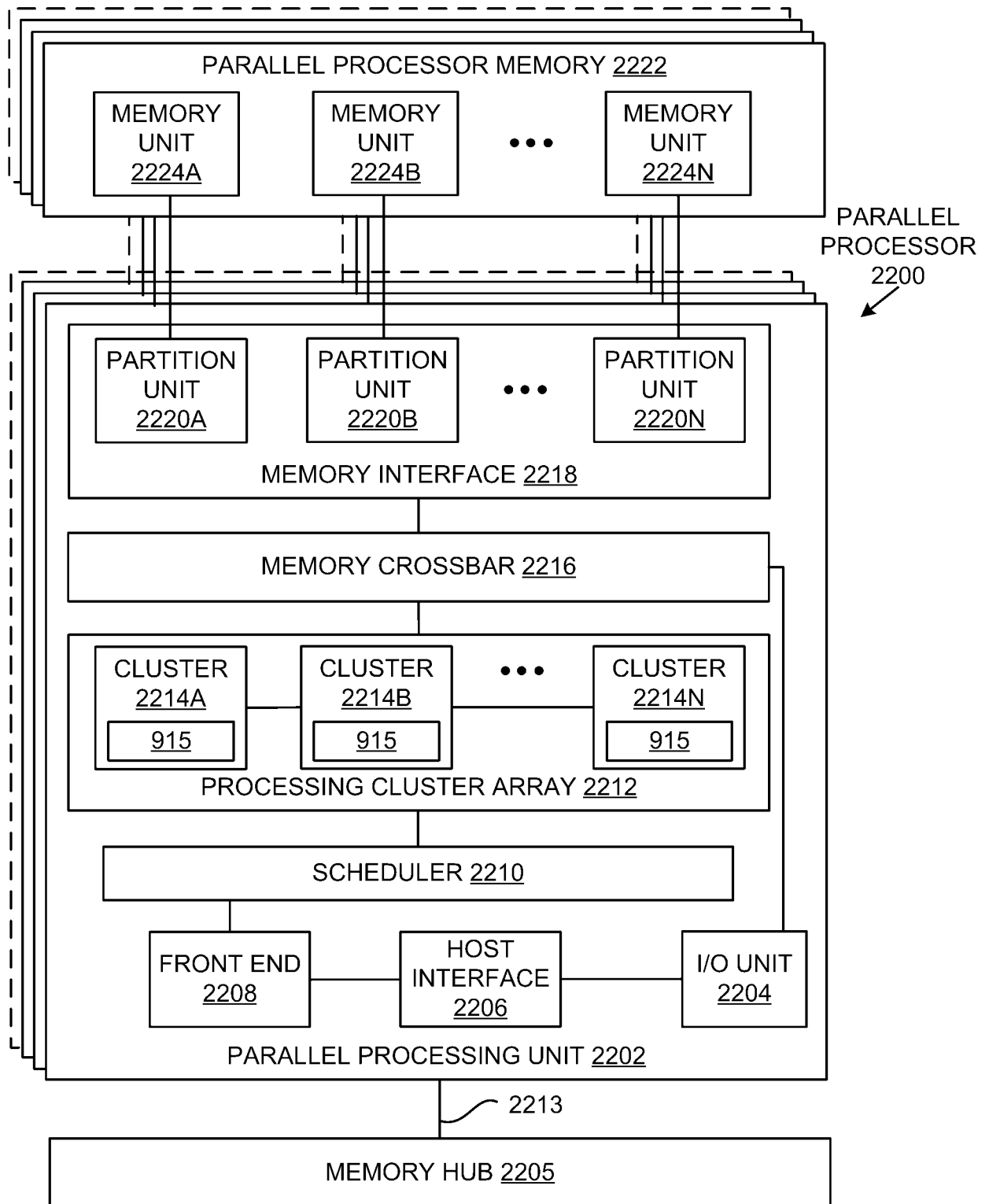
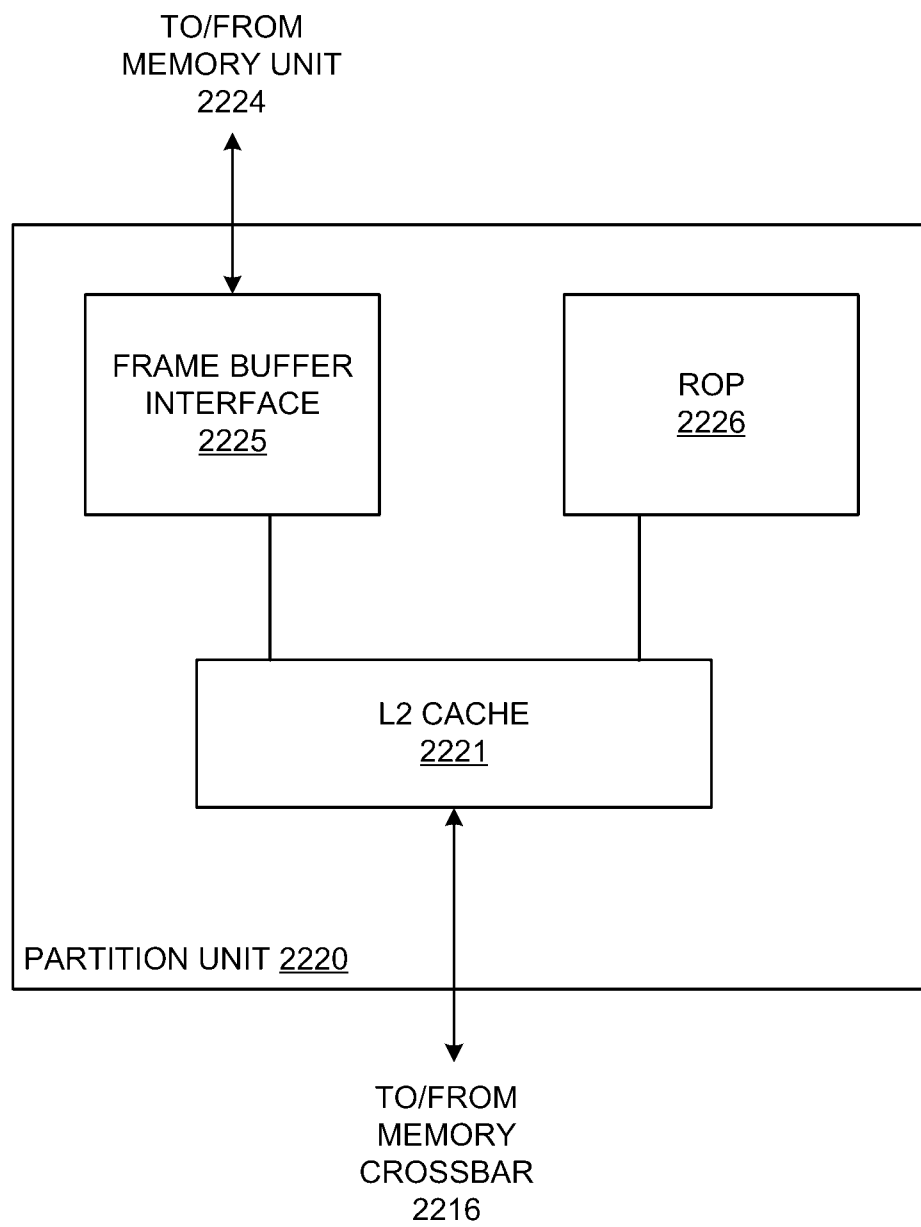
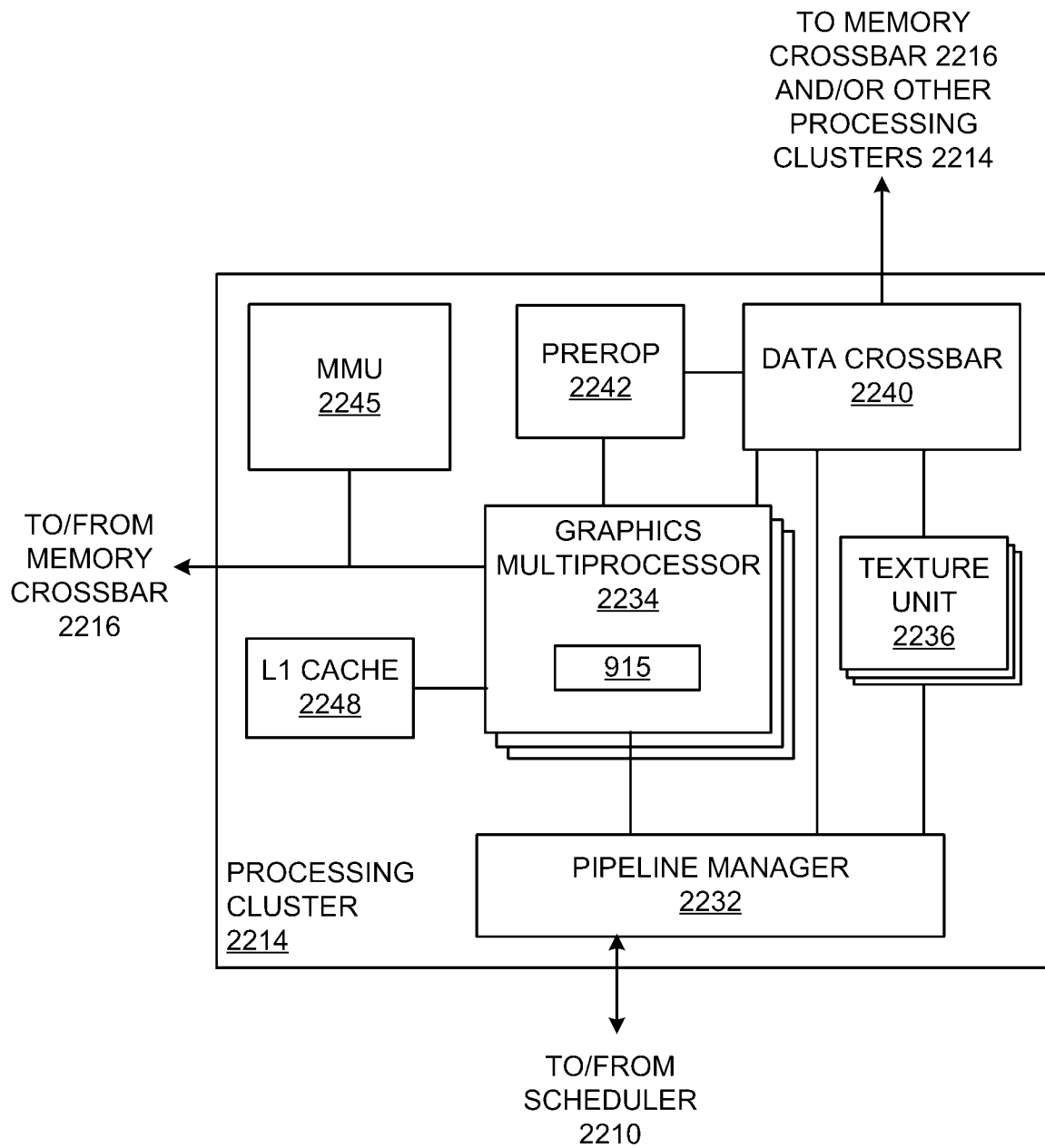


