



US 20200394458A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2020/0394458 A1**

Yu et al.

(43) **Pub. Date:** **Dec. 17, 2020**

(54) **WEAKLY-SUPERVISED OBJECT
DETECTION USING ONE OR MORE
NEURAL NETWORKS**

(71) Applicant: **Nvidia Corporation**, Santa Clara, CA
(US)

(72) Inventors: **Zhidong Yu**, Santa Clara, CA (US);
Jason Ren, Davis, CA (US); **Xiaodong Yang**, Fremont, CA (US); **Ming-Yu Liu**, San Jose, CA (US); **Jan Kautz**, Lexington, MA (US)

(21) Appl. No.: **16/443,346**

(22) Filed: **Jun. 17, 2019**

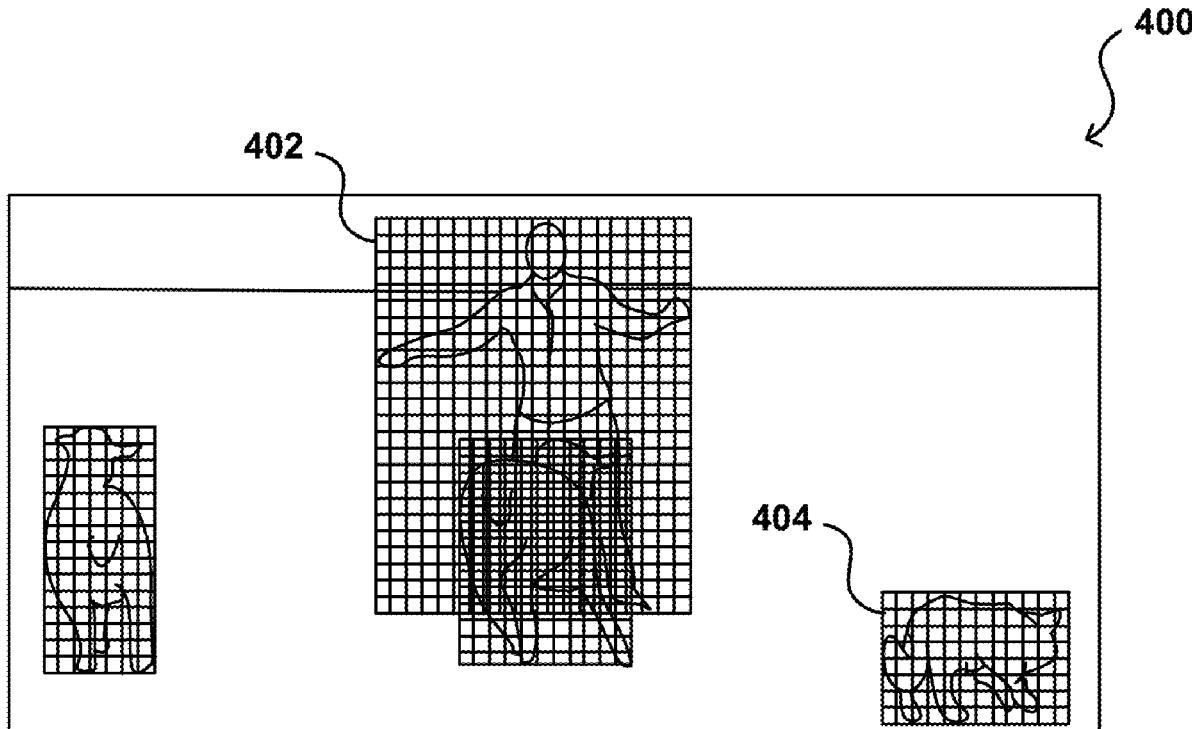
Publication Classification

(51) **Int. Cl.**
G06K 9/62 (2006.01)
G06N 3/08 (2006.01)
G06N 3/04 (2006.01)

(52) **U.S. Cl.**
CPC **G06K 9/6256** (2013.01); **G06K 9/6262** (2013.01); **G06N 3/0454** (2013.01); **G06N 3/08** (2013.01); **G06K 9/623** (2013.01)

(57) **ABSTRACT**

Apparatuses, systems, and techniques to detect object in images including digital representations of those objects. In at least one embodiment, one or more objects are detected in an image based, at least in part, on one or more pseudo-labels corresponding to said one or more objects.