FIG. 20B

**FIG. 21**

**FIG. 22A**

**FIG. 22B**

**FIG. 22C**

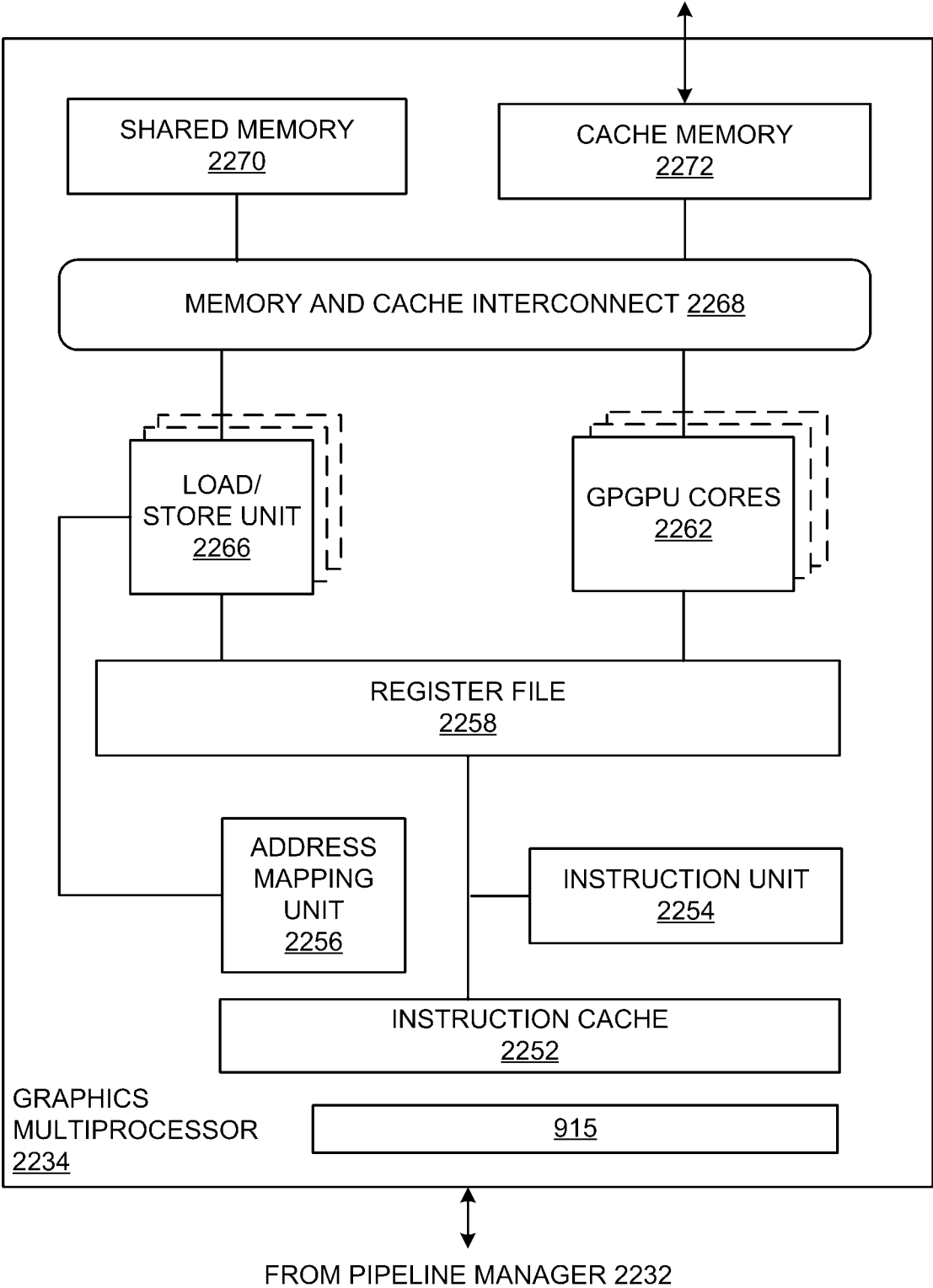


FIG. 22D

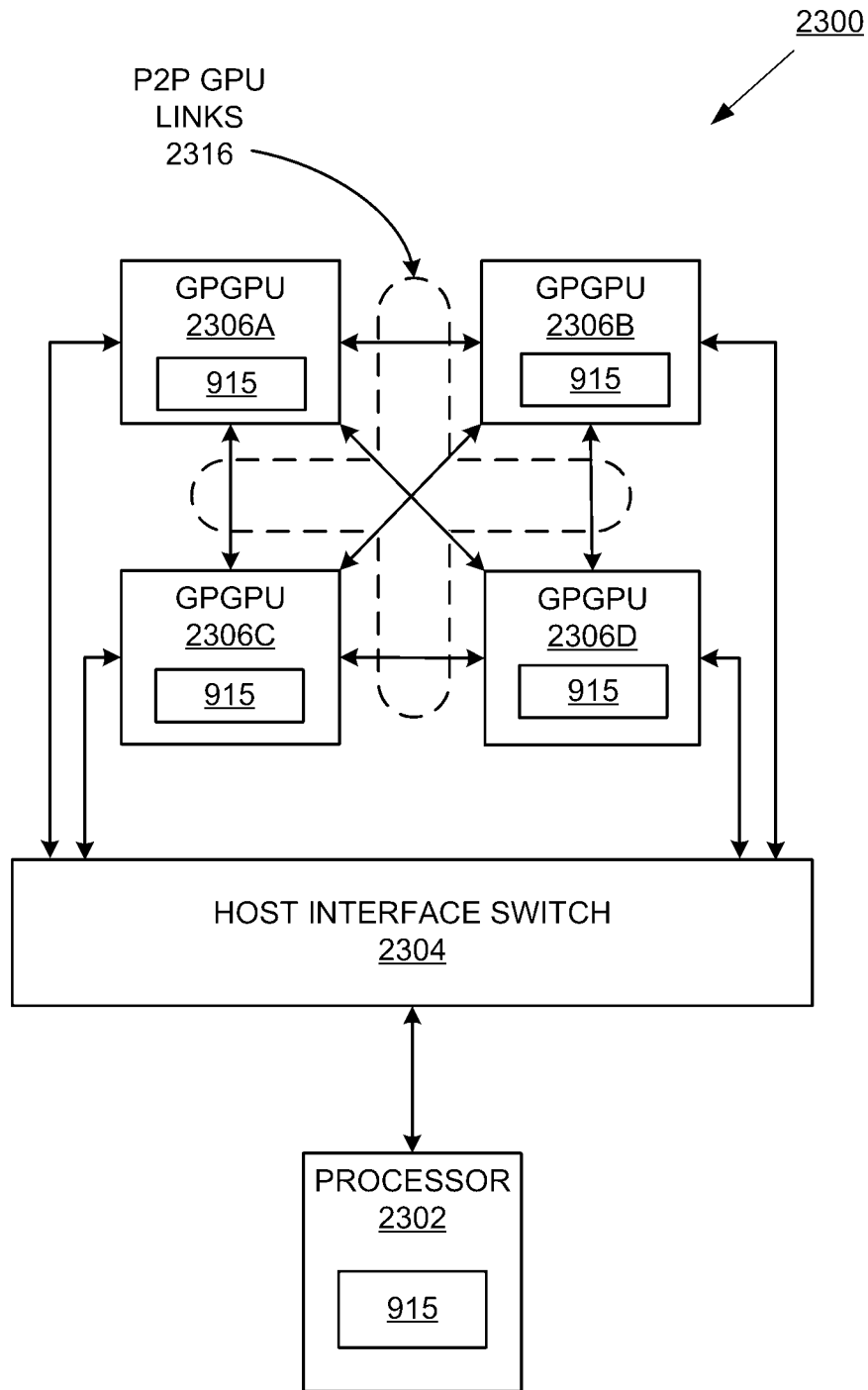


FIG. 23

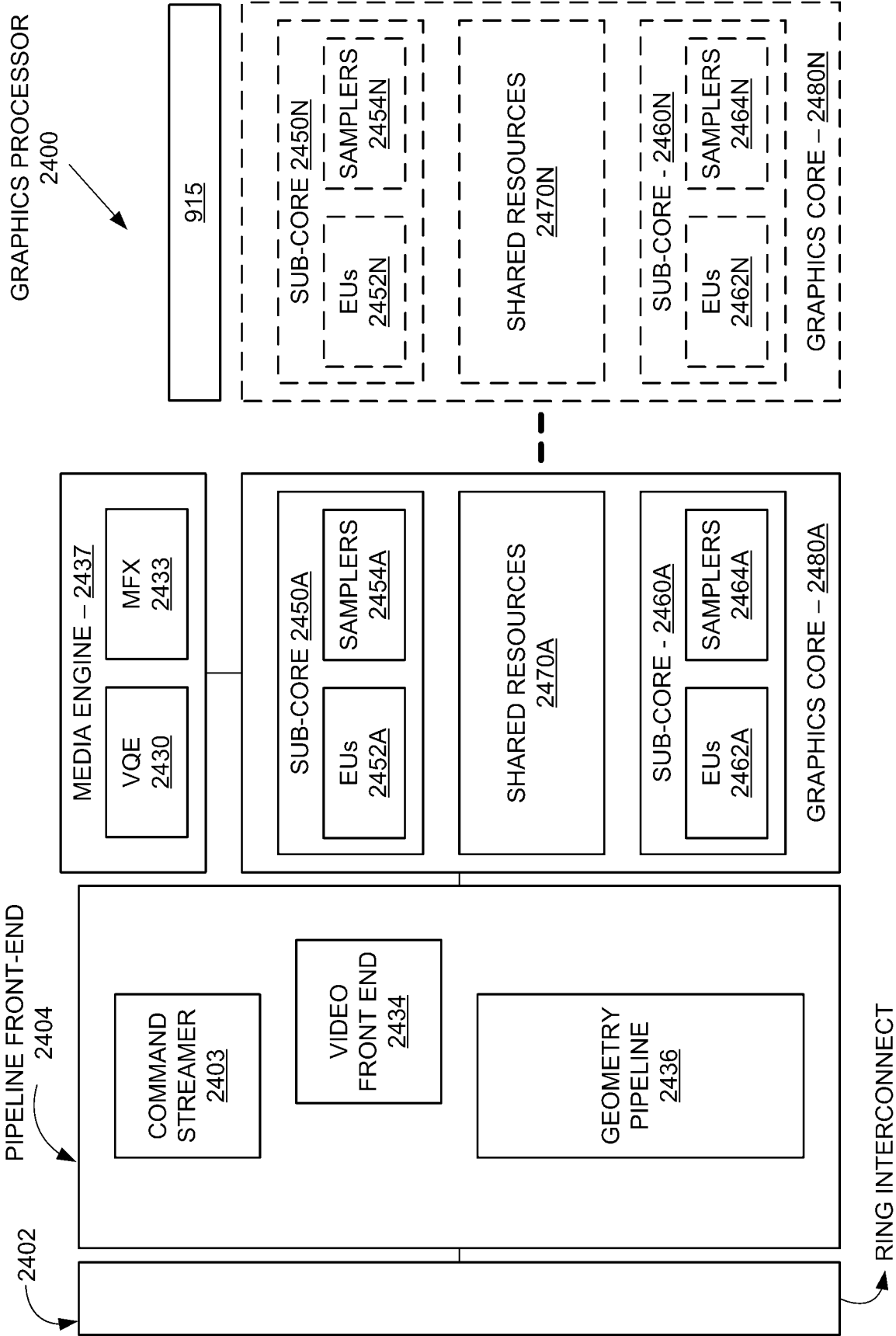


FIG. 24

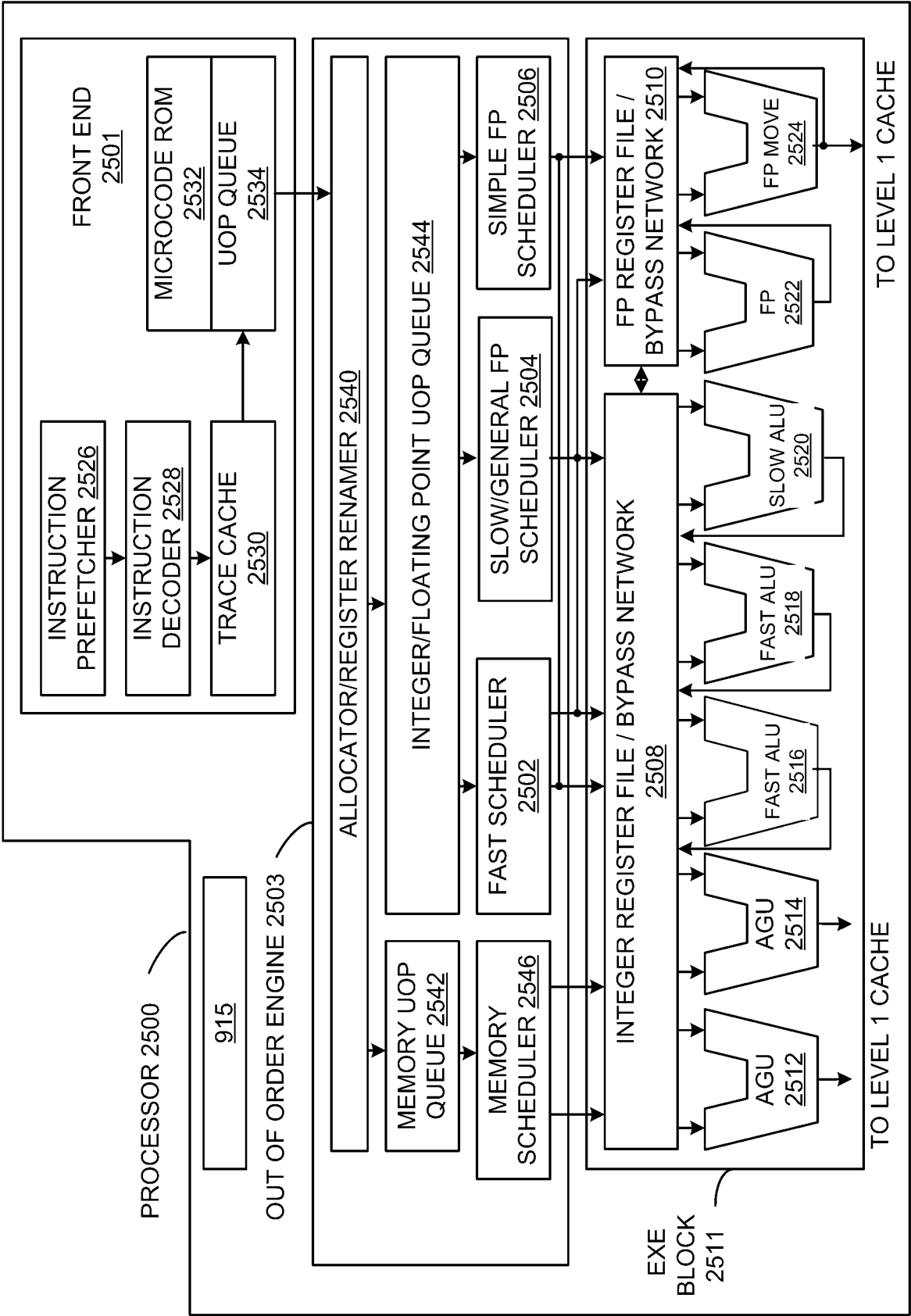


FIG. 25

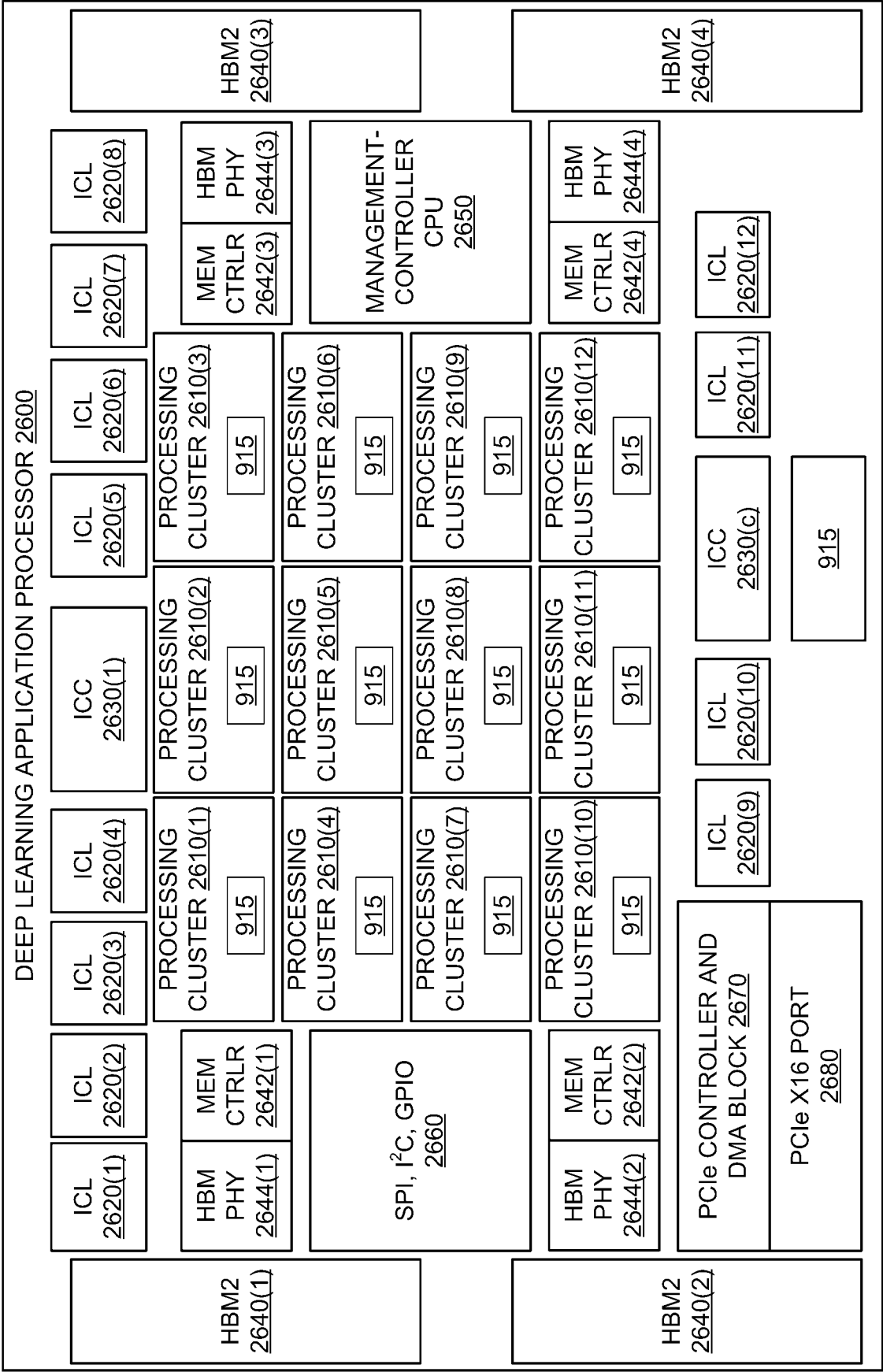
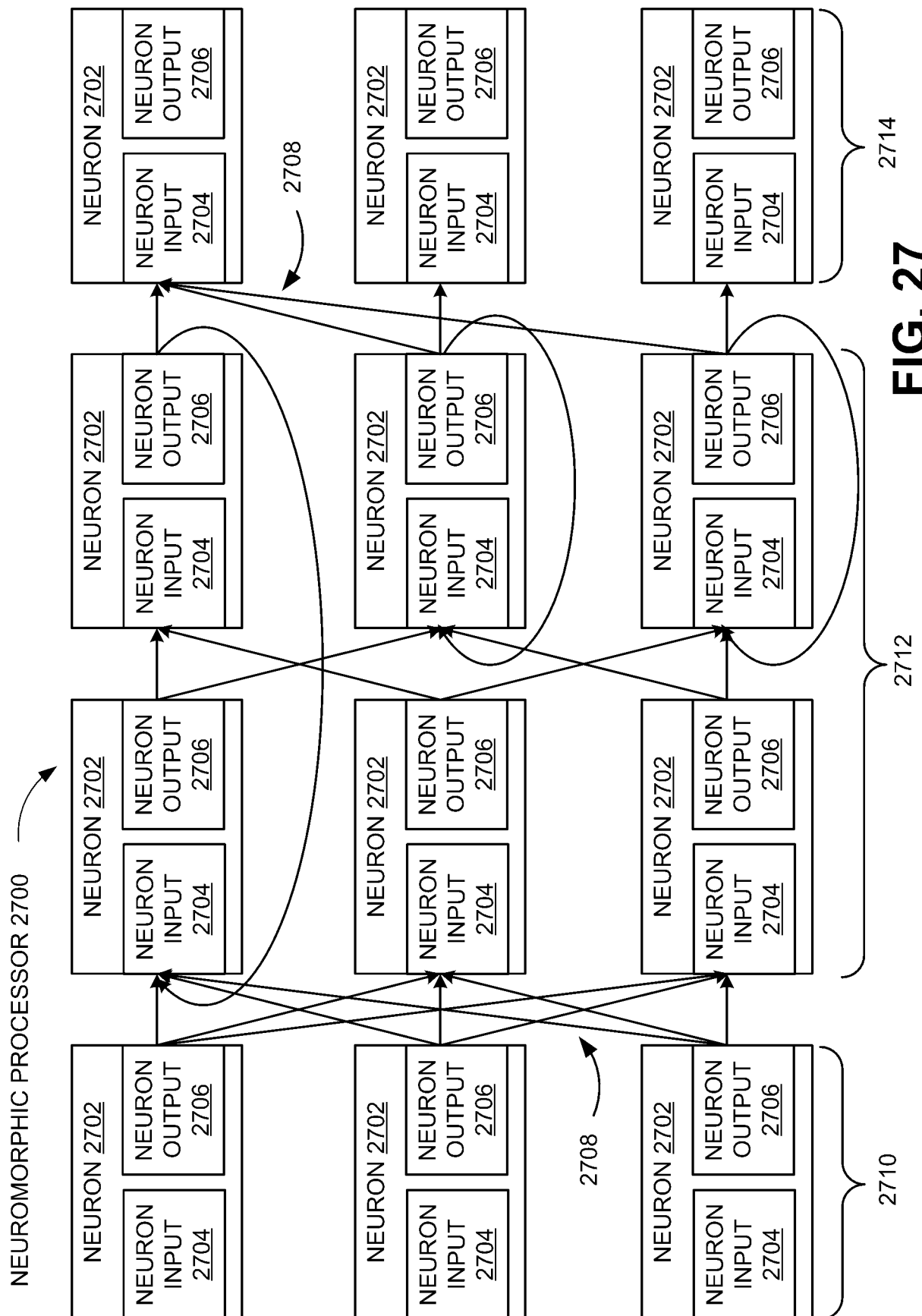