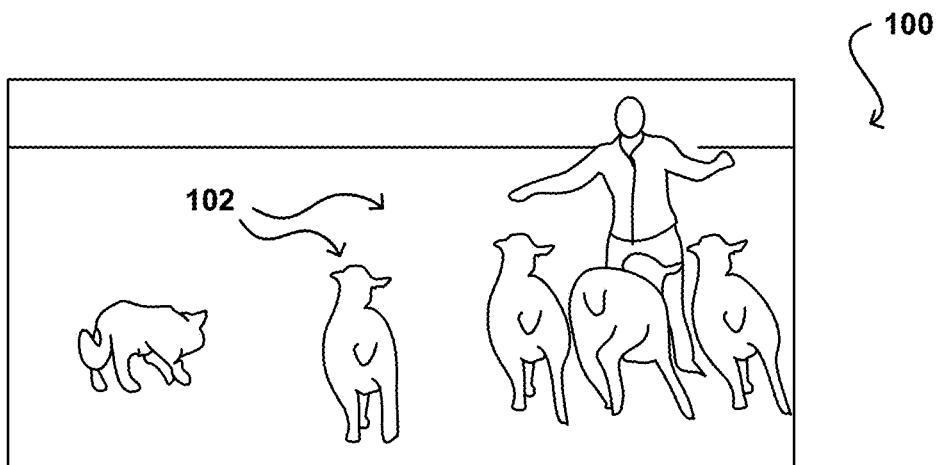


FIG. 1A

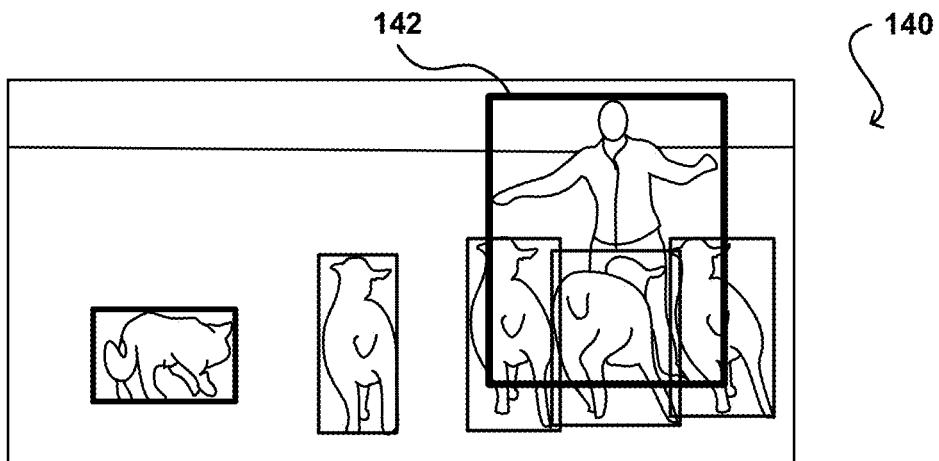


FIG. 1B

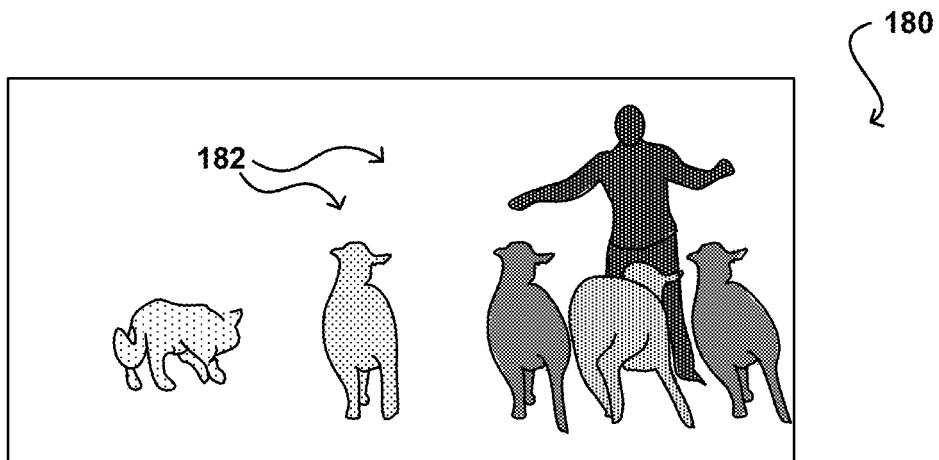


FIG. 1C

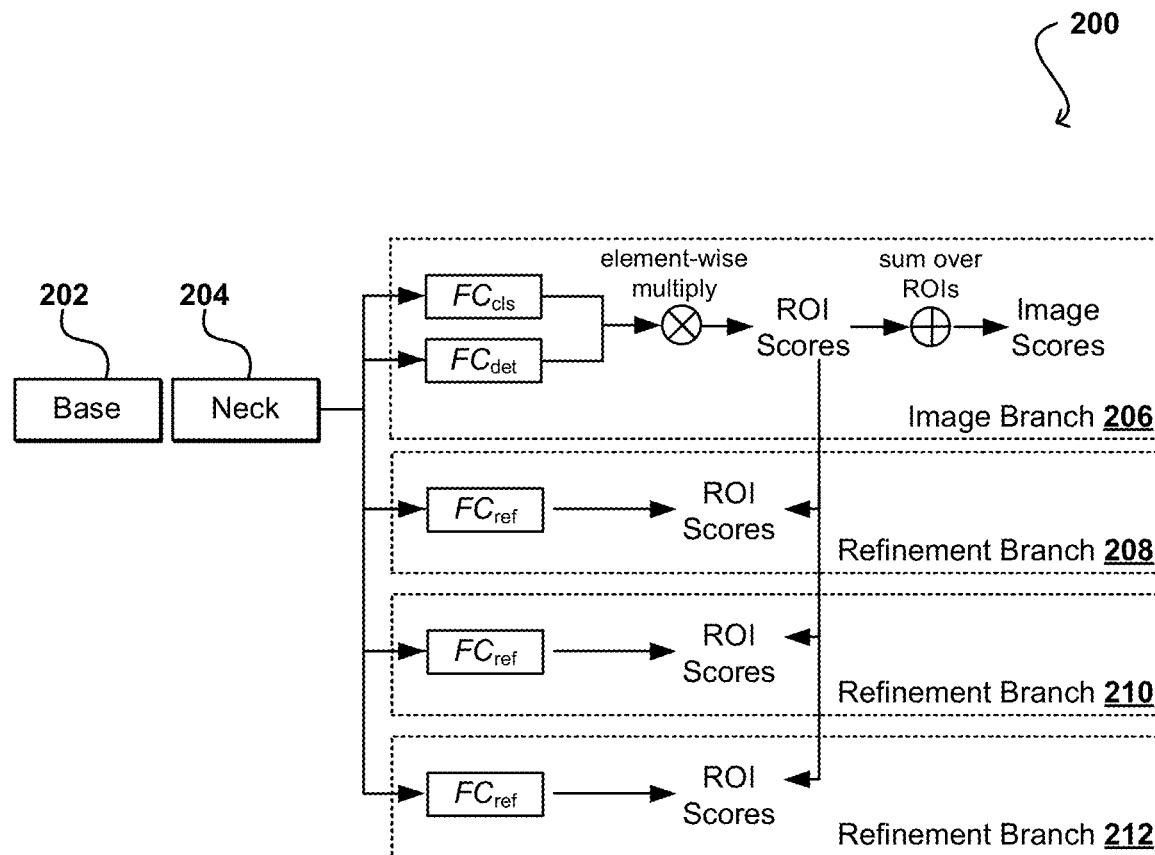


FIG. 2

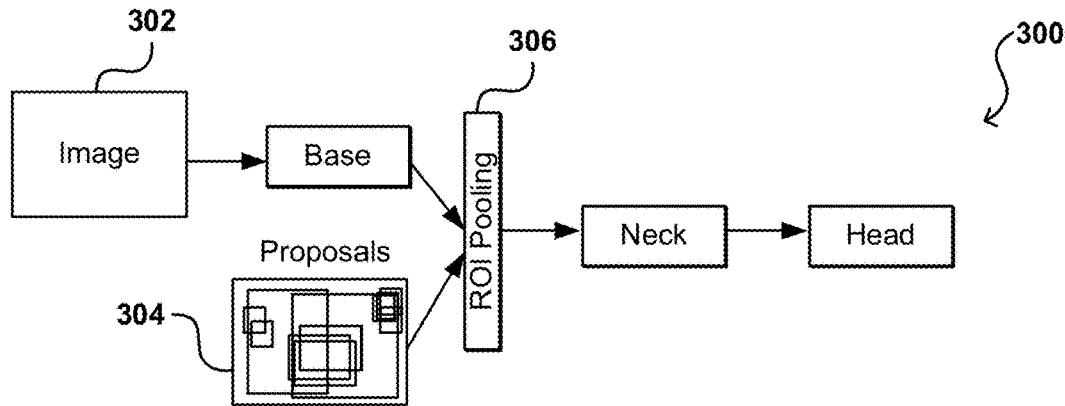


FIG. 3A