FIG. 26



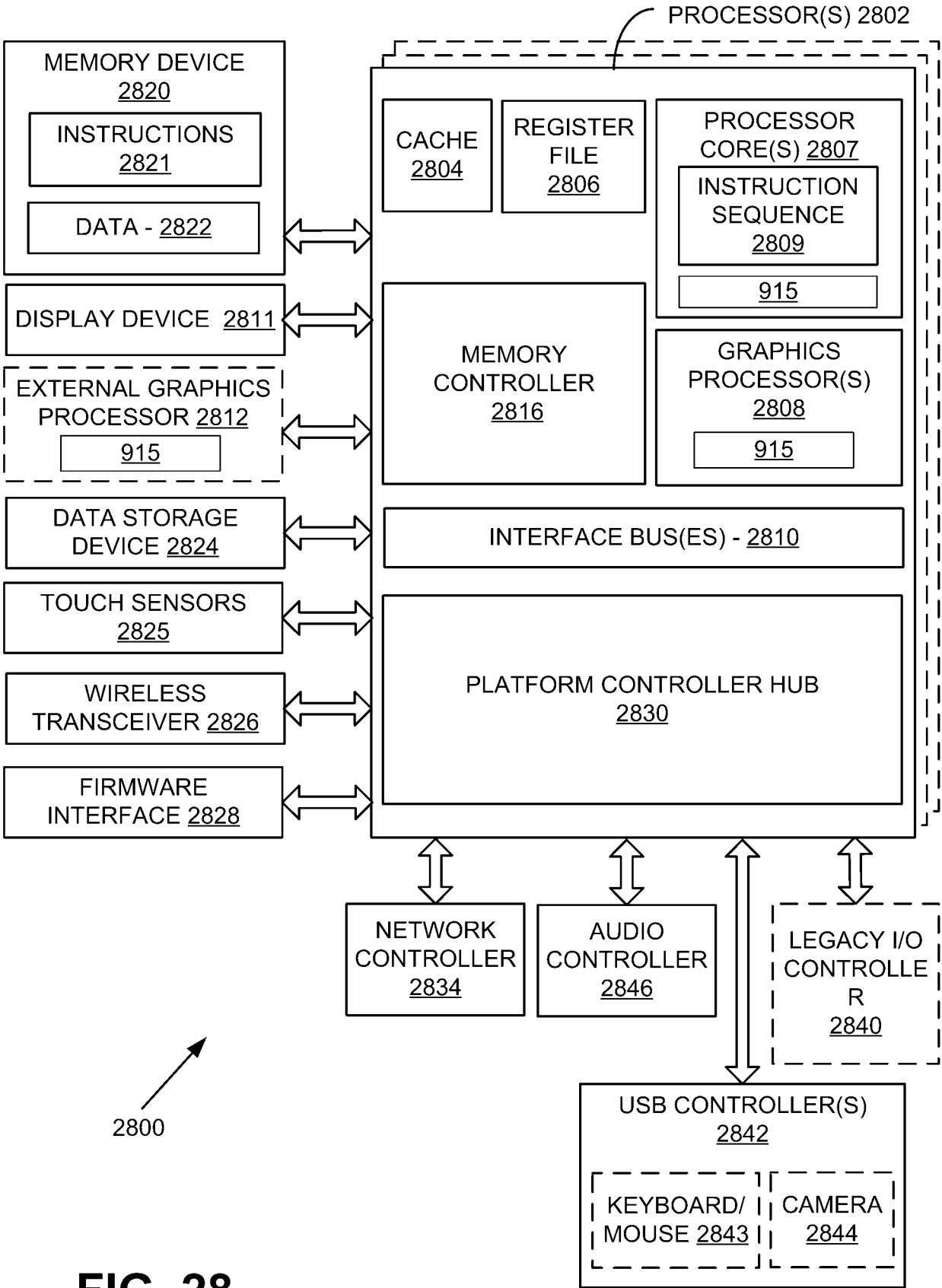


FIG. 28

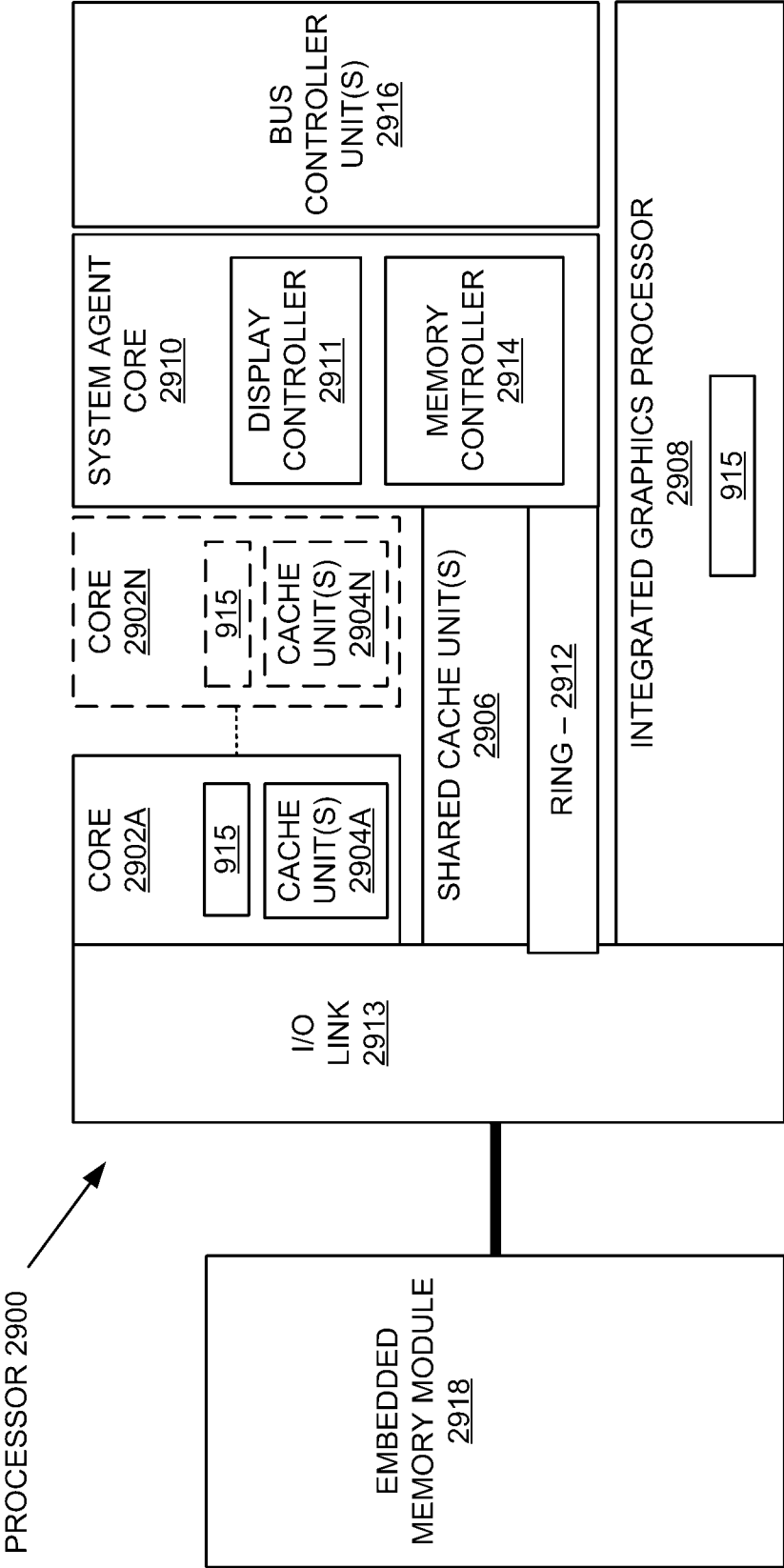


FIG. 29

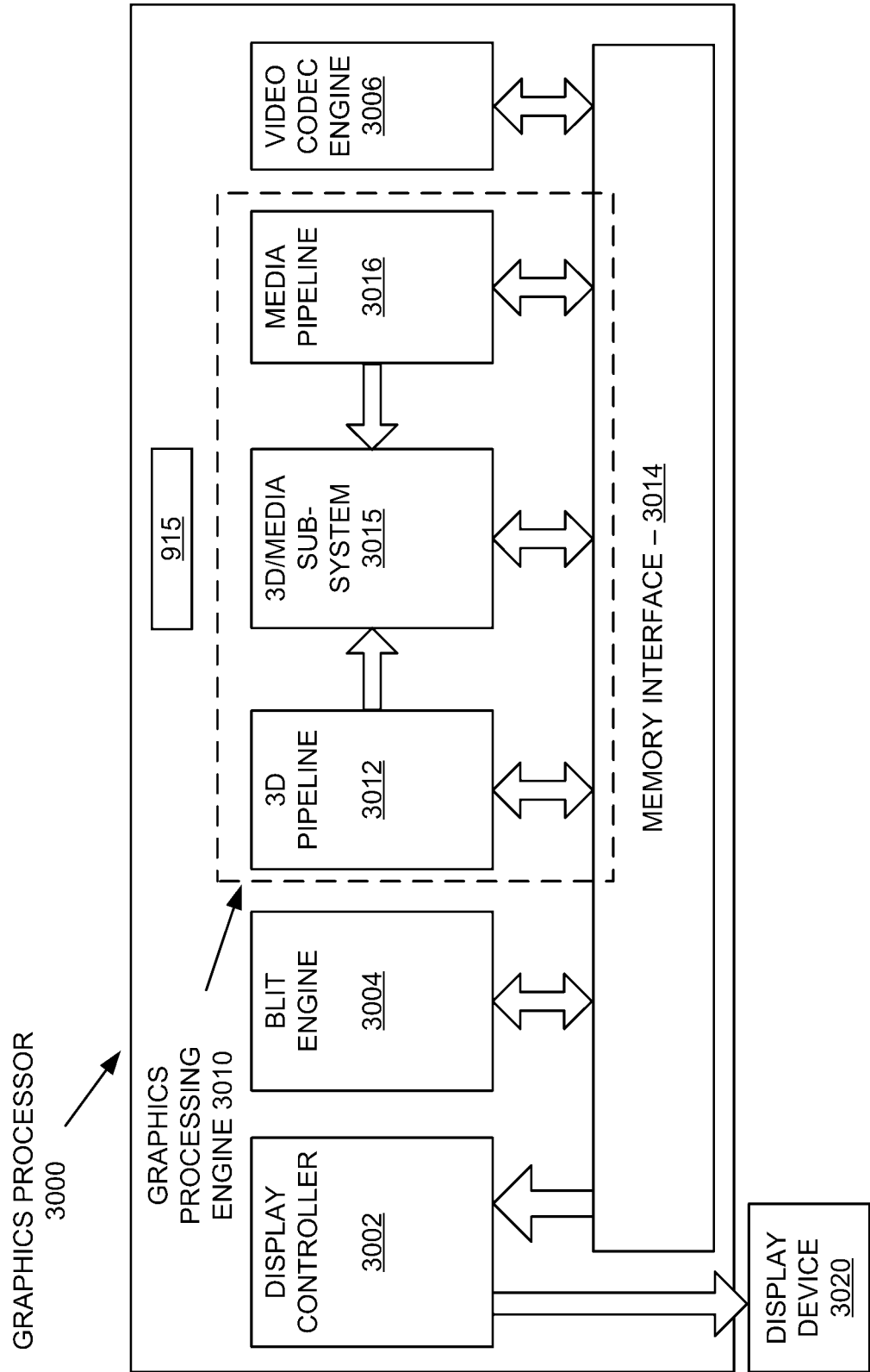


FIG. 30

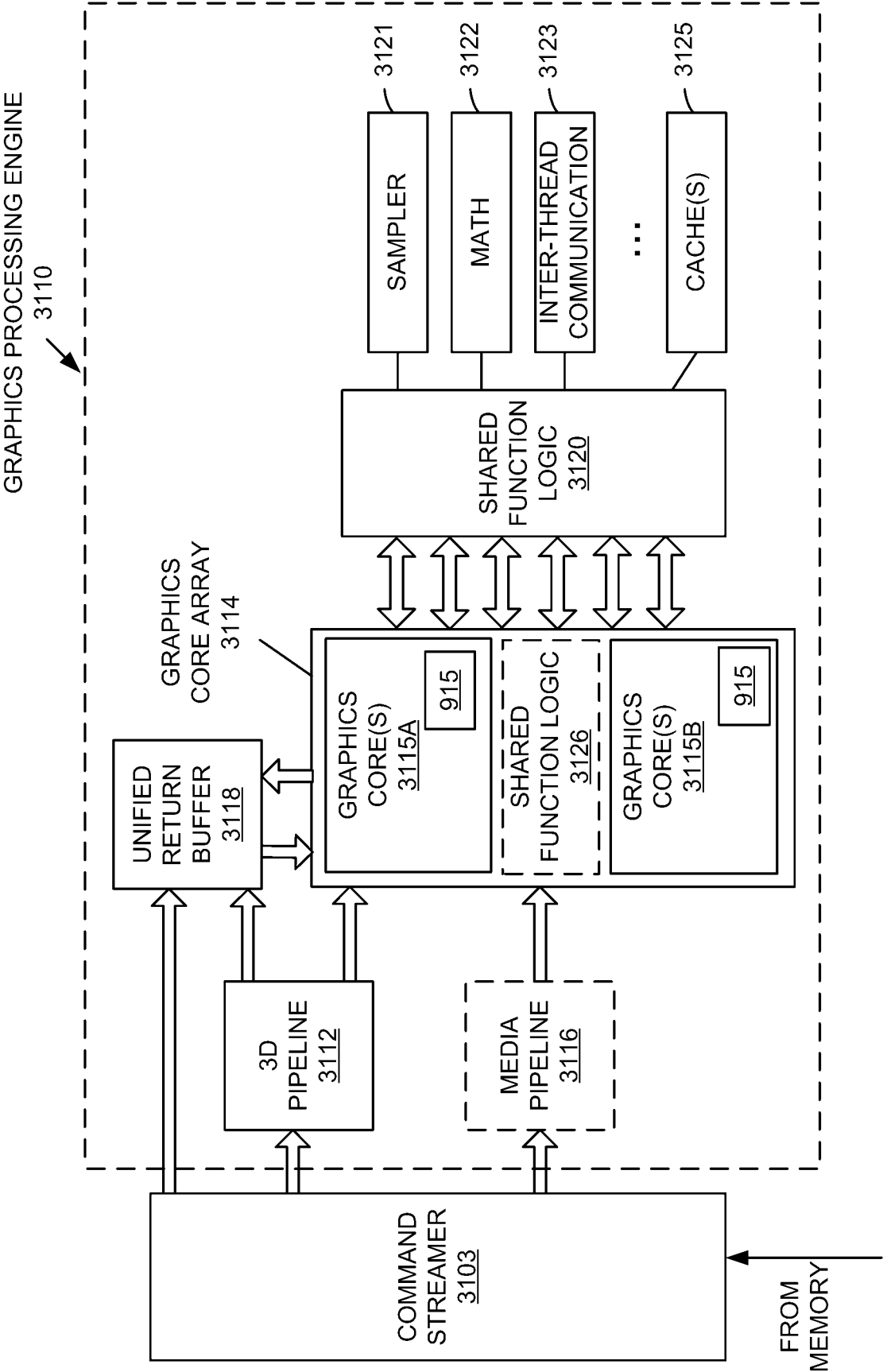
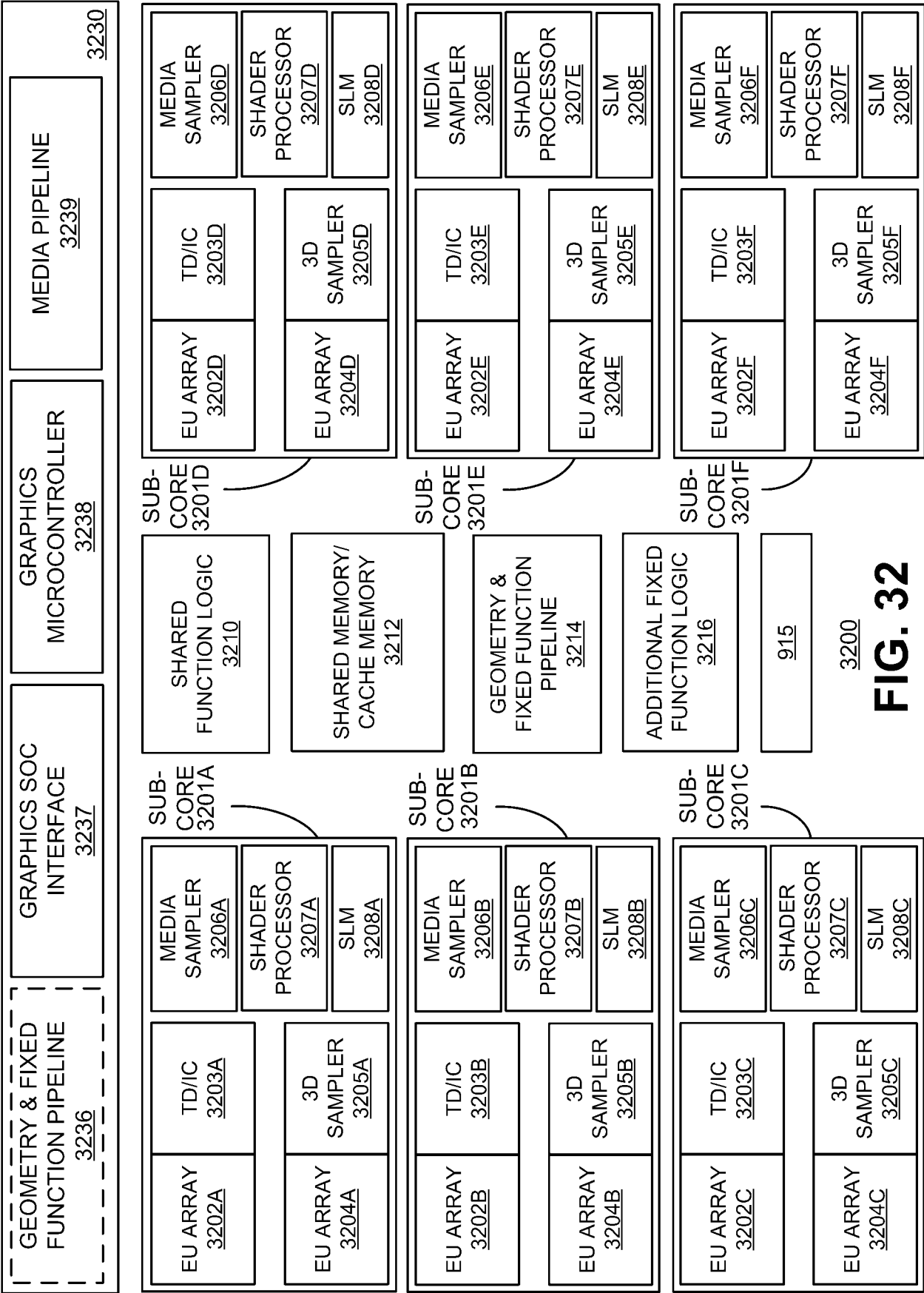