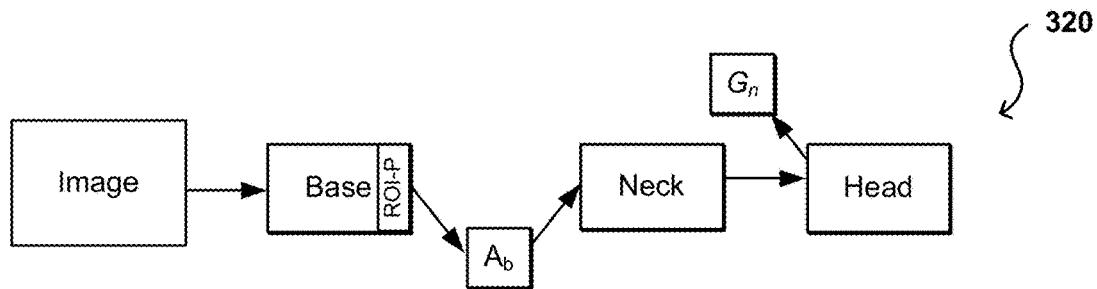


FIG. 3B

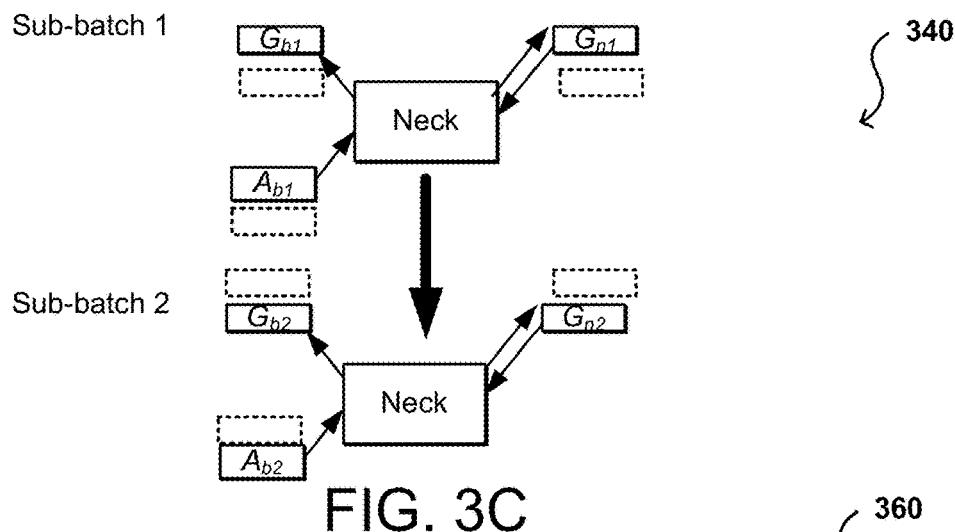


FIG. 3C

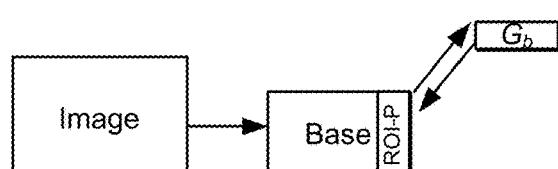


FIG. 3D

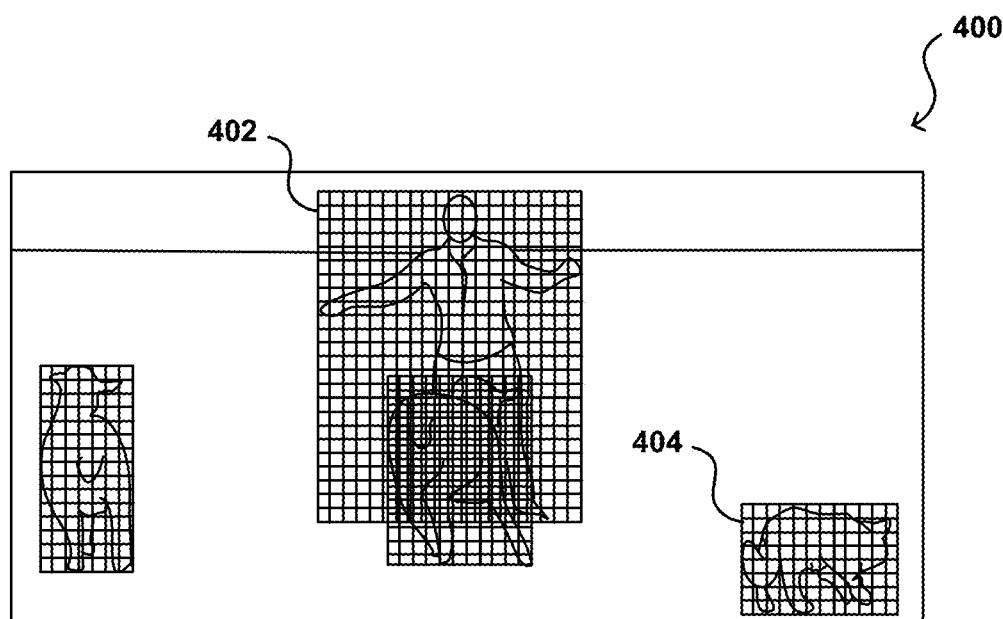


FIG. 4A

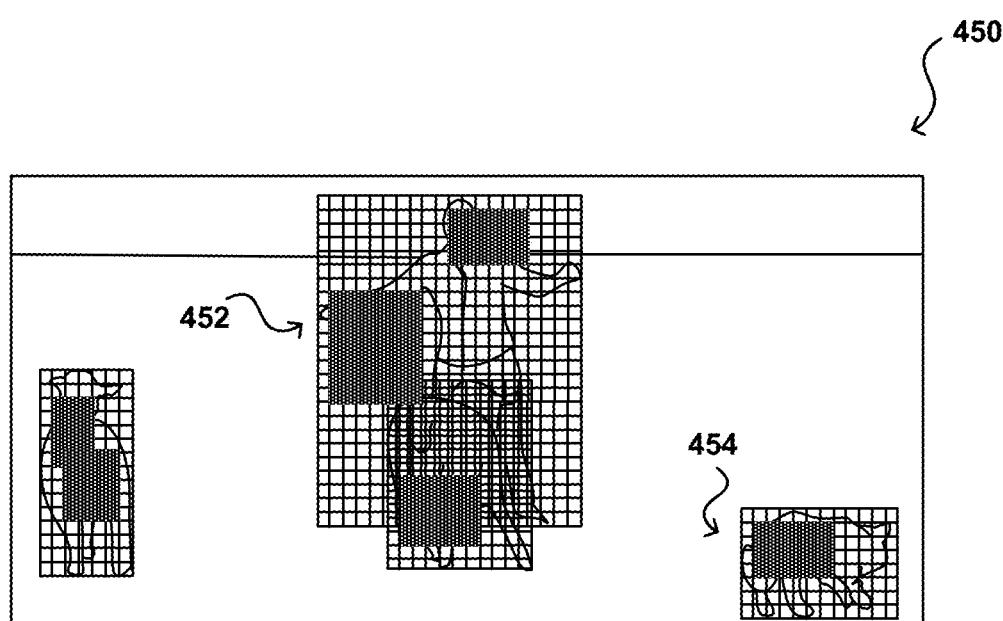


FIG. 4B

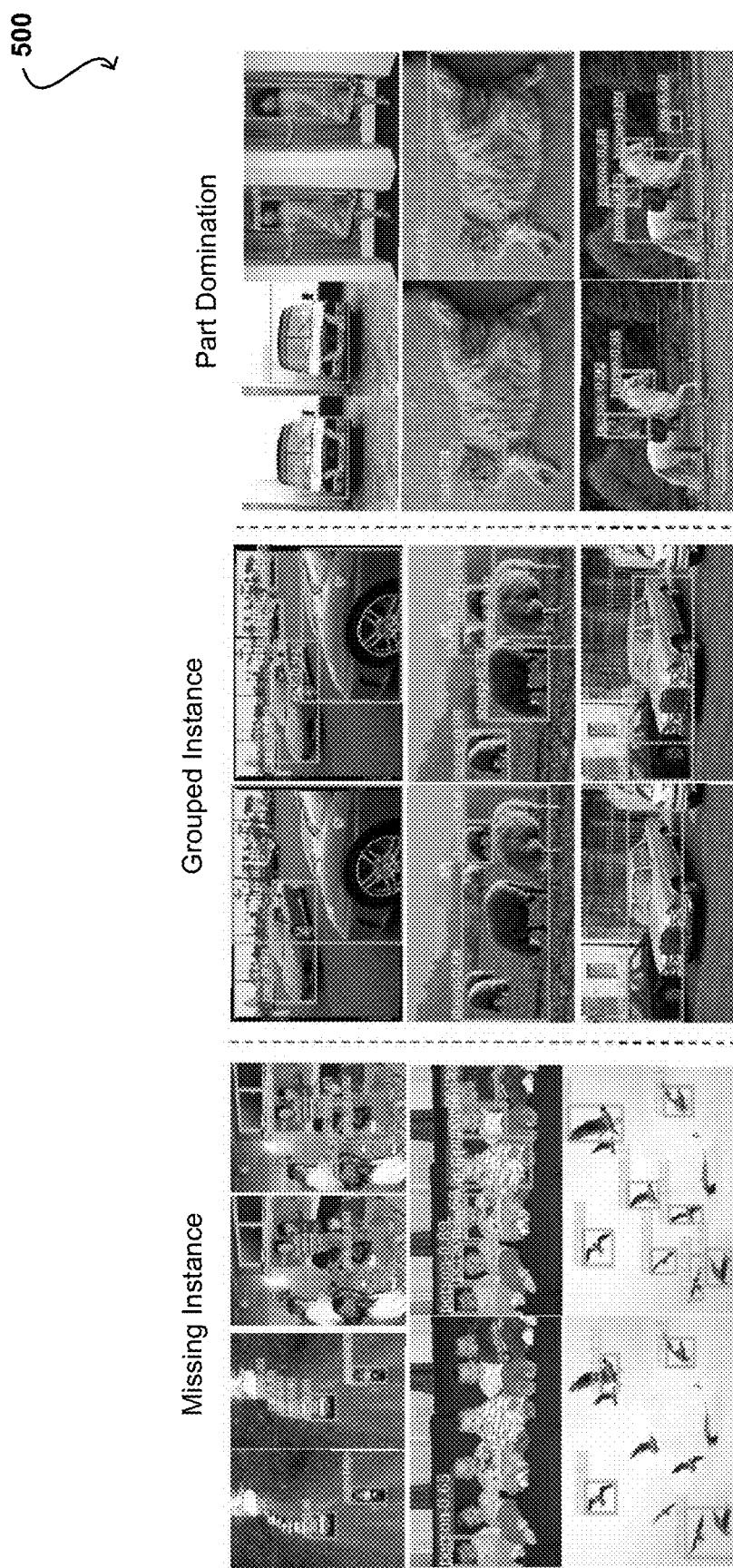


FIG. 5

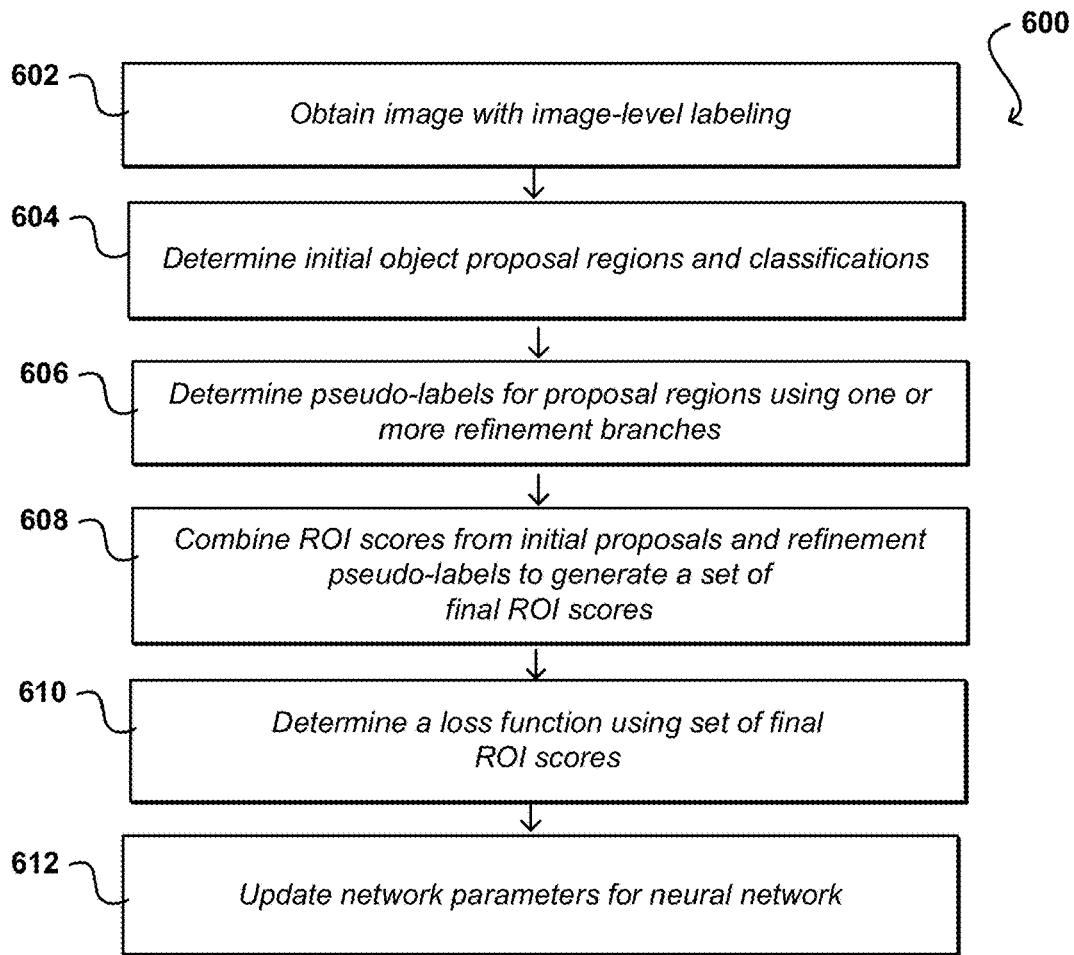


FIG. 6A

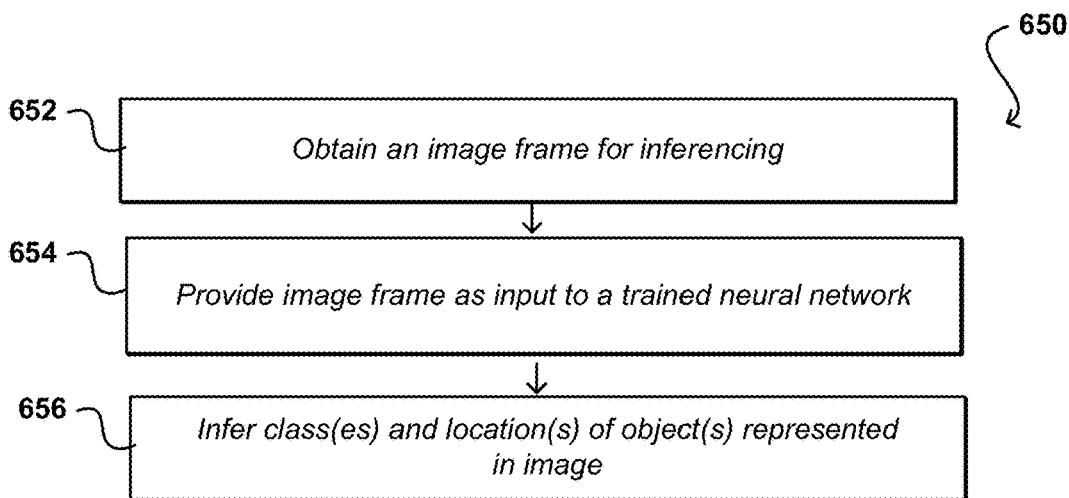


FIG. 6B

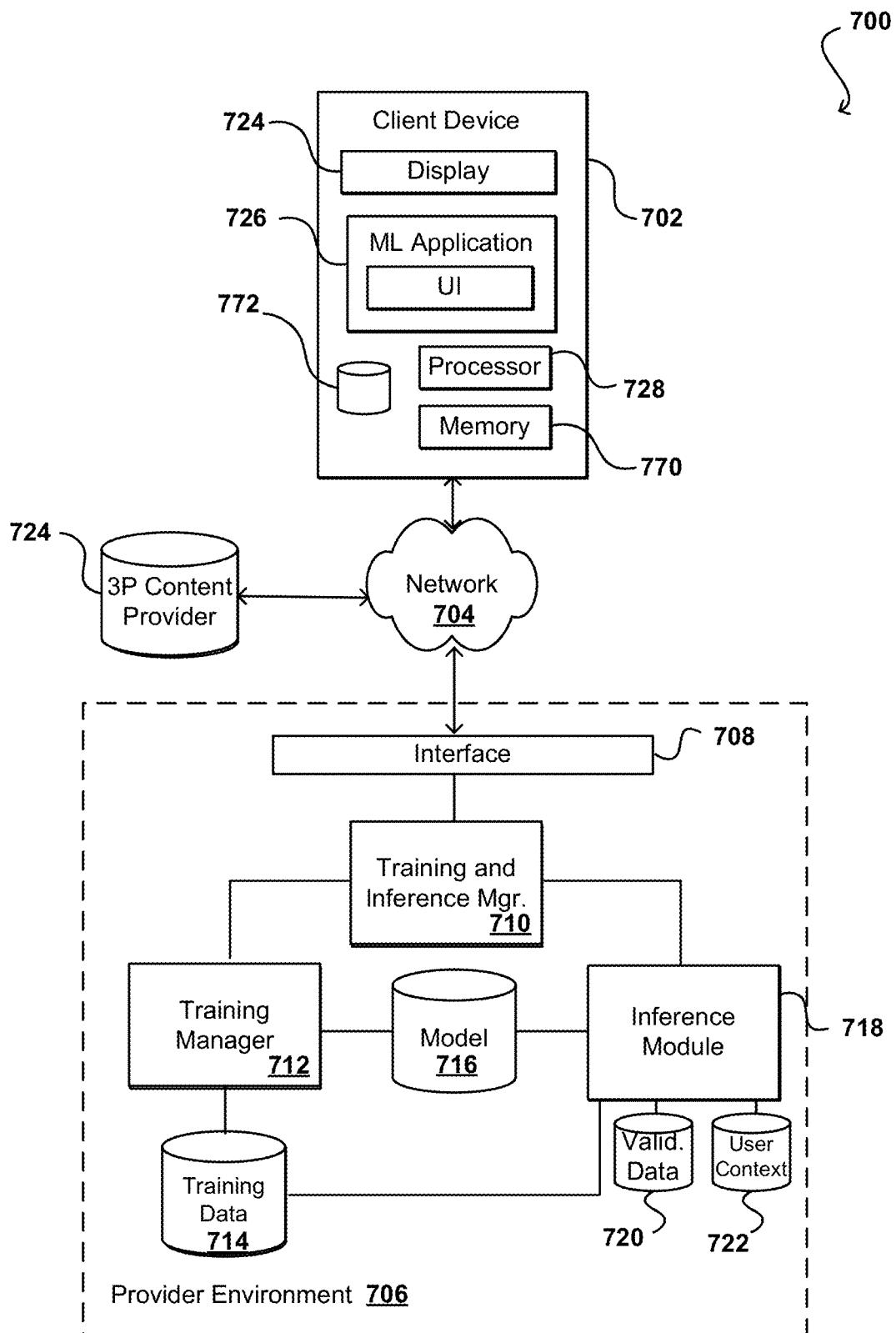


FIG. 7

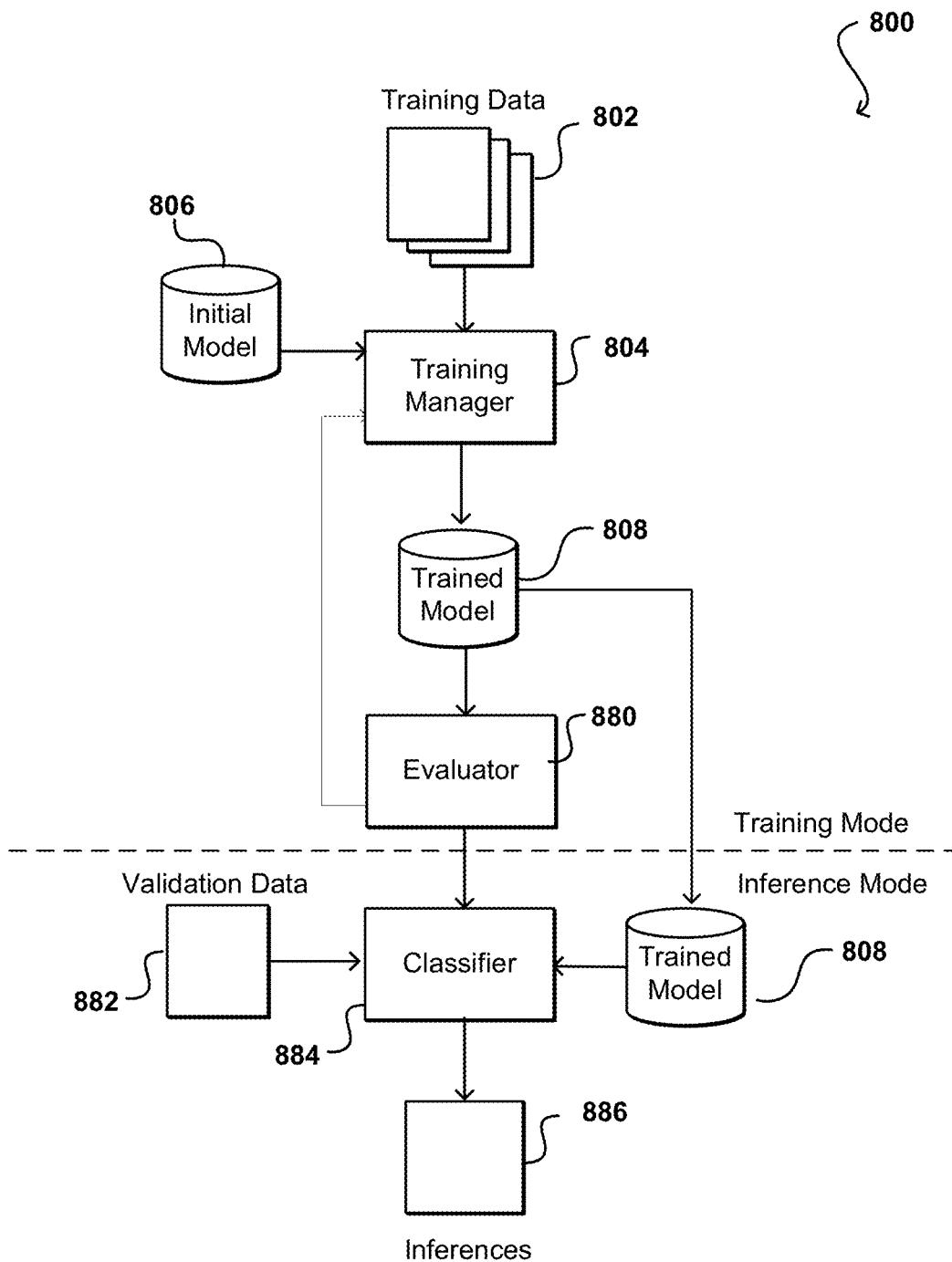


FIG. 8

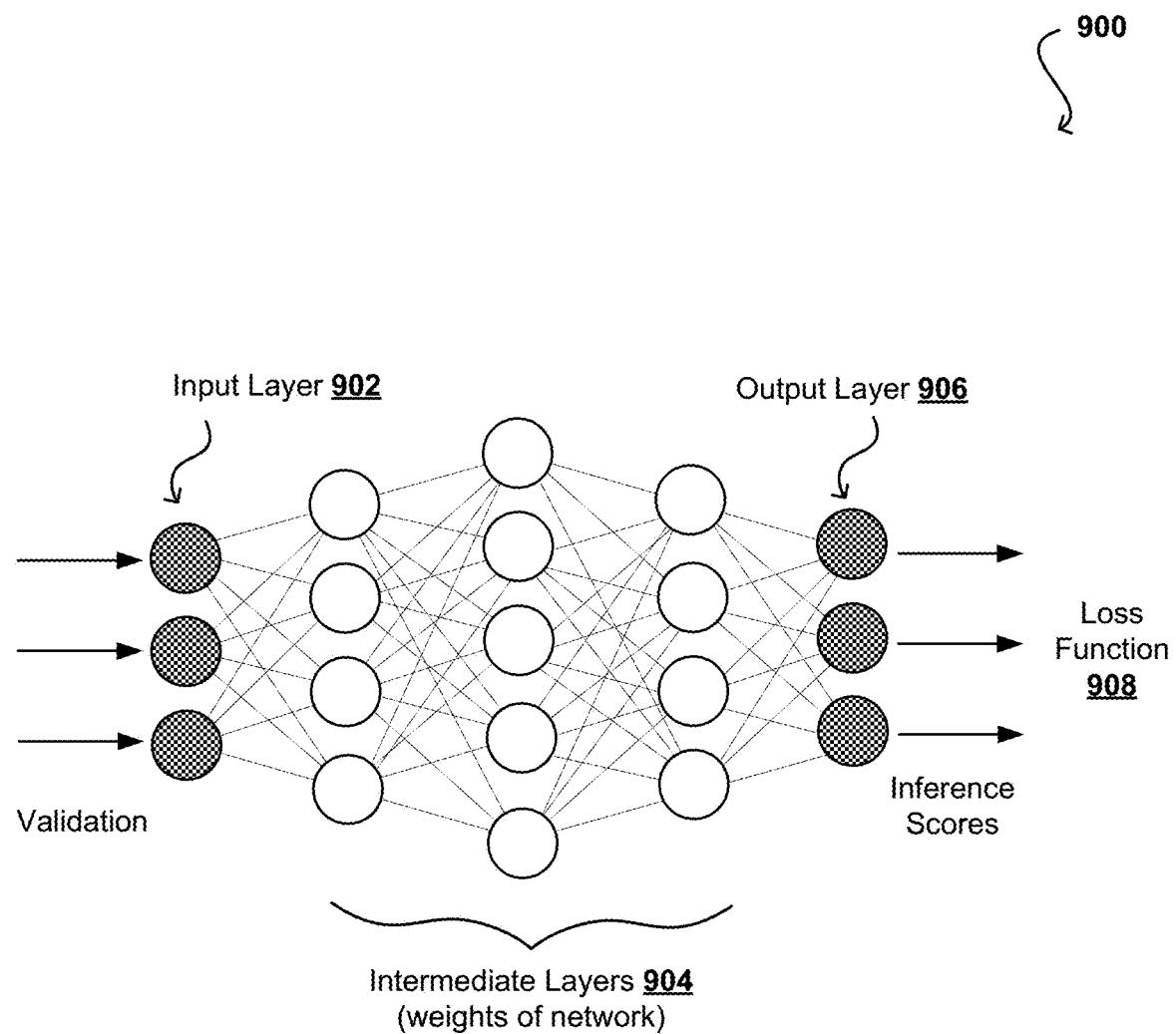


FIG. 9