FIG. 31



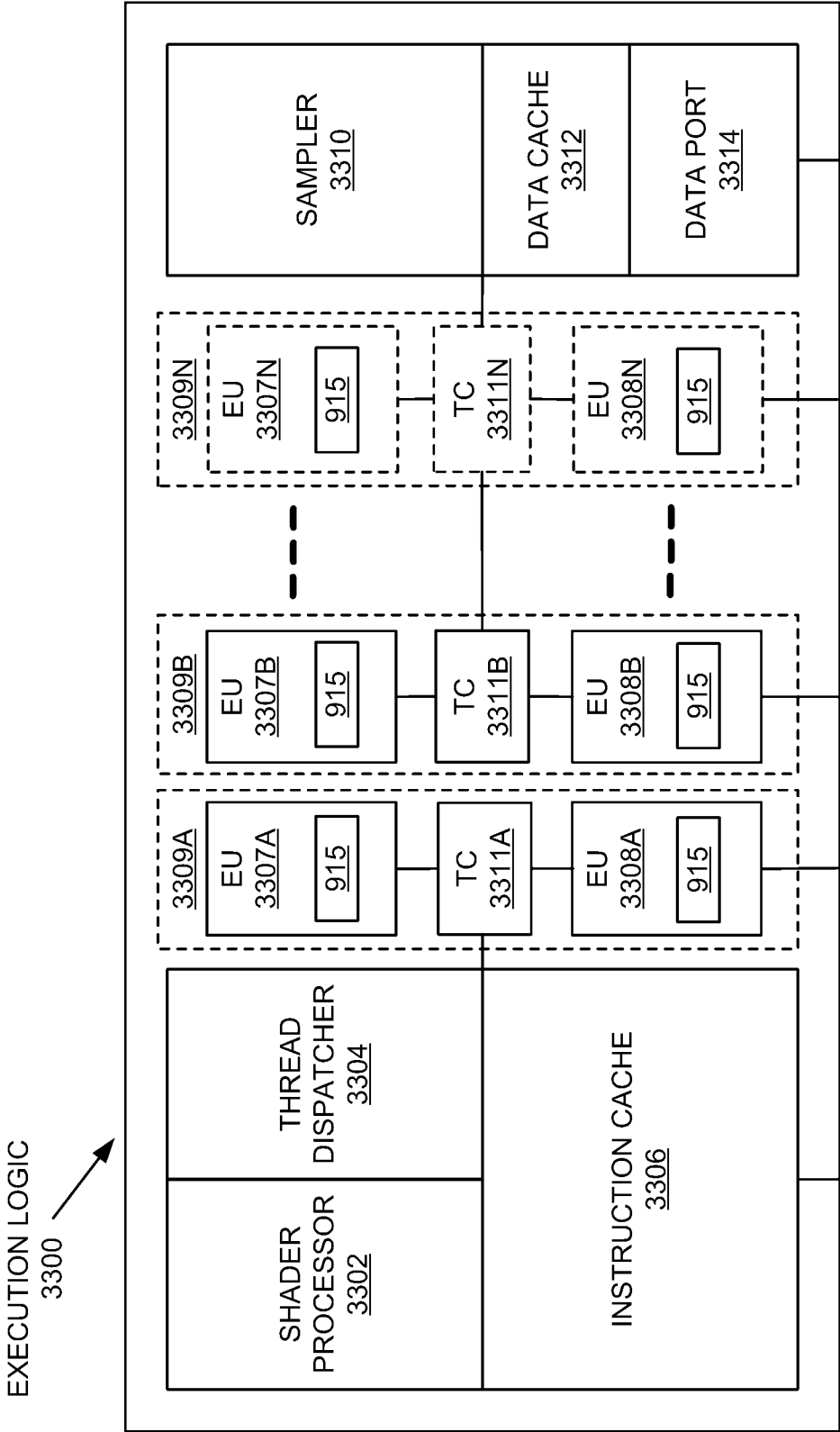


FIG. 33A

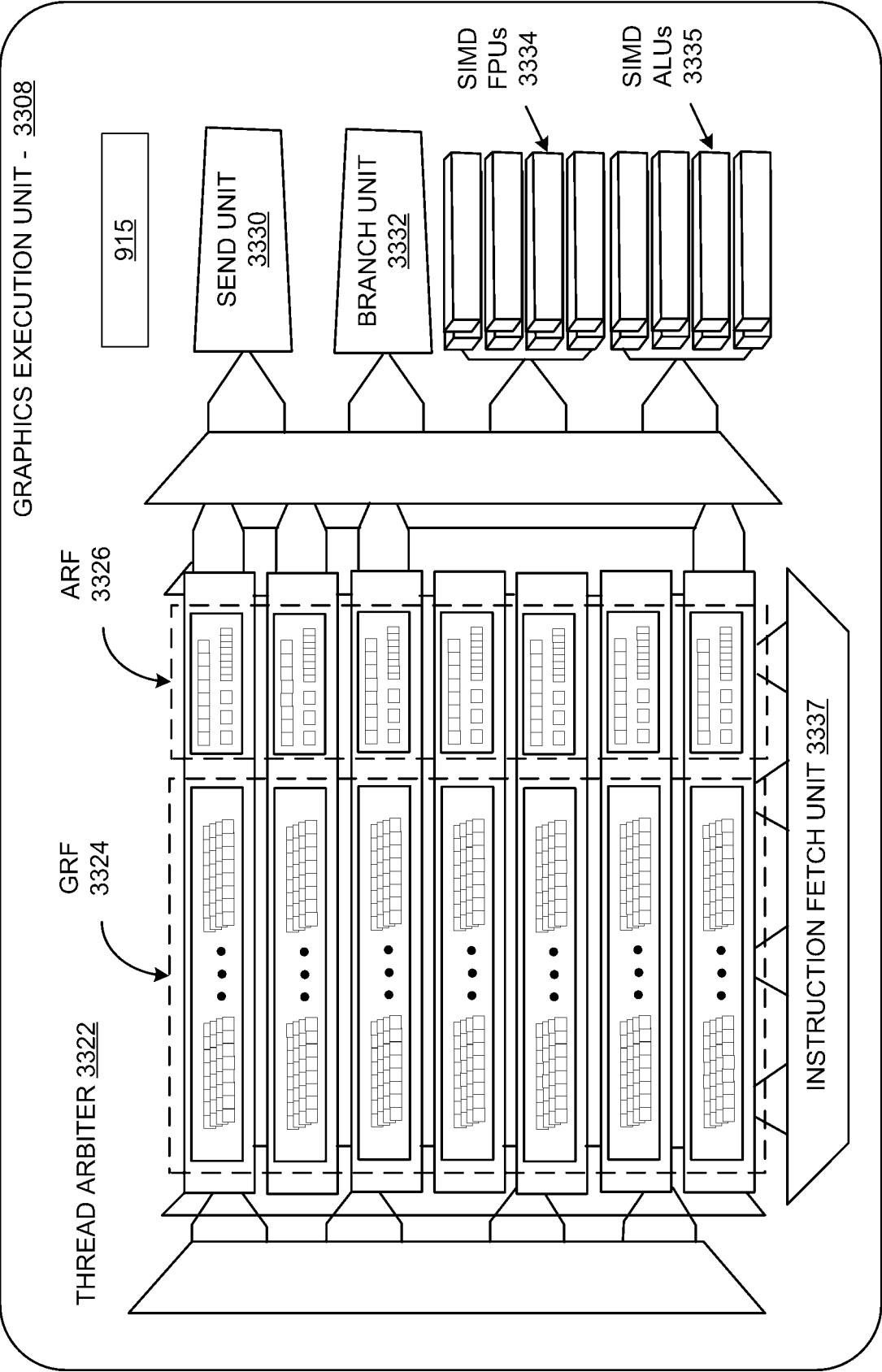
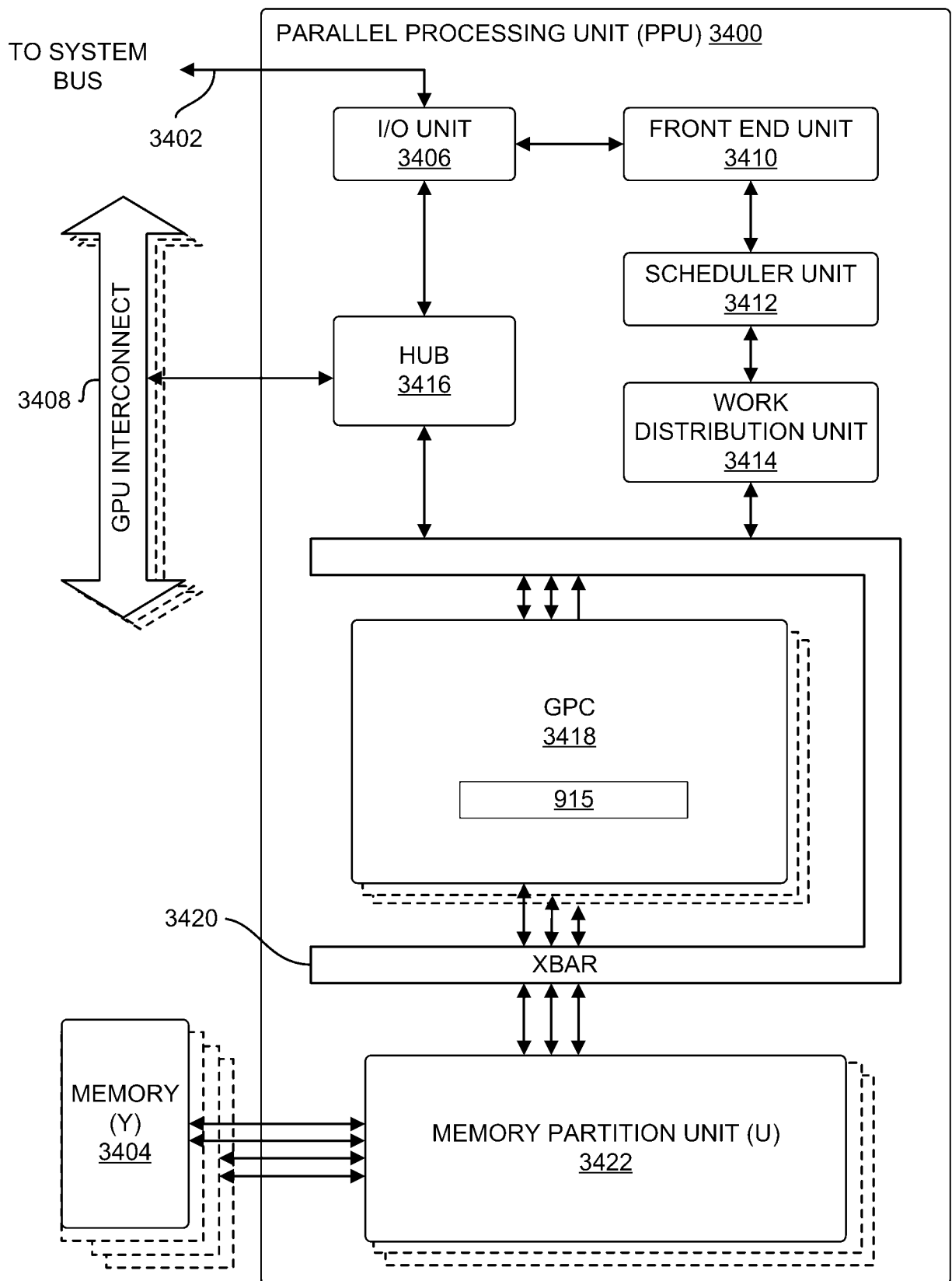