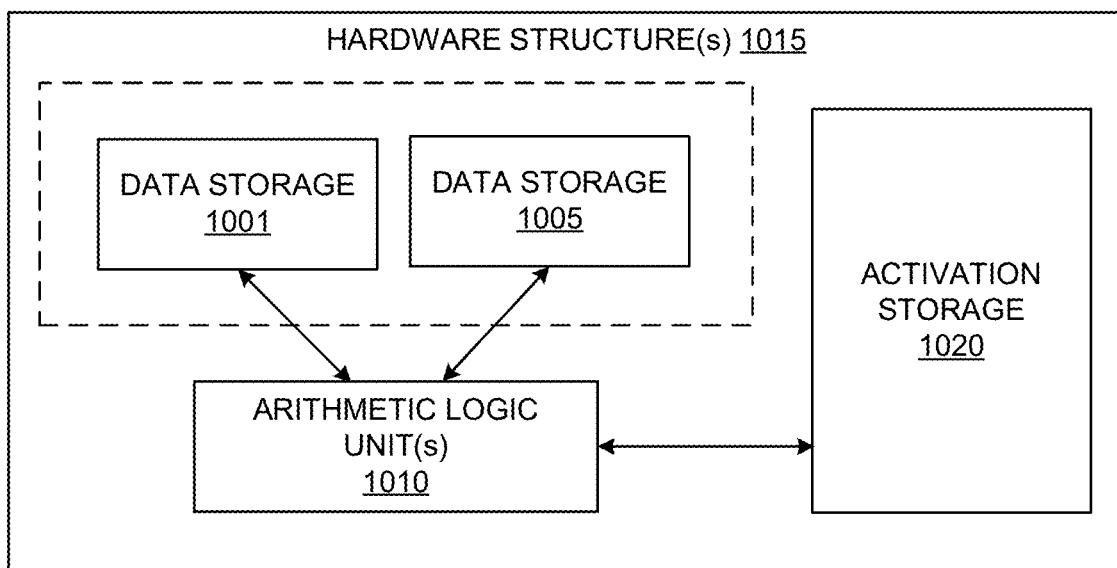


FIG. 10

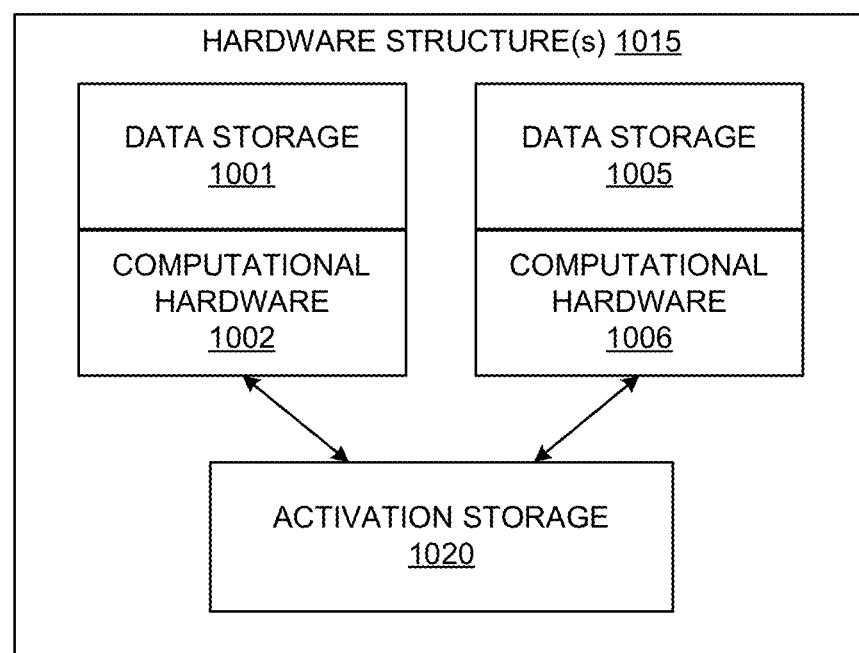


FIG. 11

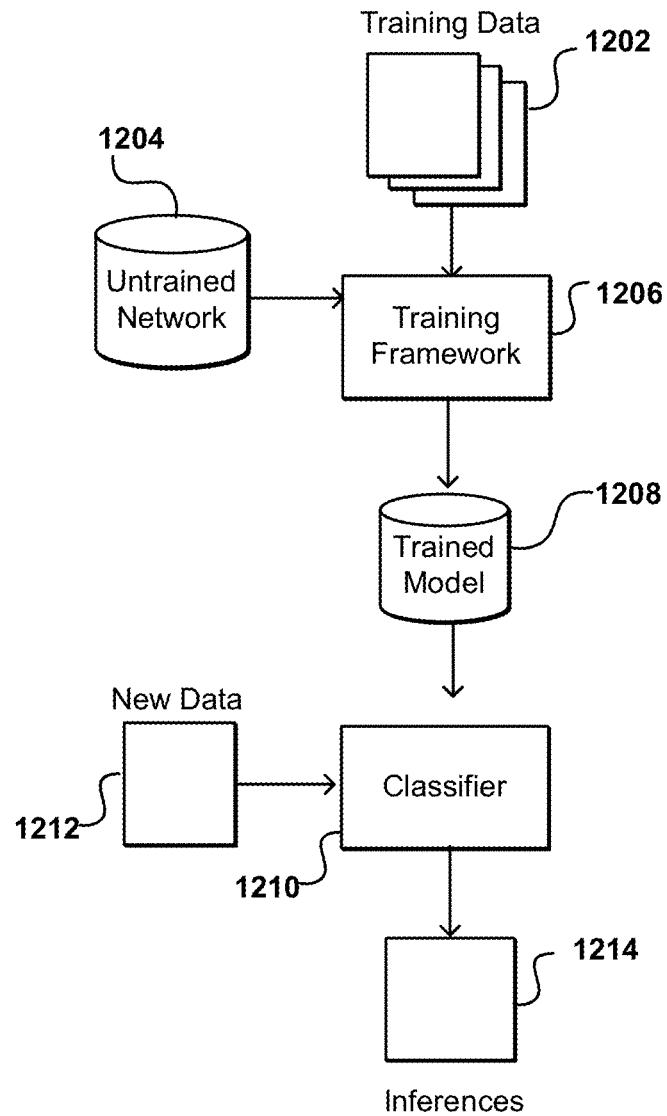


FIG. 12

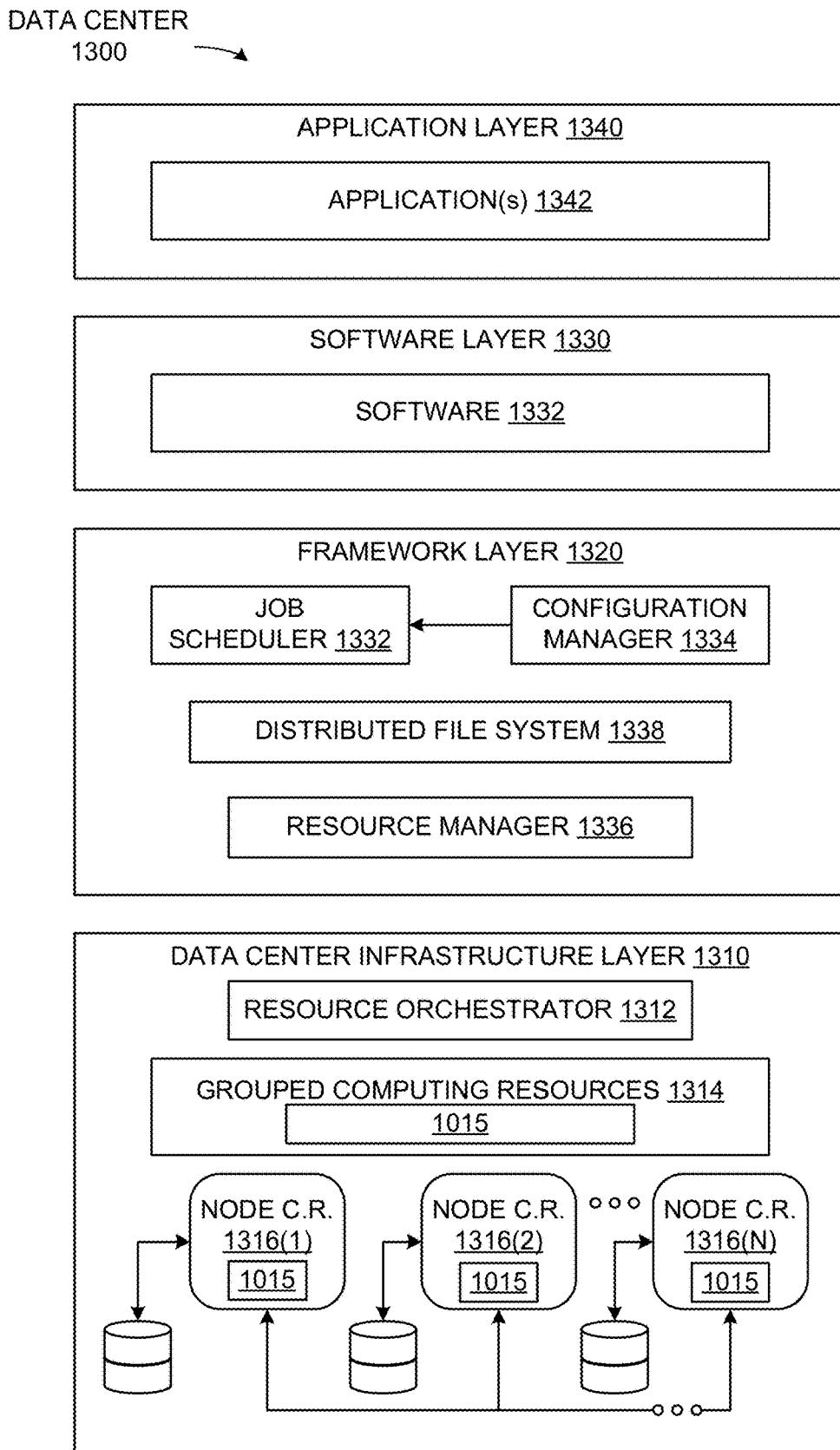


FIG. 13

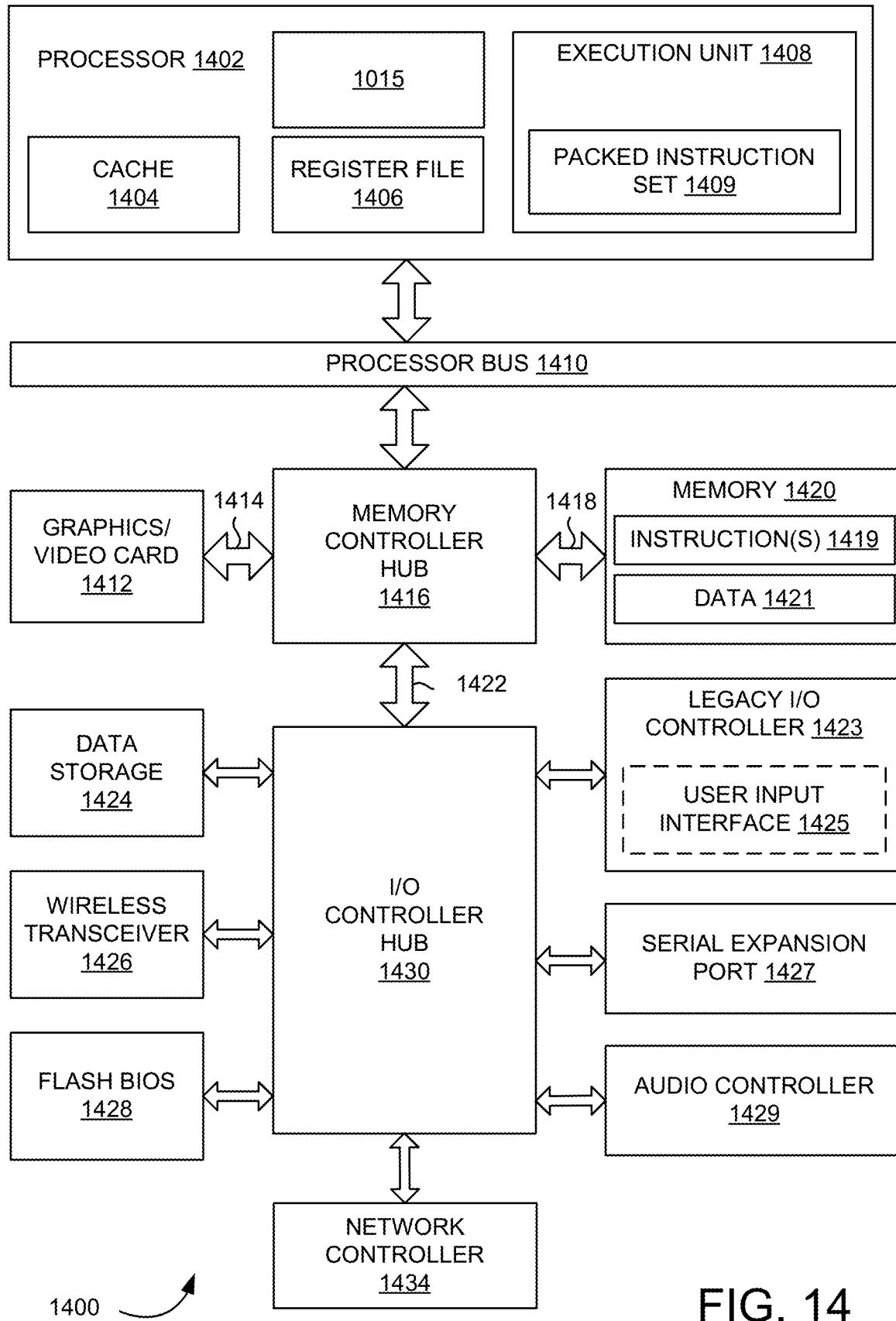


FIG. 14

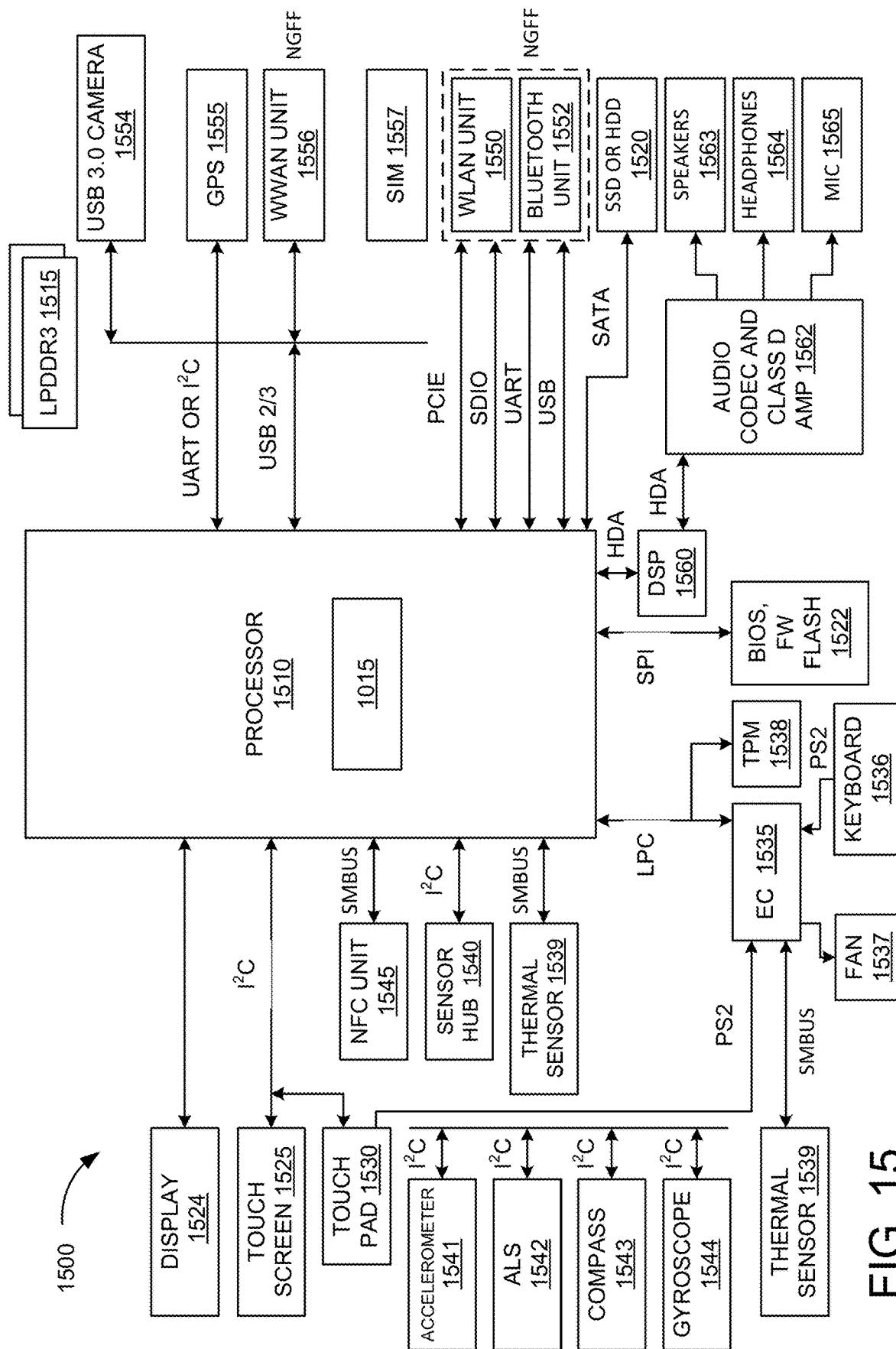


FIG. 15

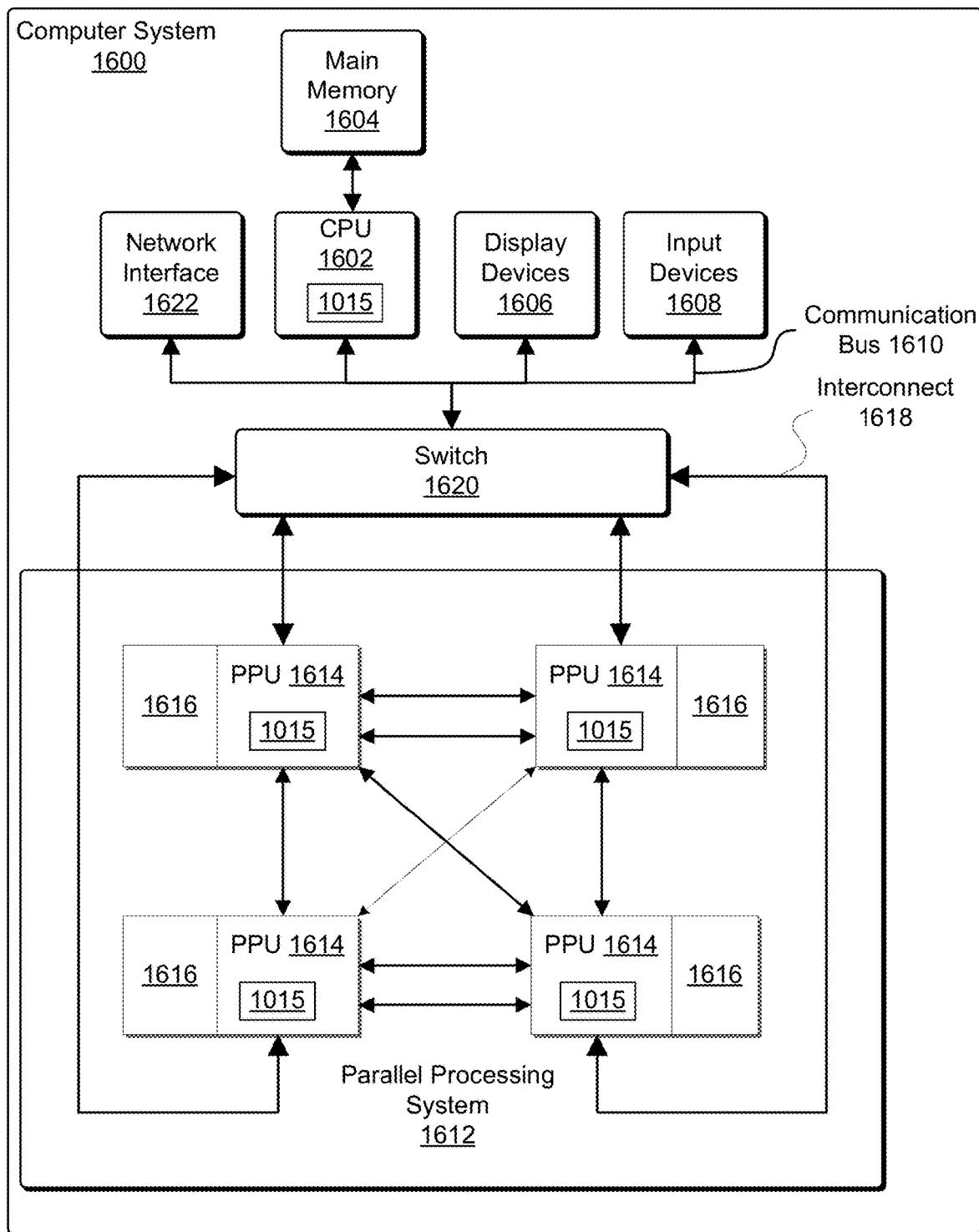


FIG. 16

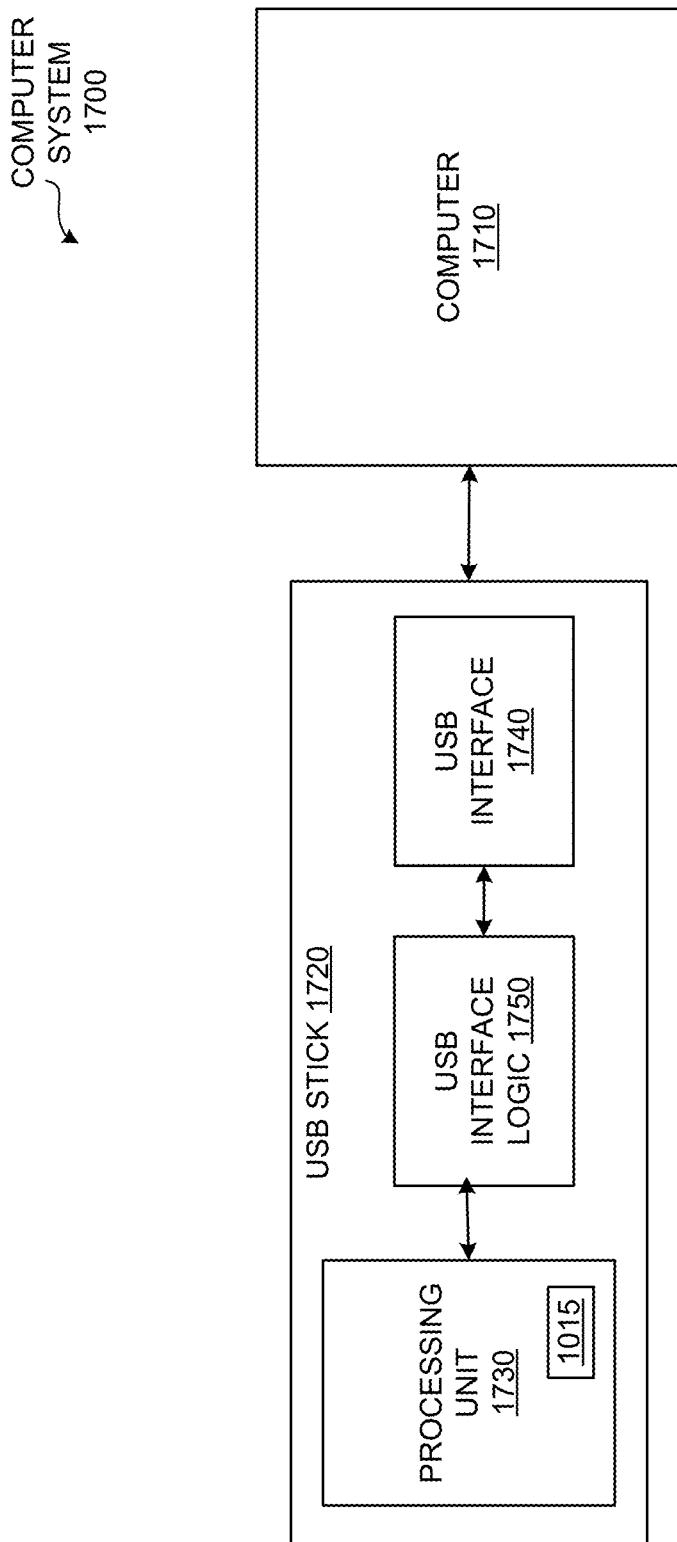


FIG. 17

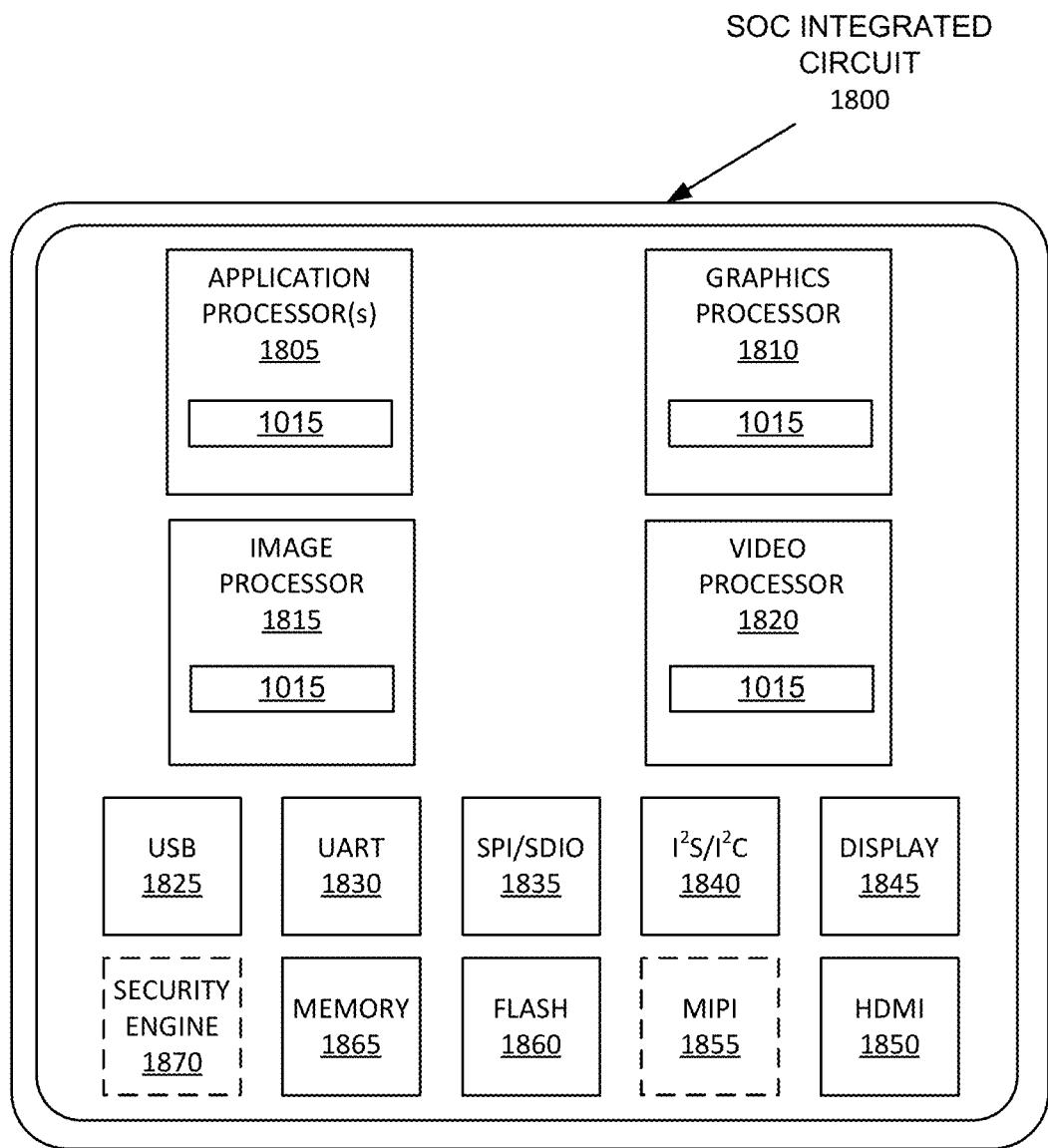


FIG. 18

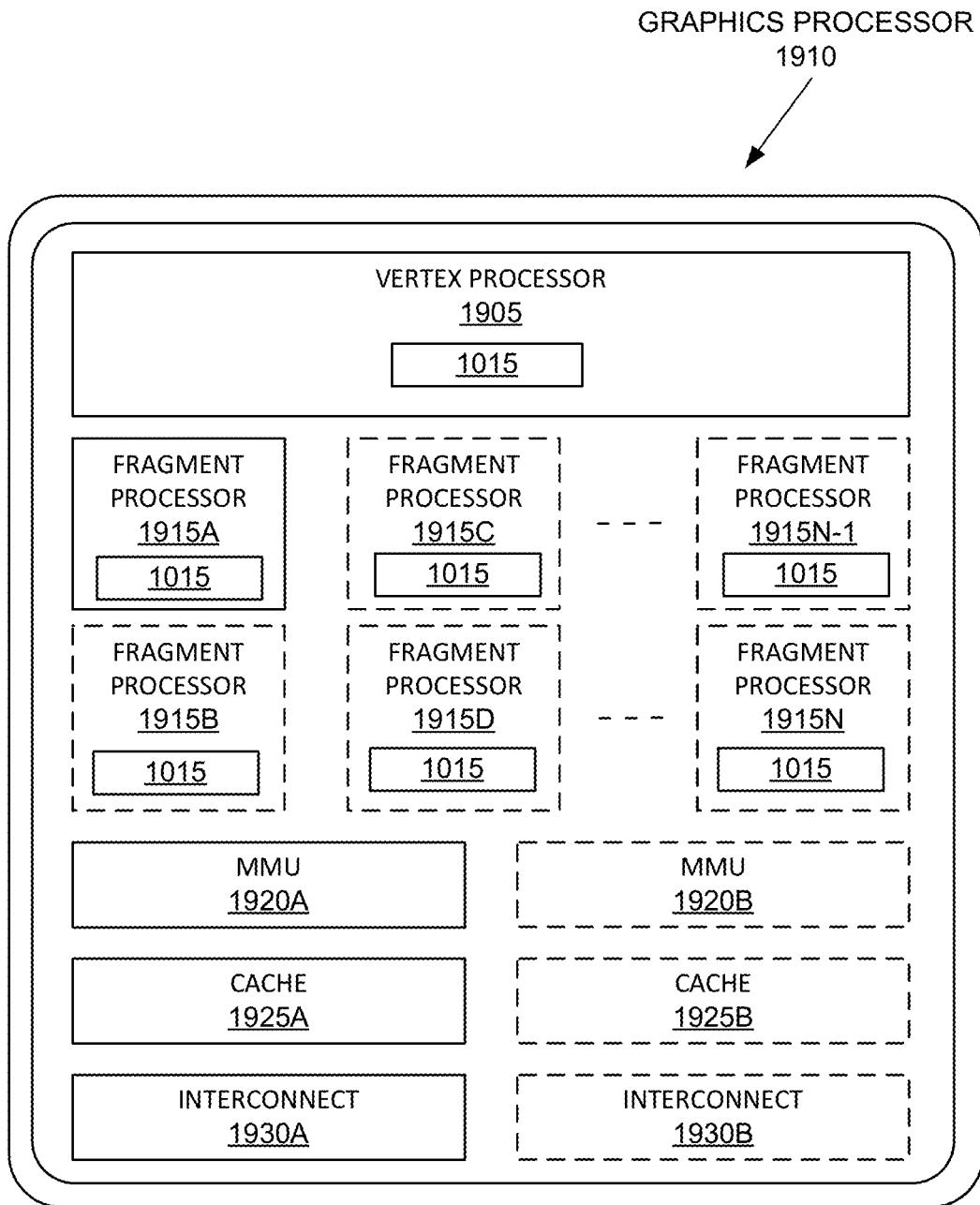


FIG. 19A

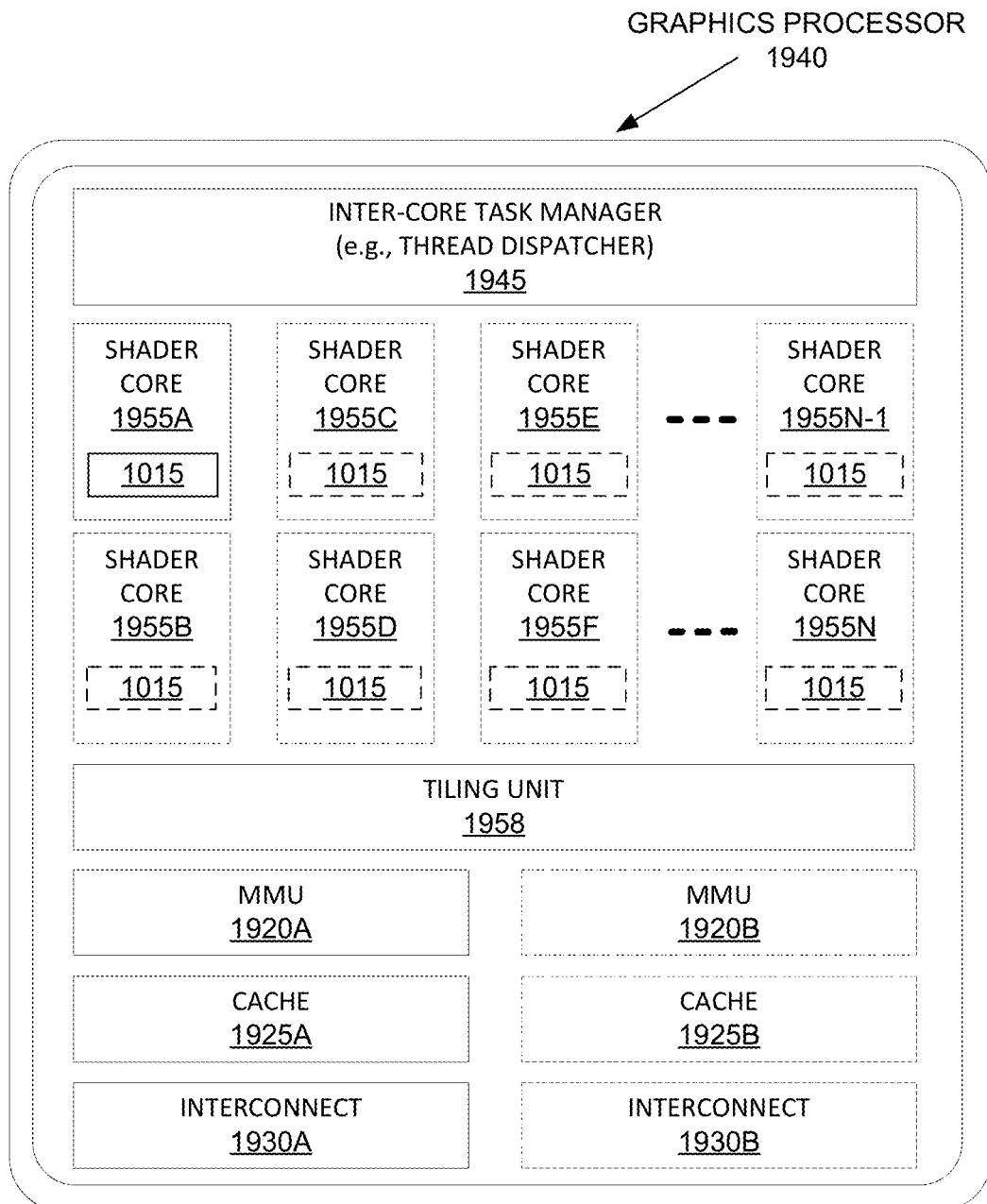