FIG. 33B

**FIG. 34**

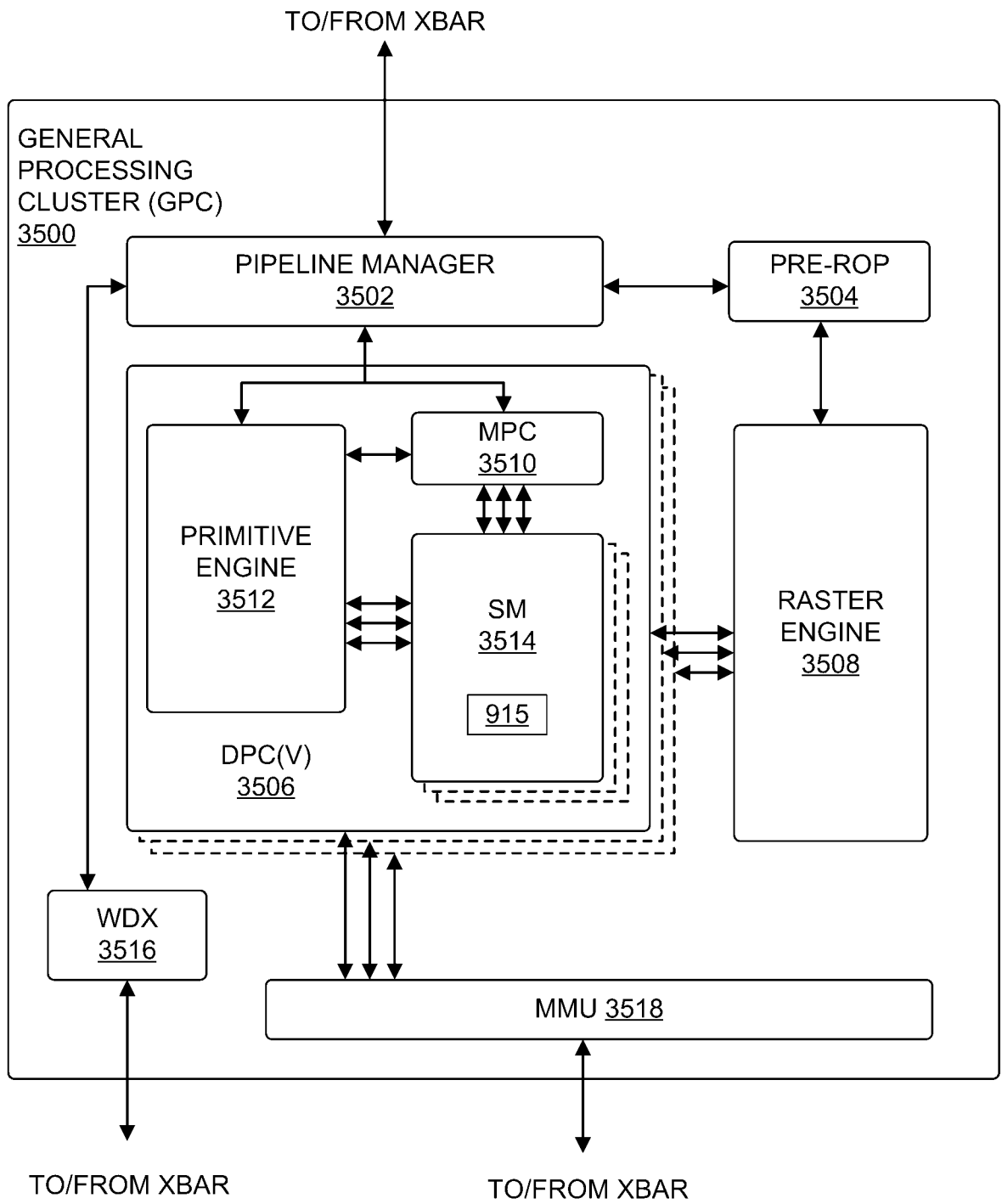
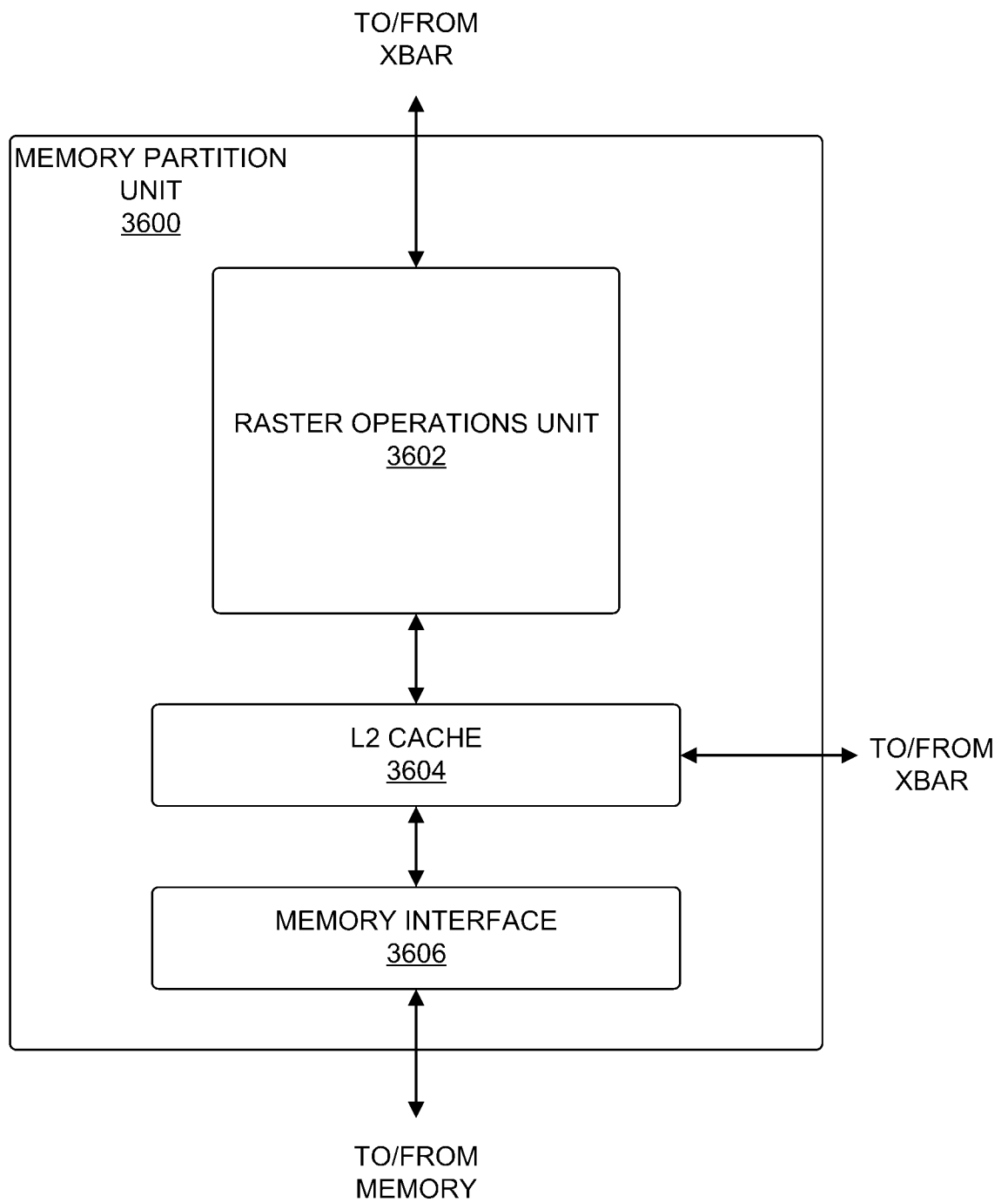
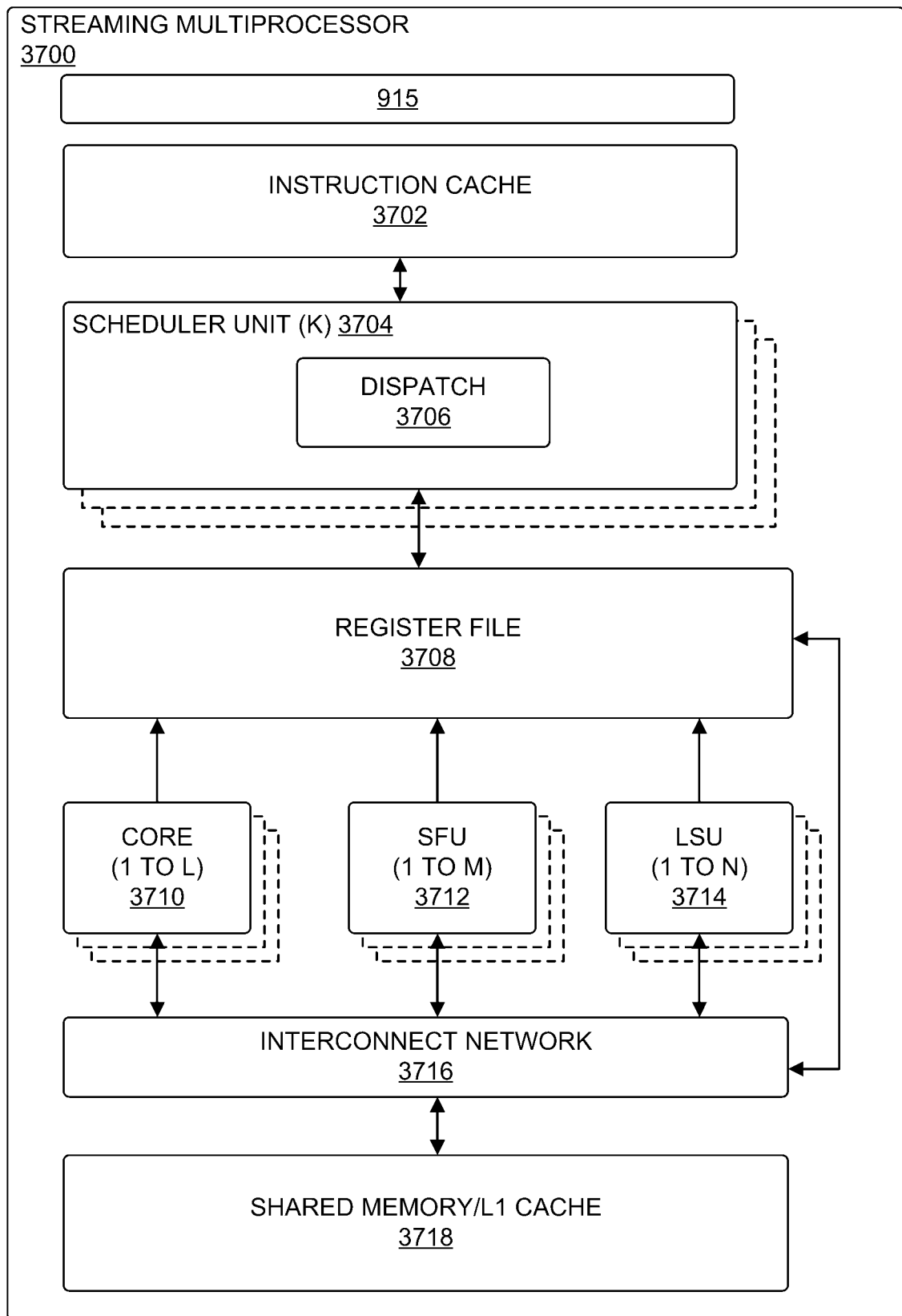