FIG. 19B

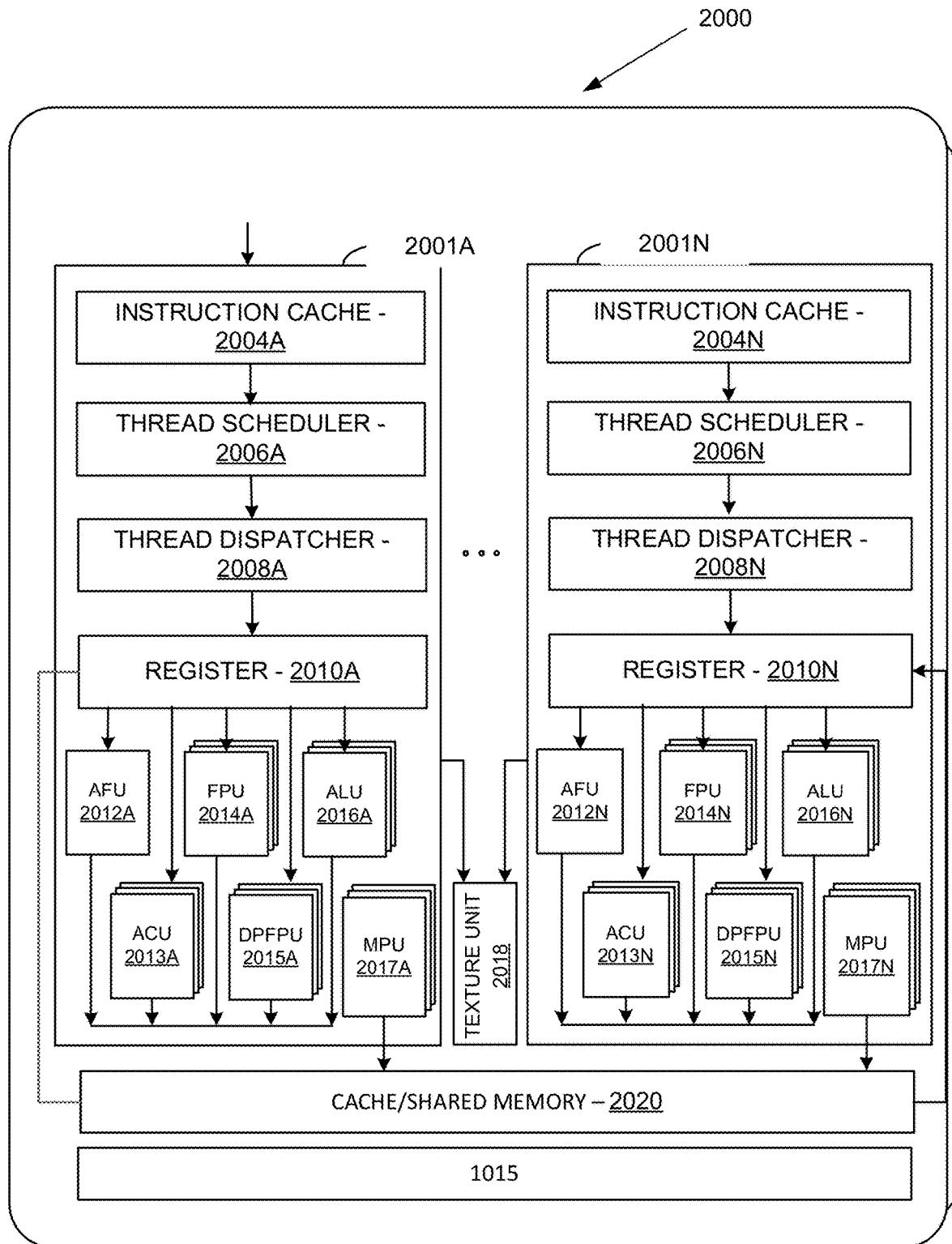


FIG. 20A

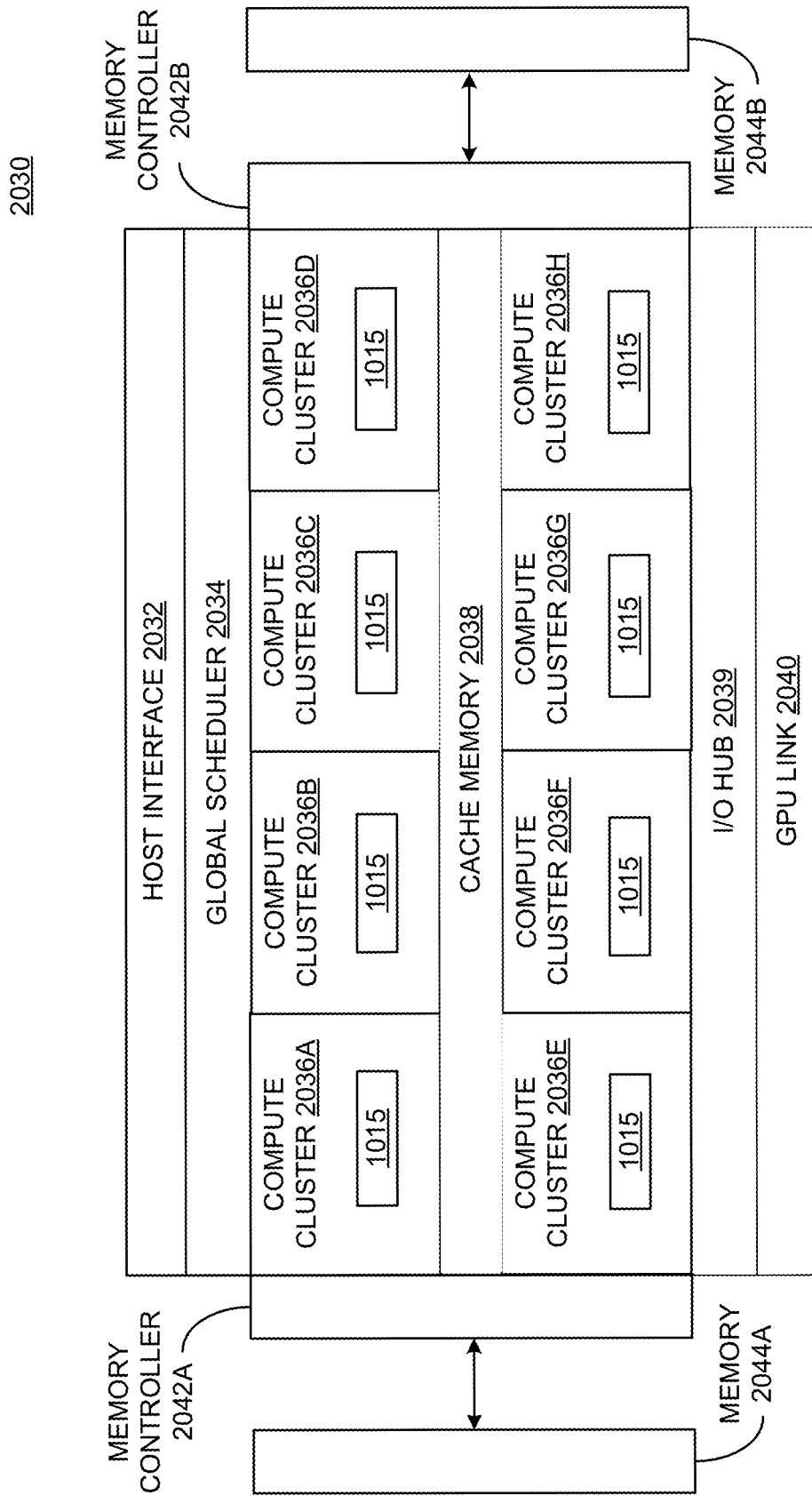


FIG. 20B

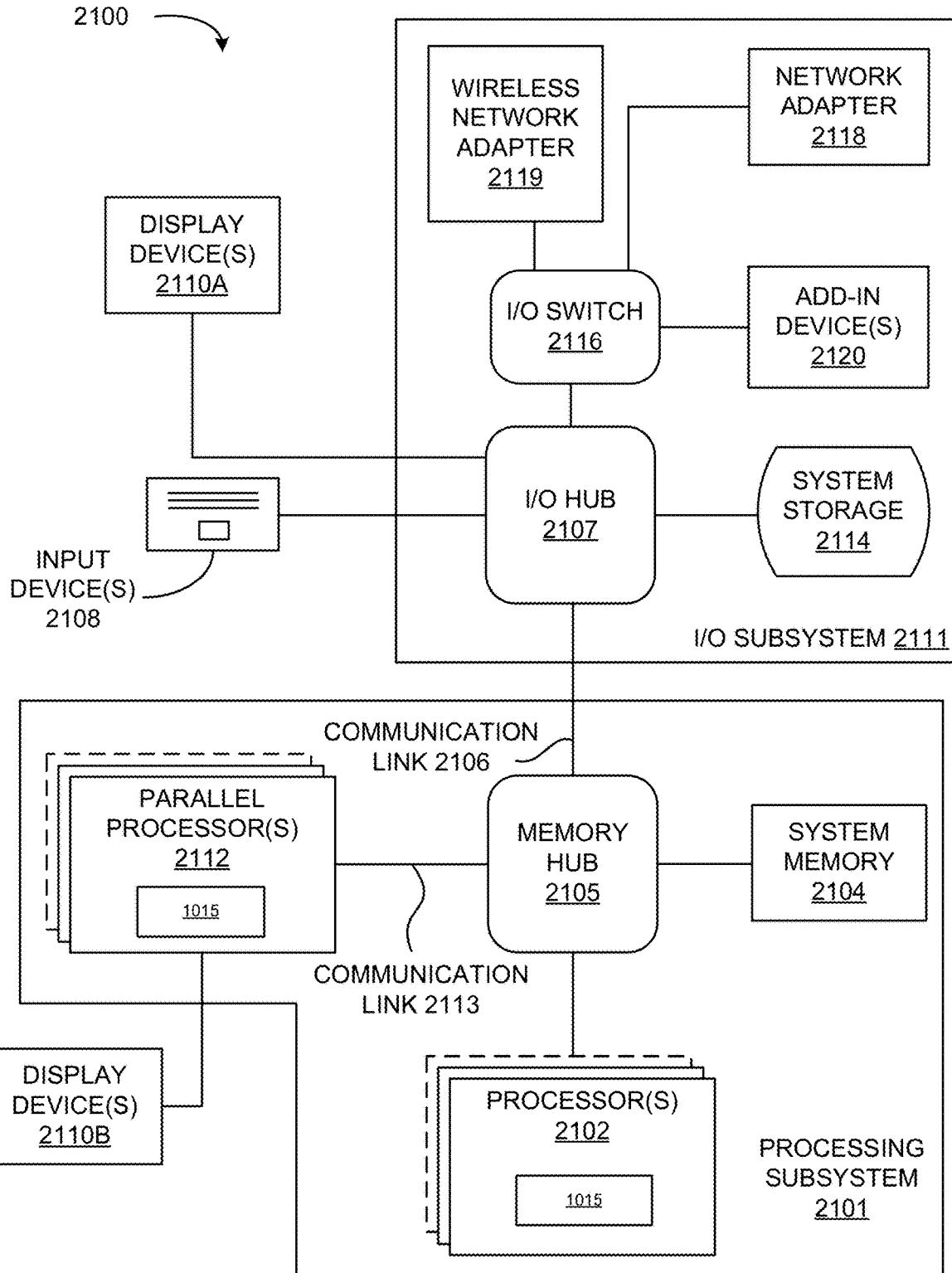


FIG. 21

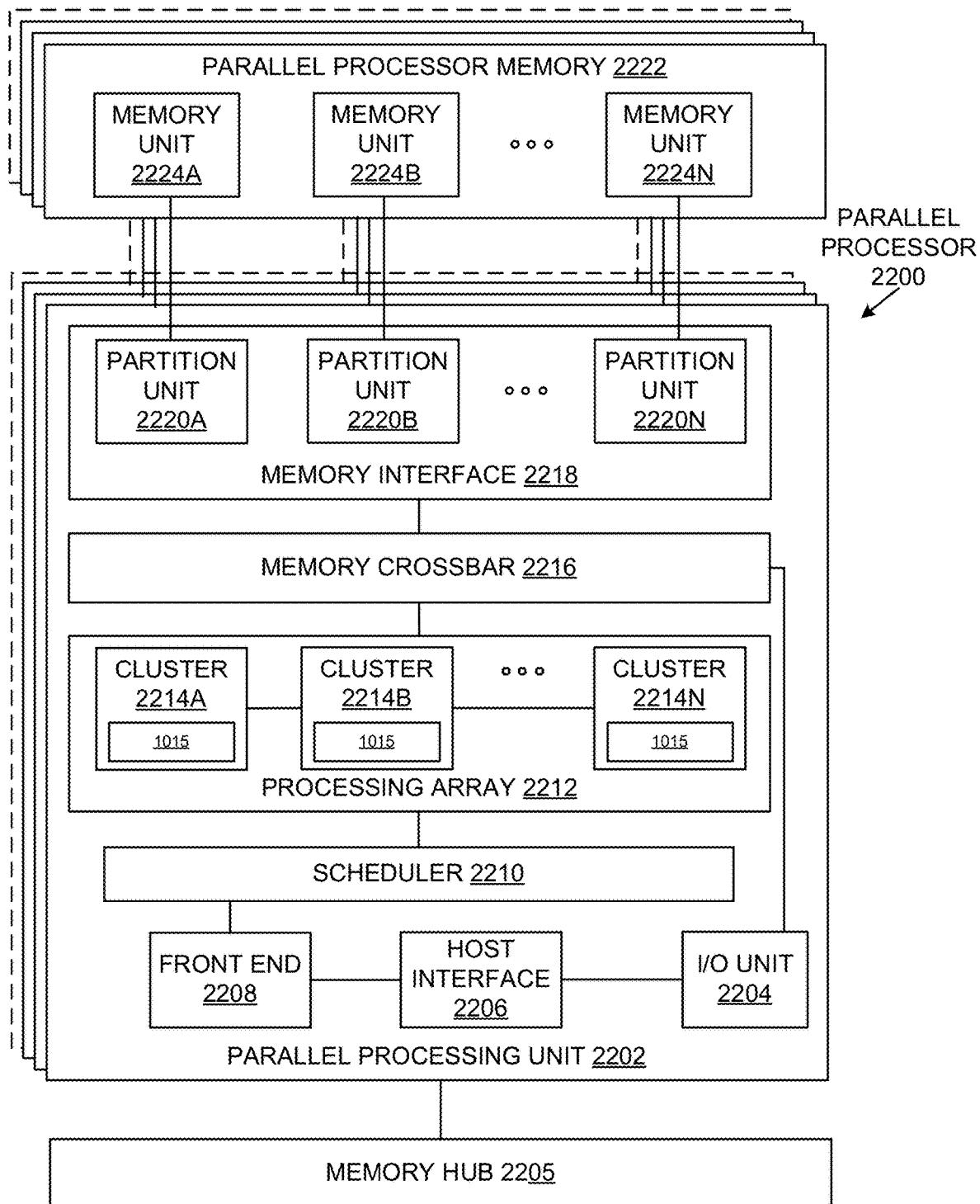


FIG. 22

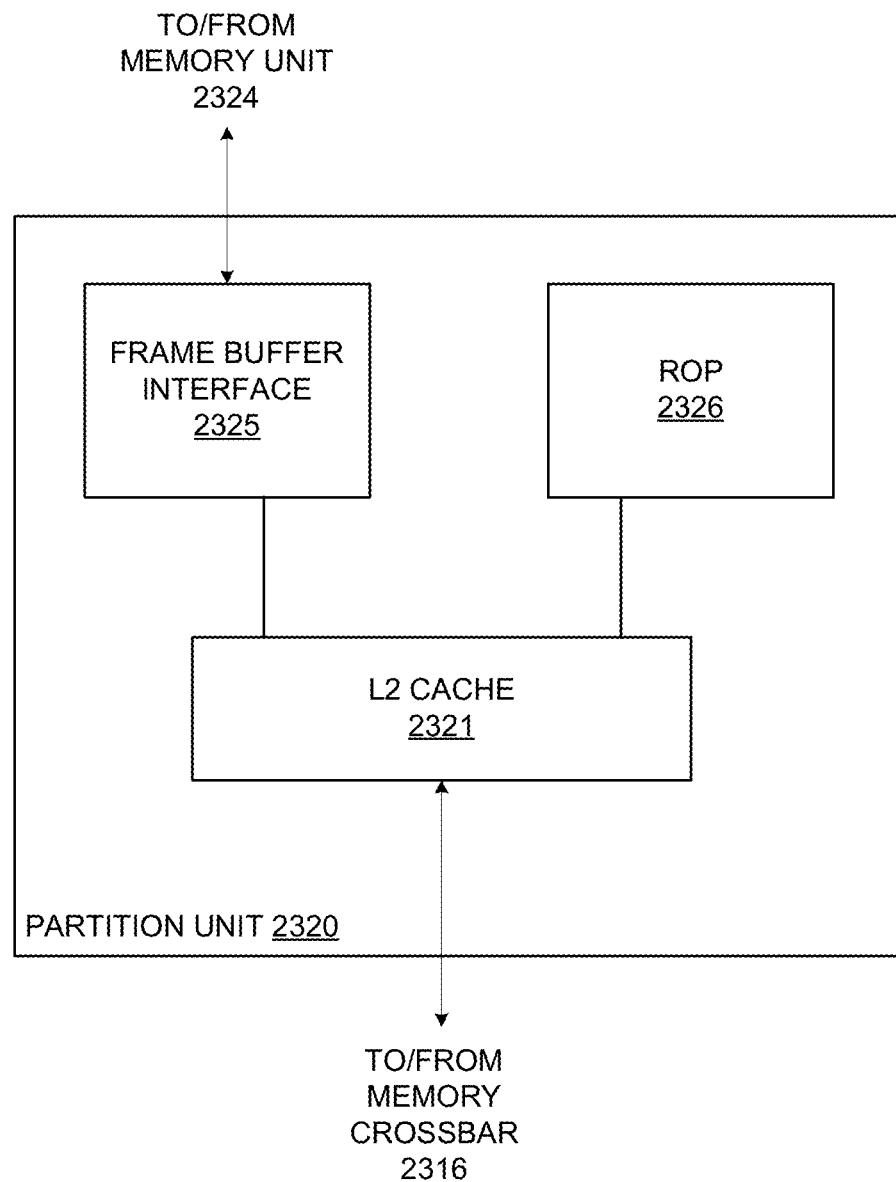


FIG. 23