FIG. 35

**FIG. 36**

**FIG. 37**

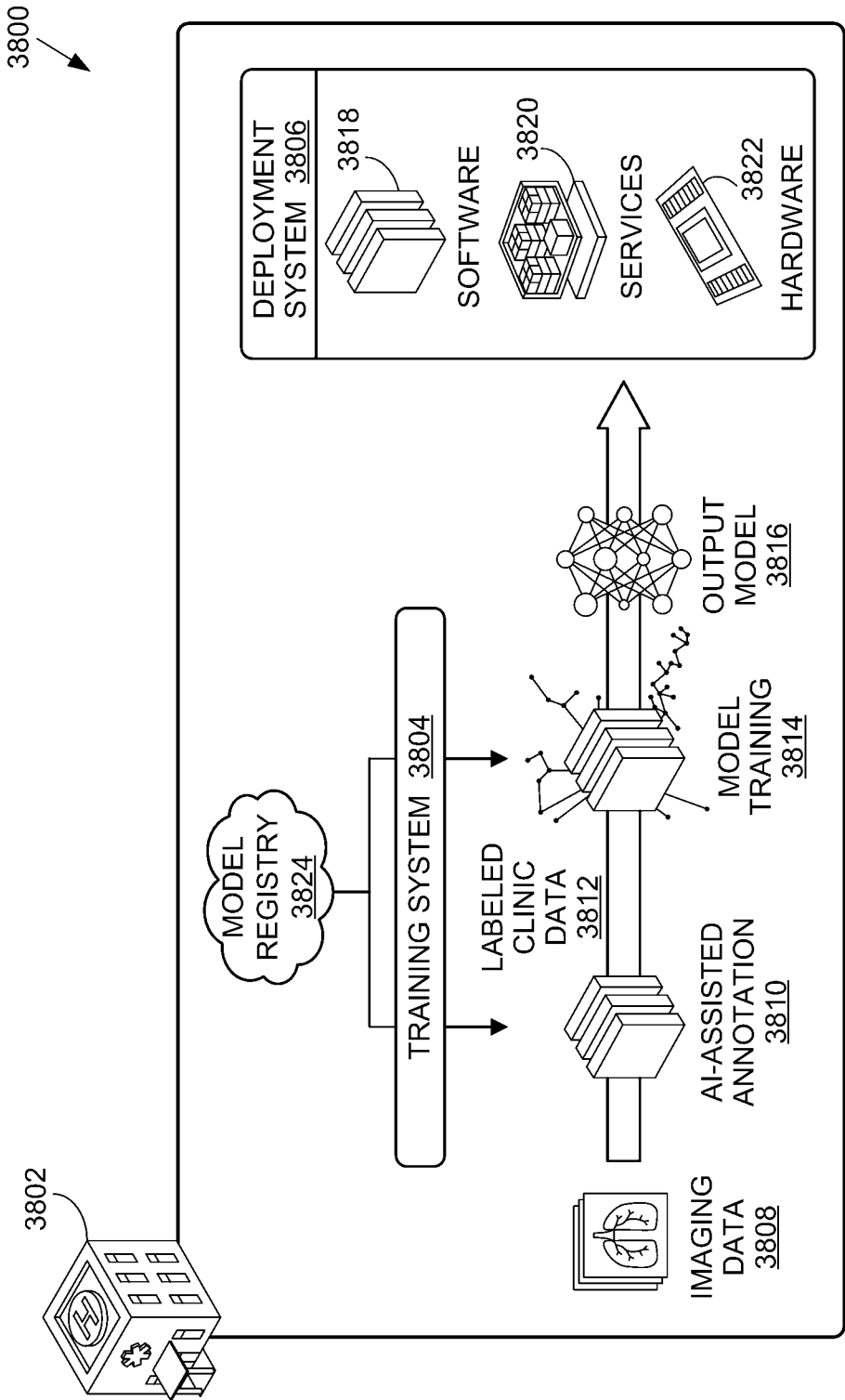


FIG. 38

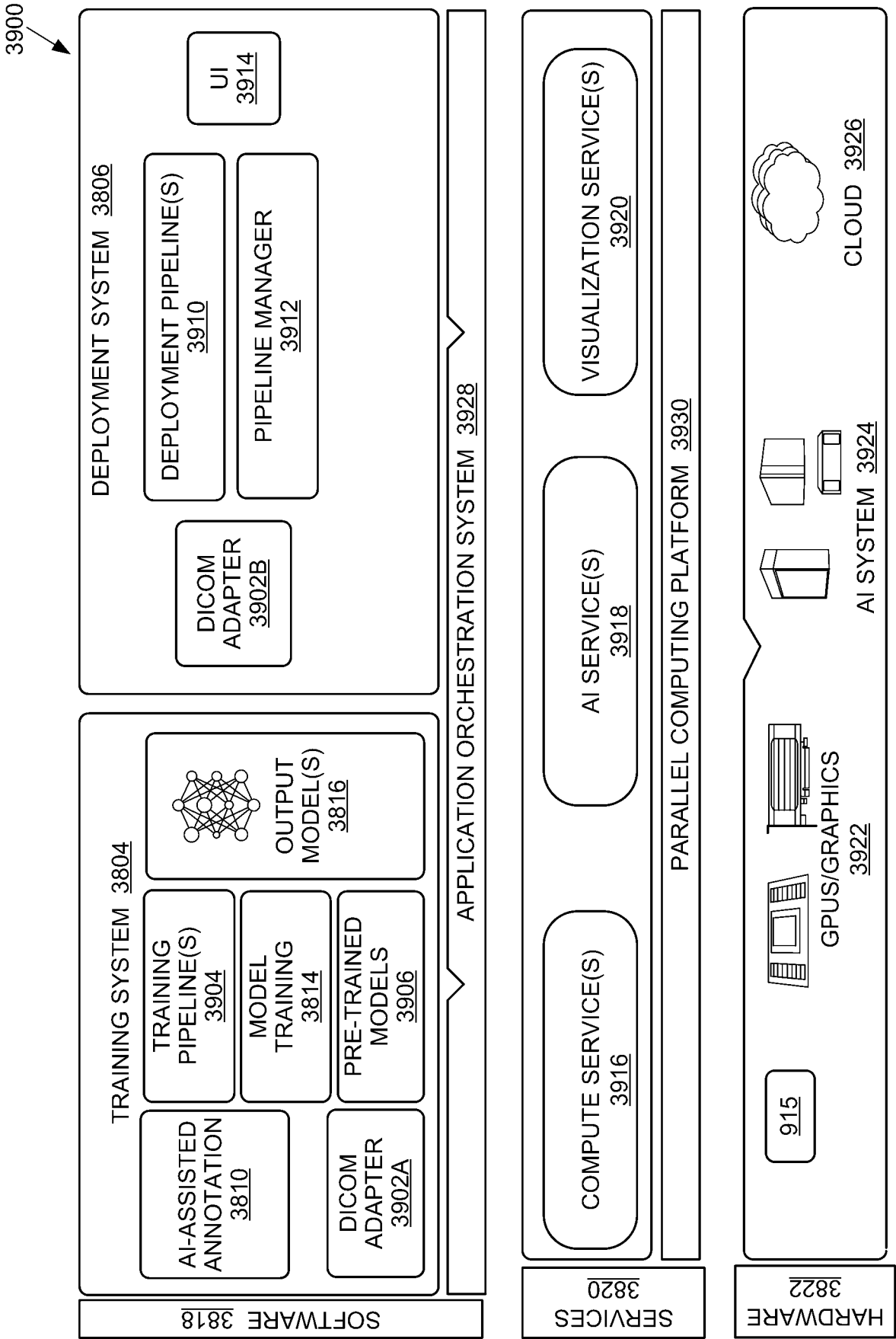


FIG. 39

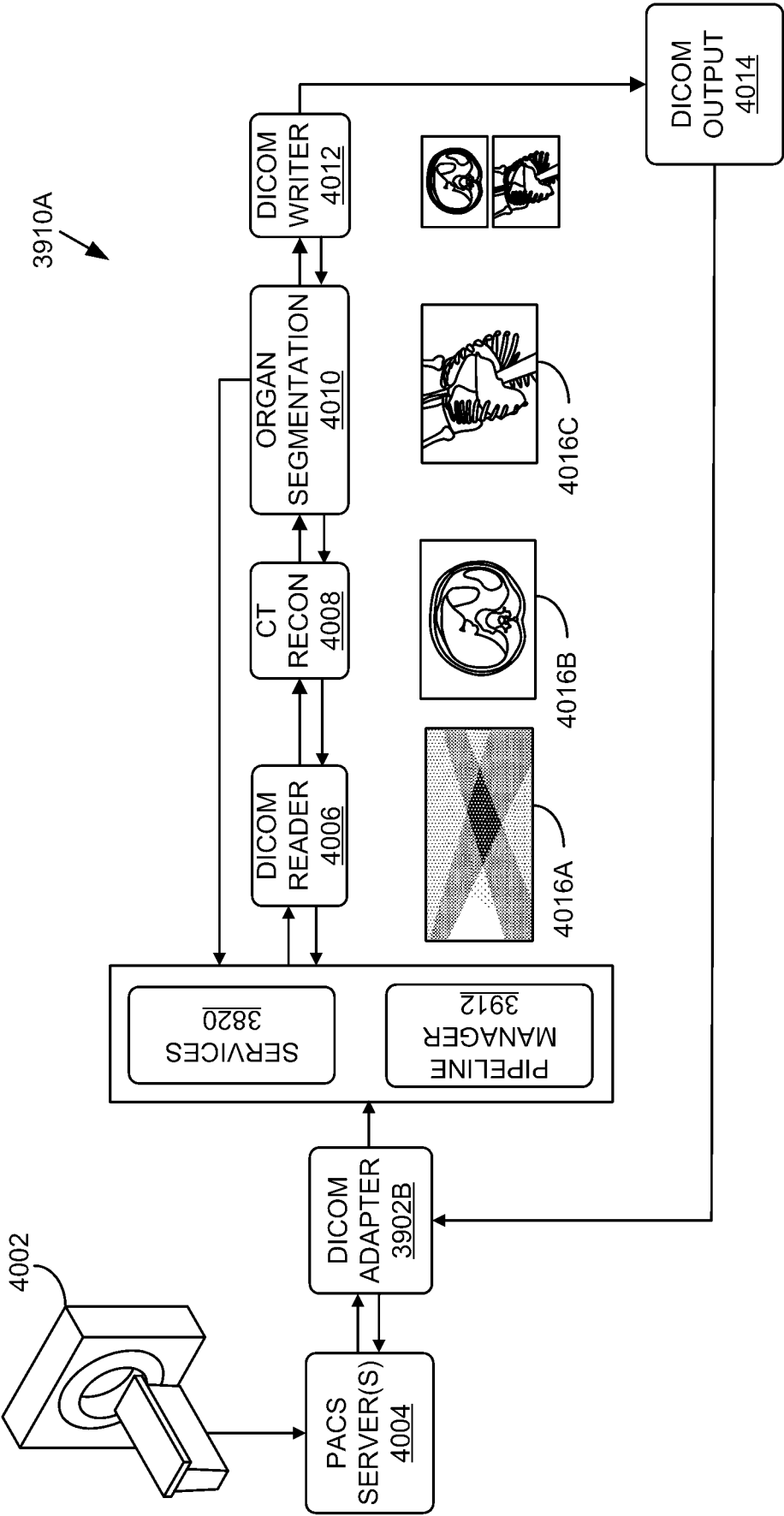


FIG. 40

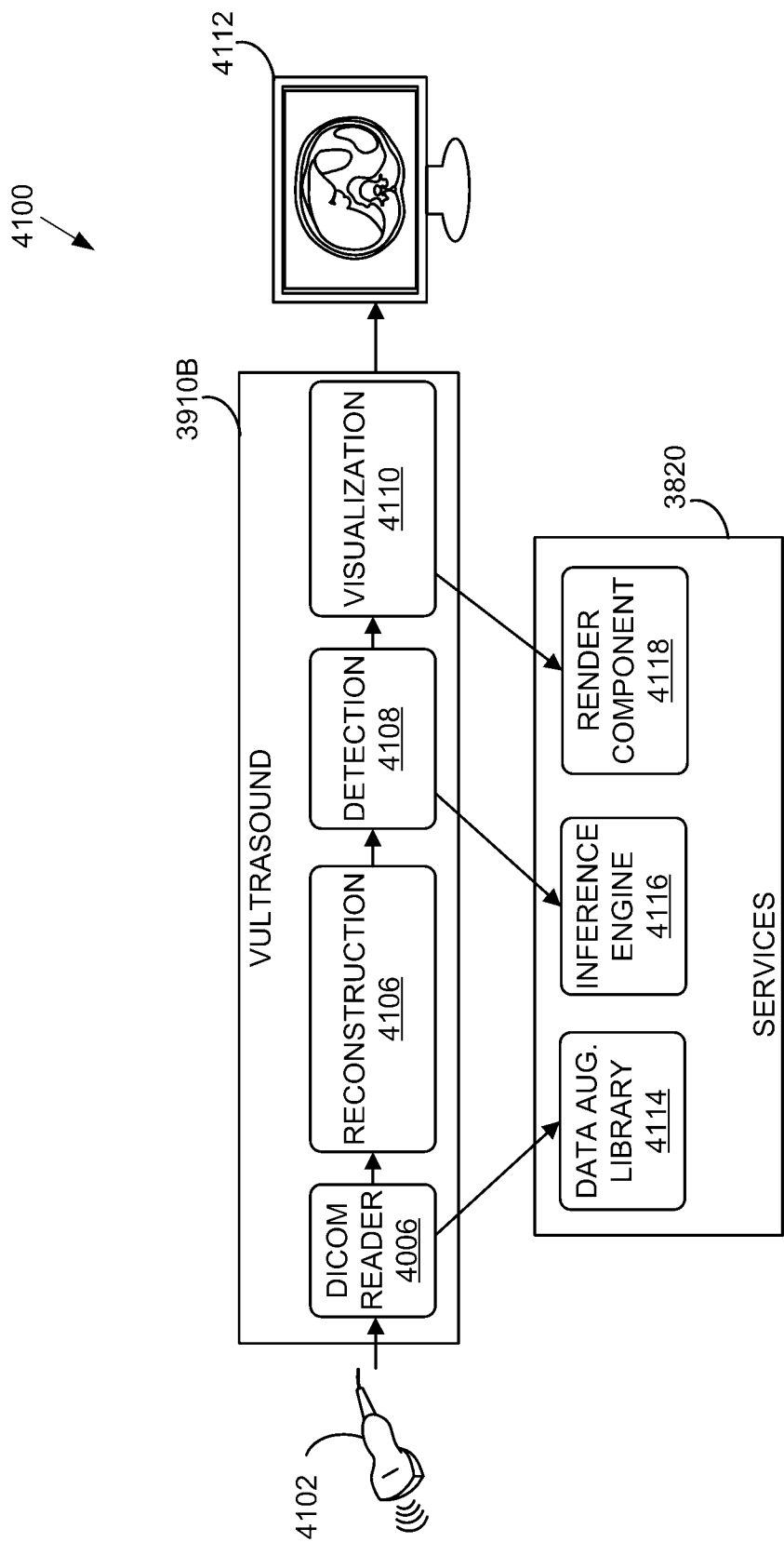


FIG. 41A

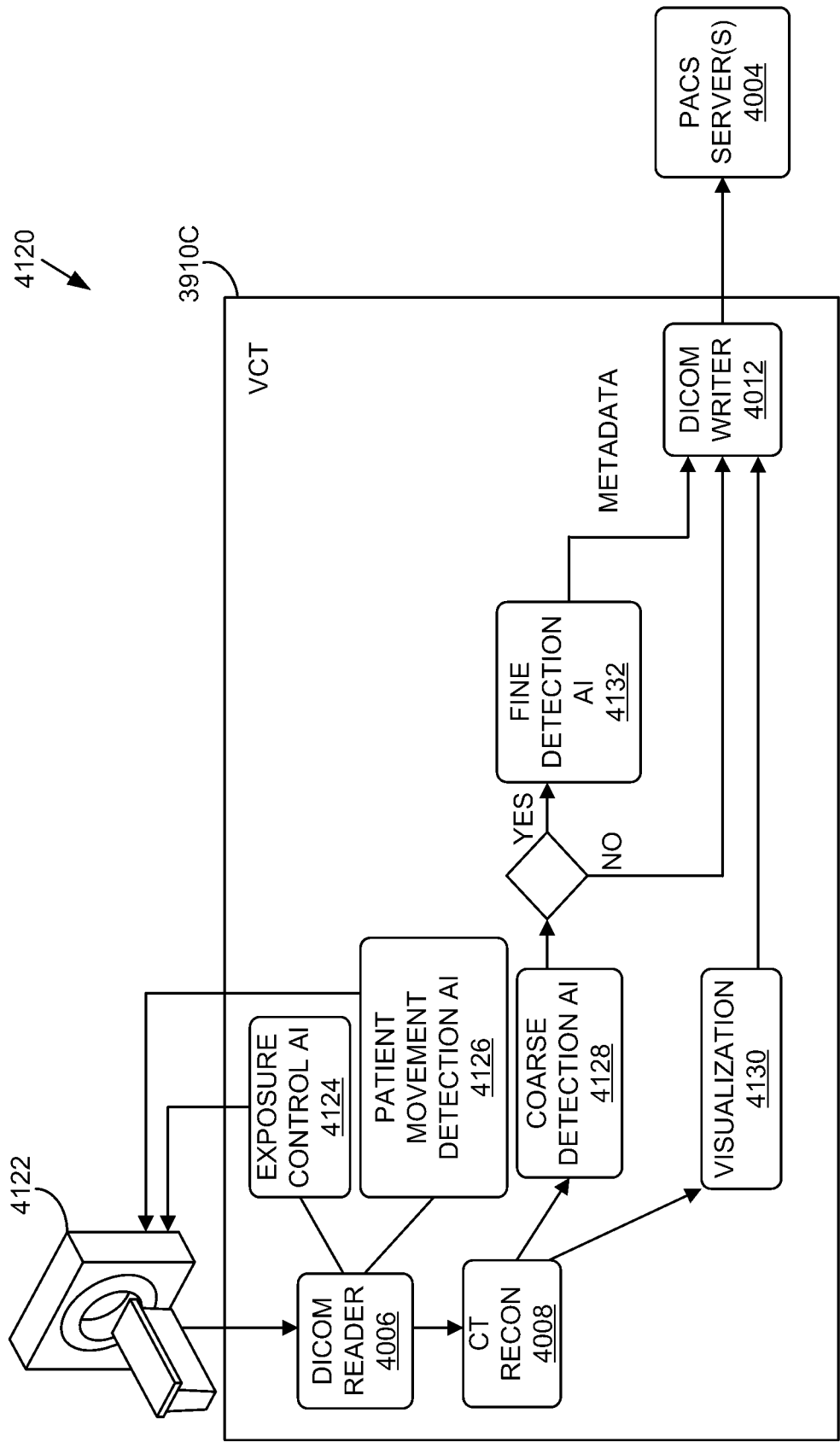


FIG. 41B

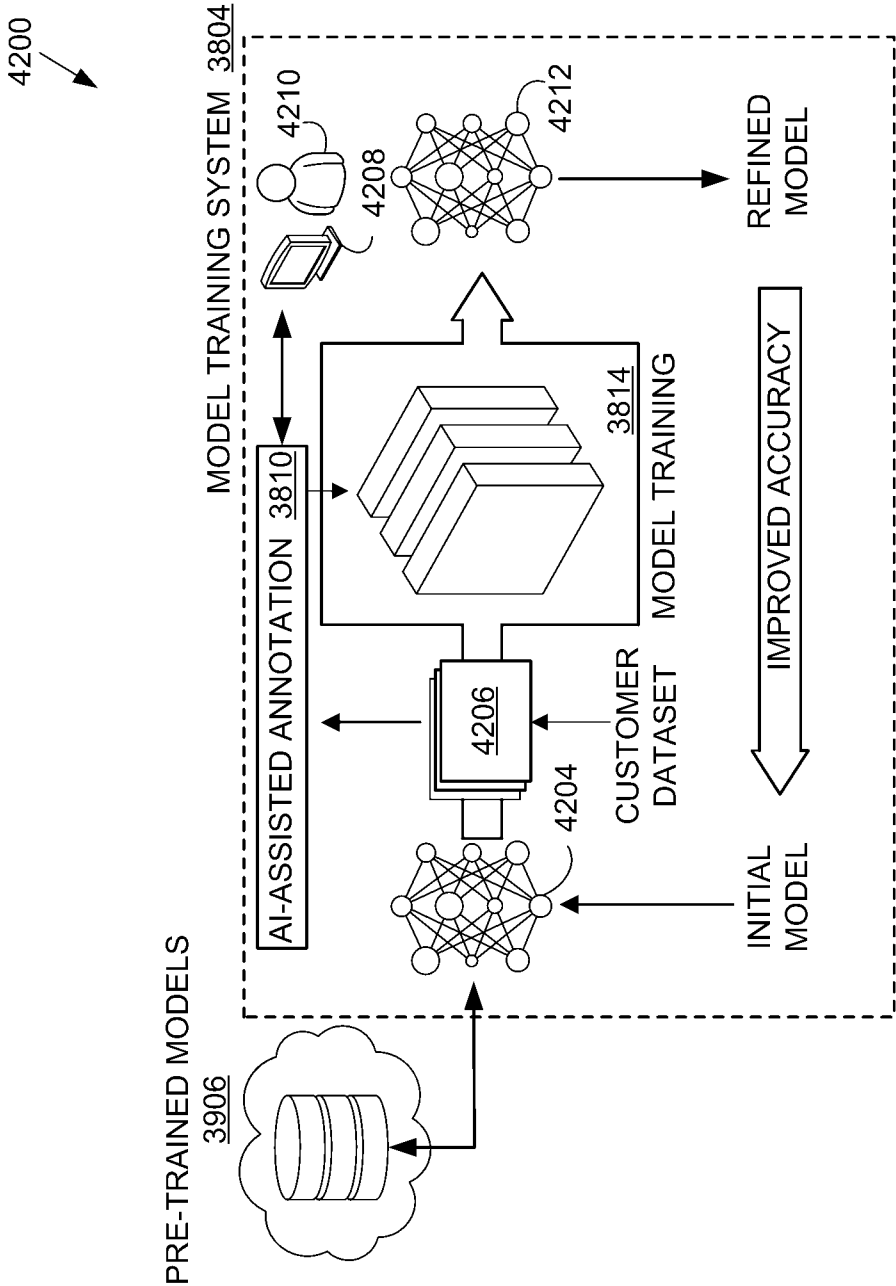


FIG. 42A

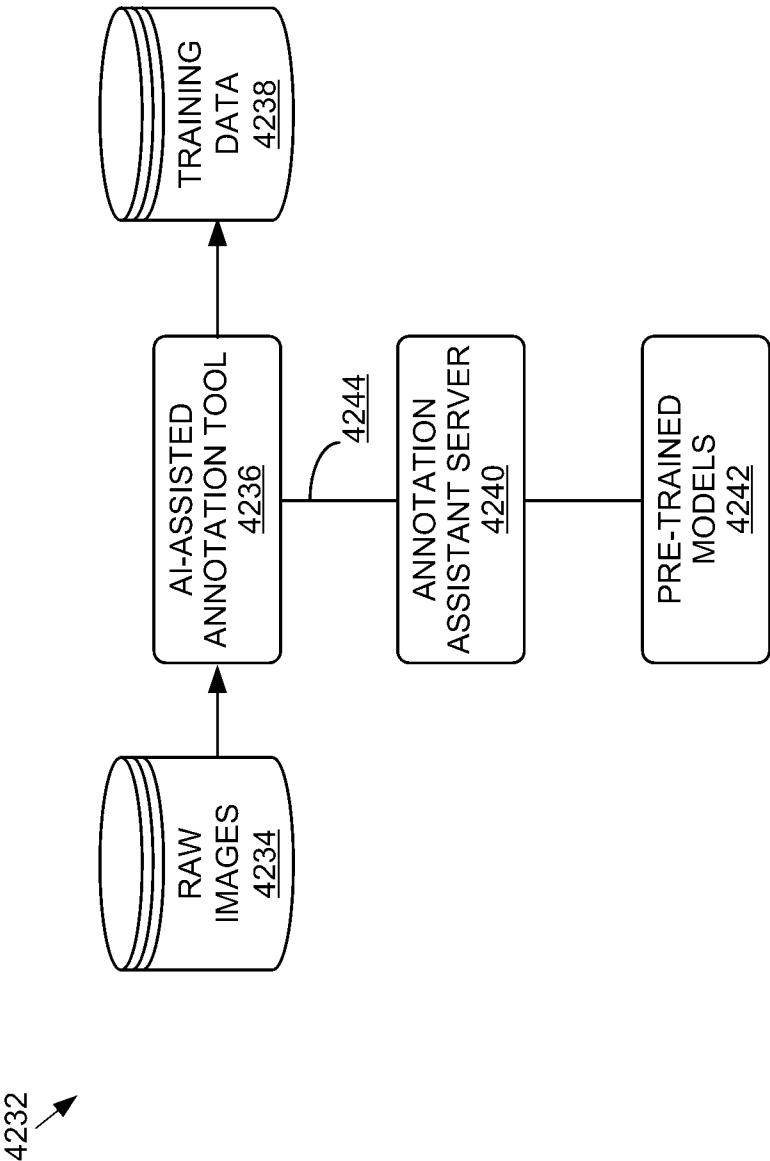


FIG. 42B

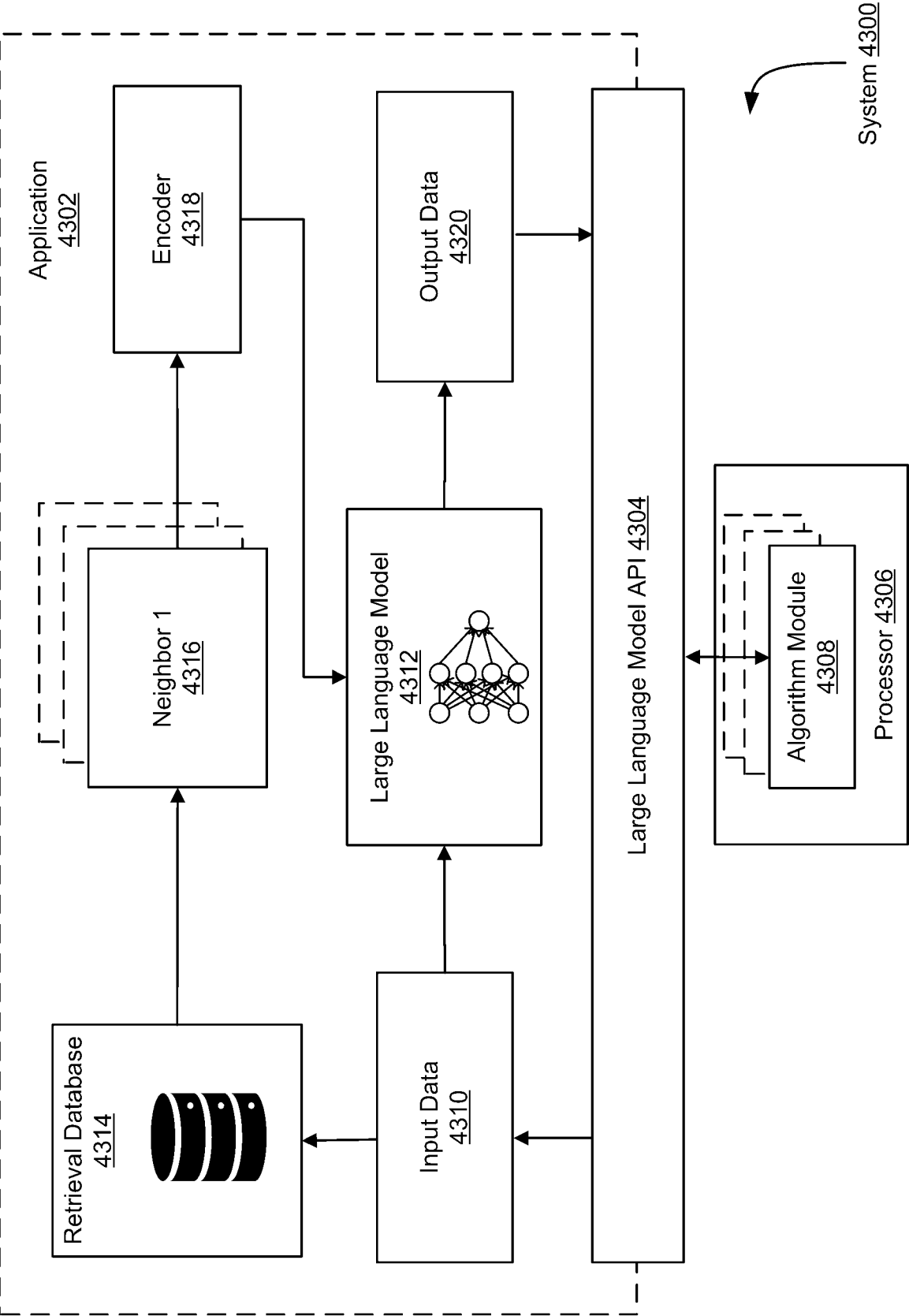


FIG. 43

INTERNATIONAL SEARCH REPORT

International application No.

PCT/CN2023/141220

A. CLASSIFICATION OF SUBJECT MATTER G06N 3/063(2023.01)i According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) IPC: G06N 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) CNABS, CNTXT, VEN, USTXT, EPTXT, WOTXT, CNKI, IEEE, ISI: neural network, hardware, performance, parameter, select, choose, mask, train, metric, layer, prune, sparse, optimize, threshold		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CN 113947185 A (BEIJING DAJIA INTERCONNECTION INFORMATION TECHNOLOGY CO., LTD.) 18 January 2022 (2022-01-18) description, paragraphs [0002]-[0352] and figures 1-10	1-20
A	CN 116306878 A (OPPO GUANGDONG MOBILE TELECOMMUNICATIONS CO., LTD.) 23 June 2023 (2023-06-23) the whole document	1-20
A	CN 116663640 A (SAMSUNG ELECTRONICS CO., LTD. et al.) 29 August 2023 (2023-08-29) the whole document	1-20
A	US 2020026992 A1 (TSINGHUA UNIVERSITY) 23 January 2020 (2020-01-23) the whole document	1-20
☐ 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 “D” document cited by the applicant in the international application “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 21 August 2024		Date of mailing of the international search report 28 August 2024
Name and mailing address of the ISA/CN CHINA NATIONAL INTELLECTUAL PROPERTY ADMINISTRATION 6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088, China		Authorized officer WEI,JingYao Telephone No. (+86) 010-53961340

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.

PCT/CN2023/141220

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
CN	113947185	A	18 January 2022	None			
CN	116306878	A	23 June 2023	None			
CN	116663640	A	29 August 2023	US	2023259775	A1	17 August 2023
				KR	20230123309	A	23 August 2023
				EP	4231203	A1	23 August 2023
US	2020026992	A1	23 January 2020	WO	2018058426	A1	05 April 2018