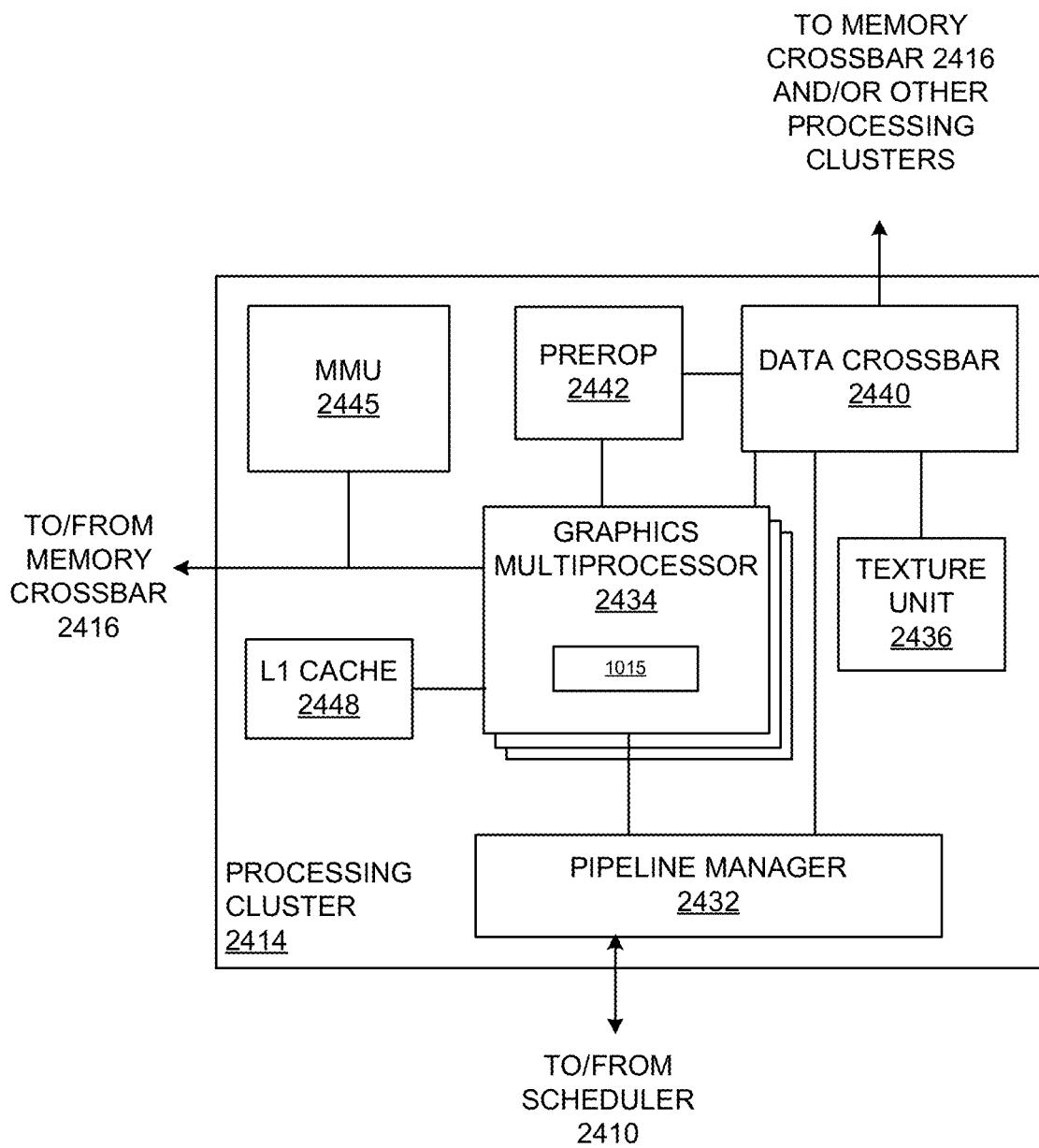


FIG. 24

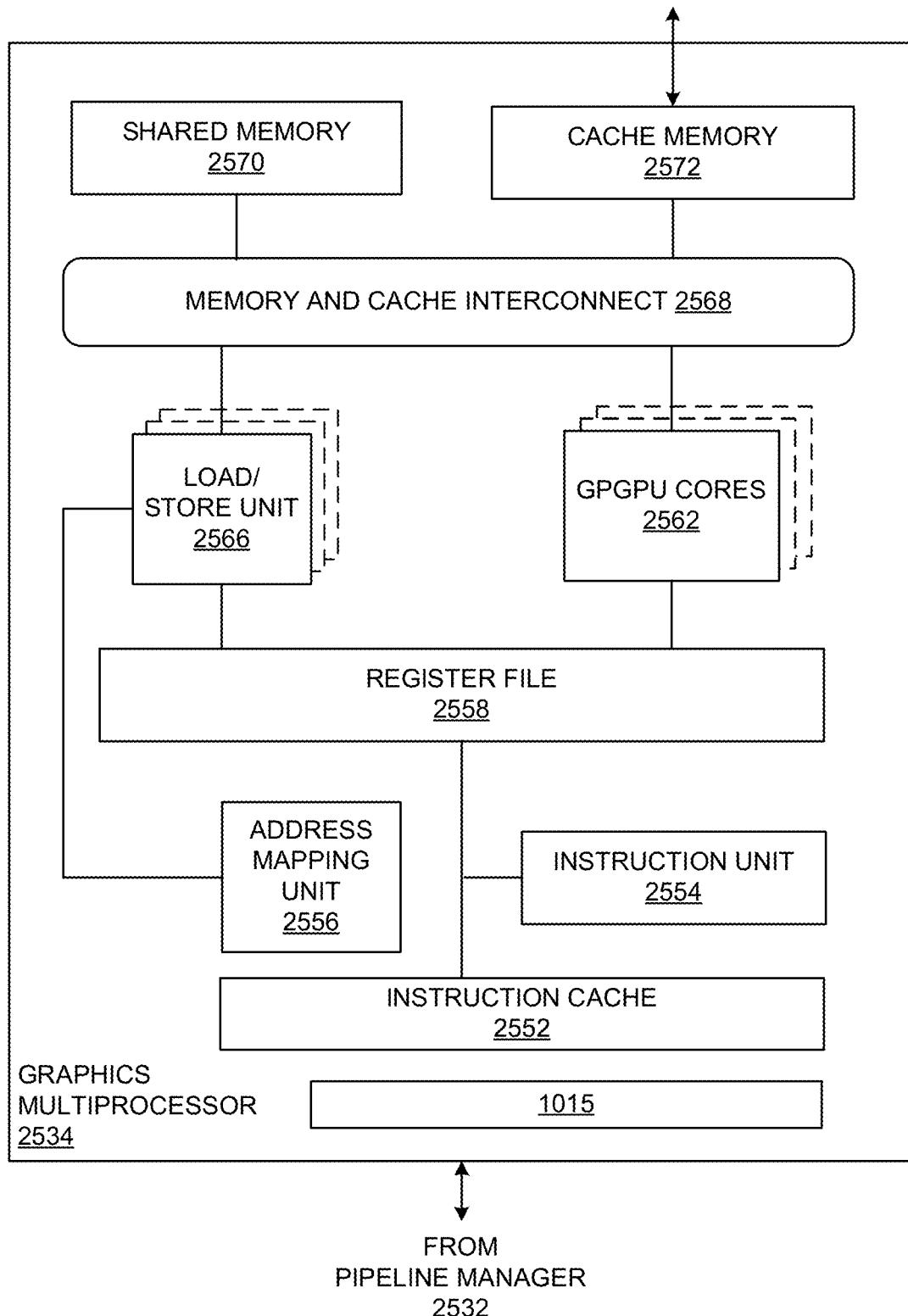


FIG. 25

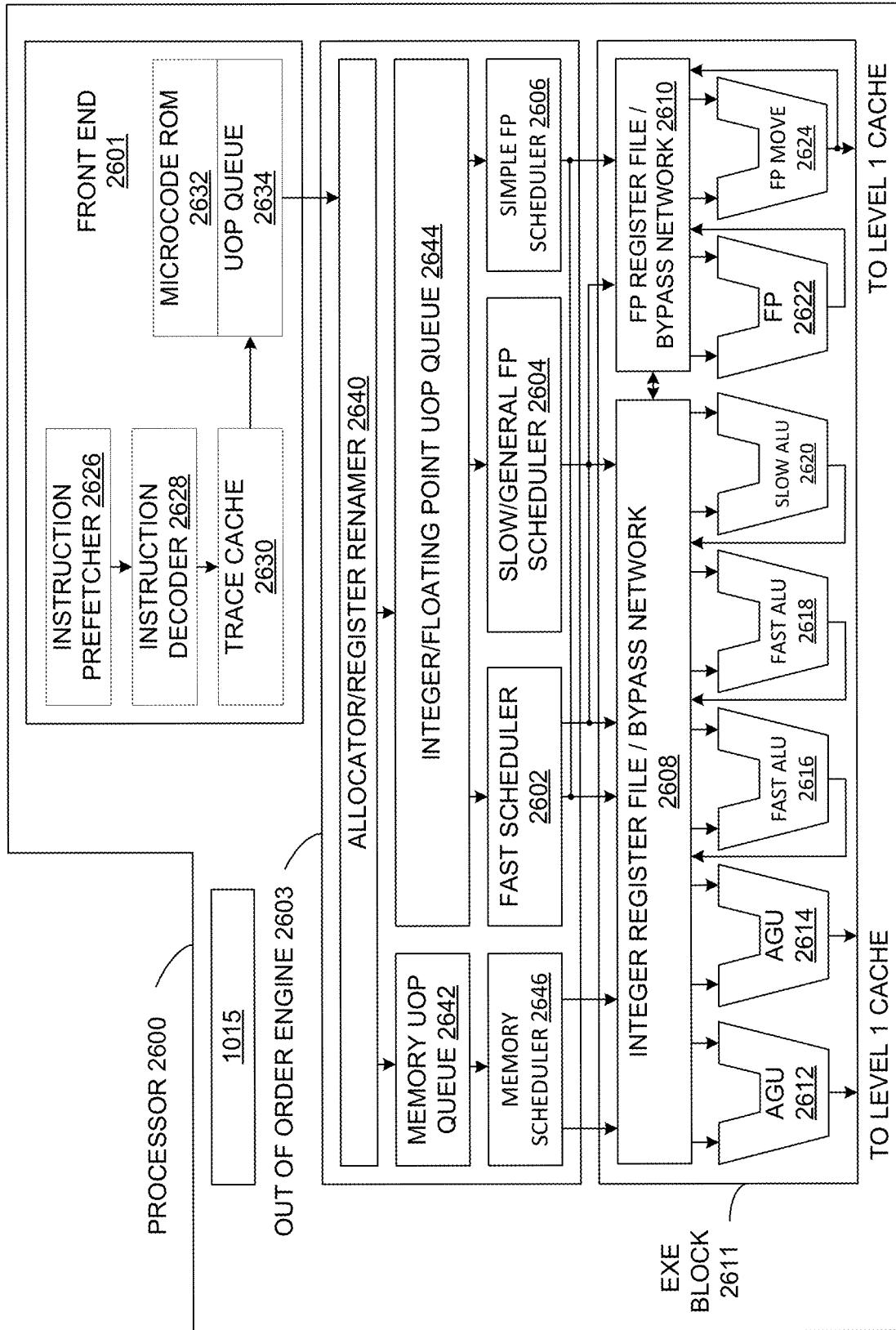


FIG. 26

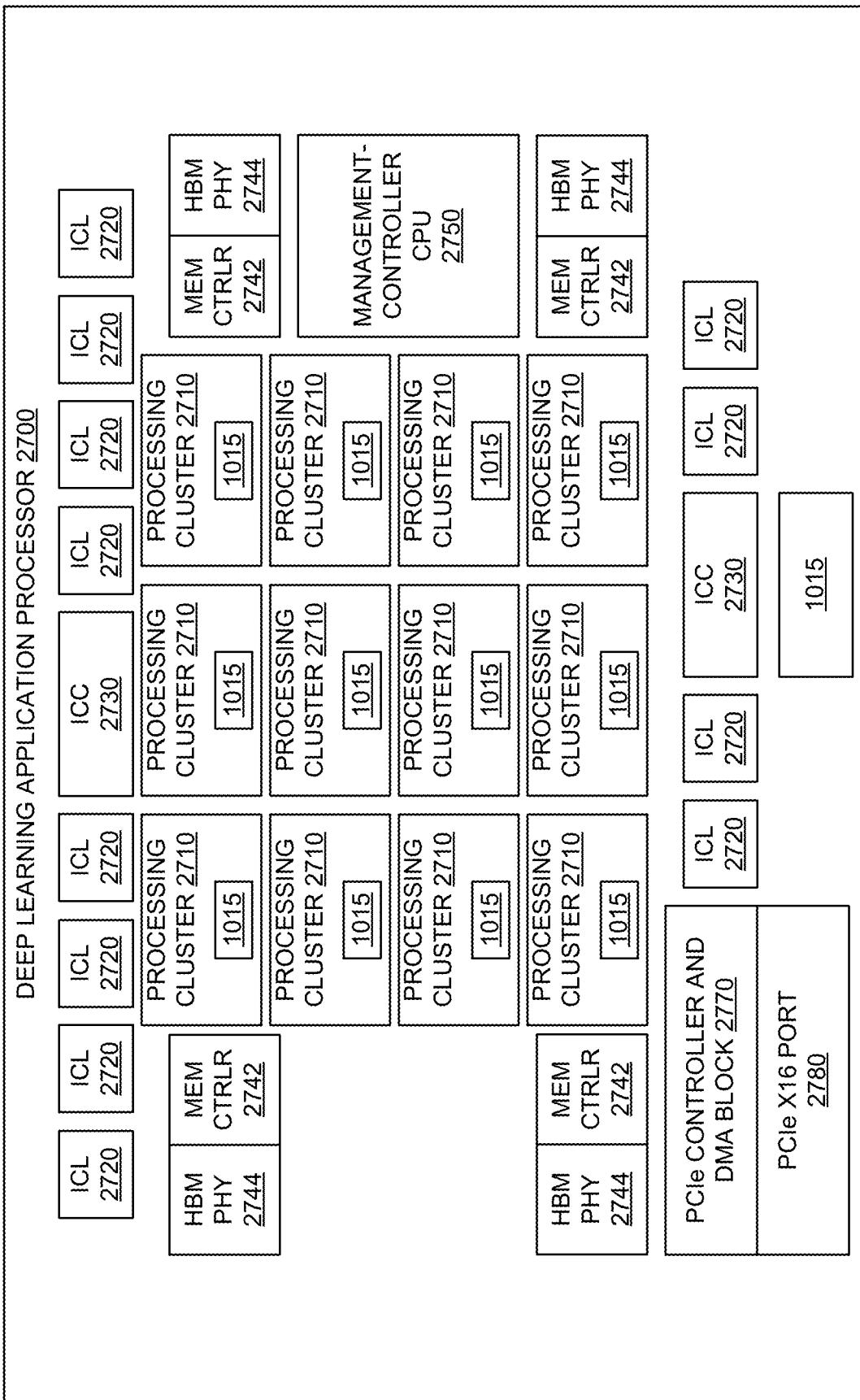
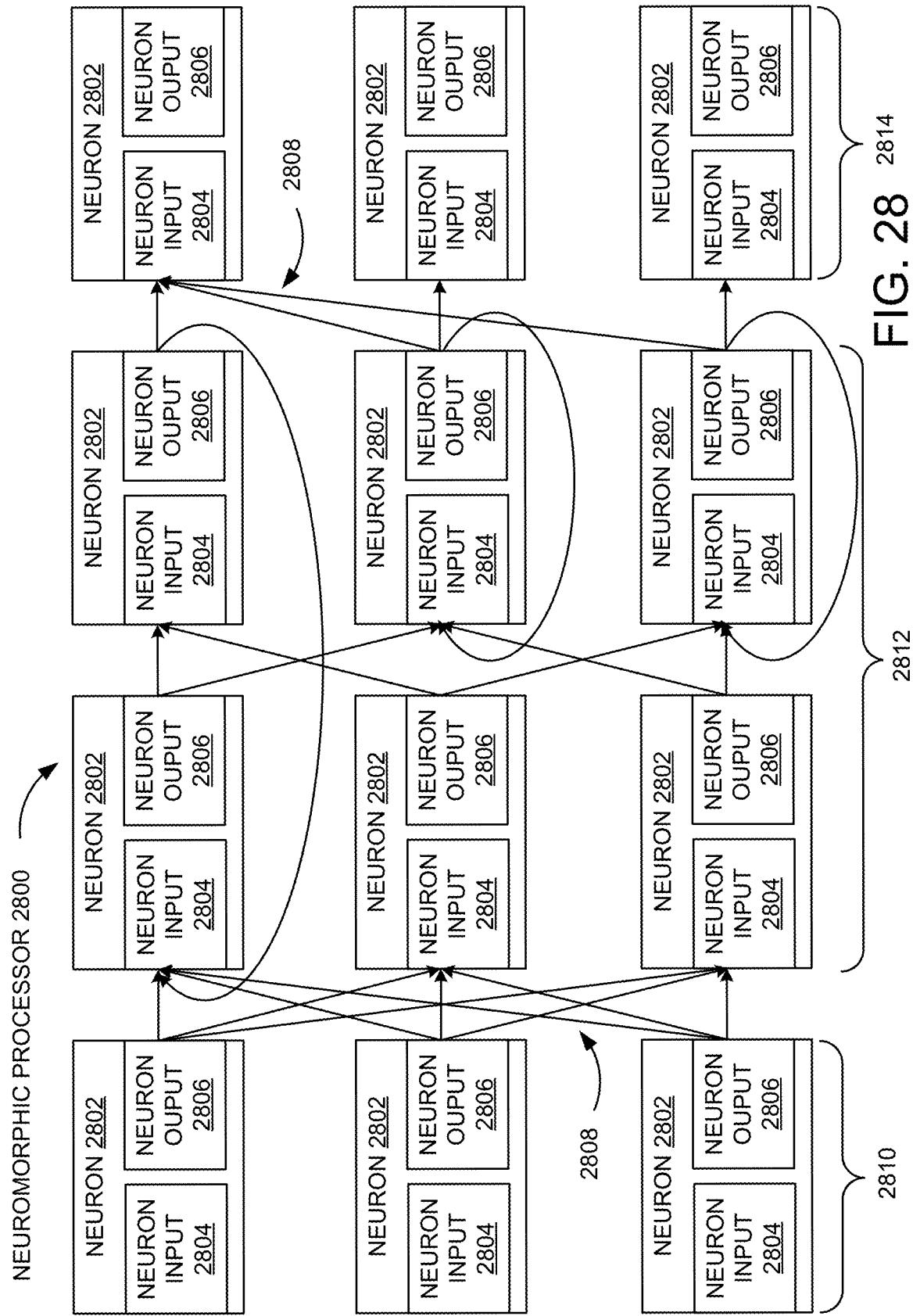


FIG. 27

**FIG. 28 2814****2812**

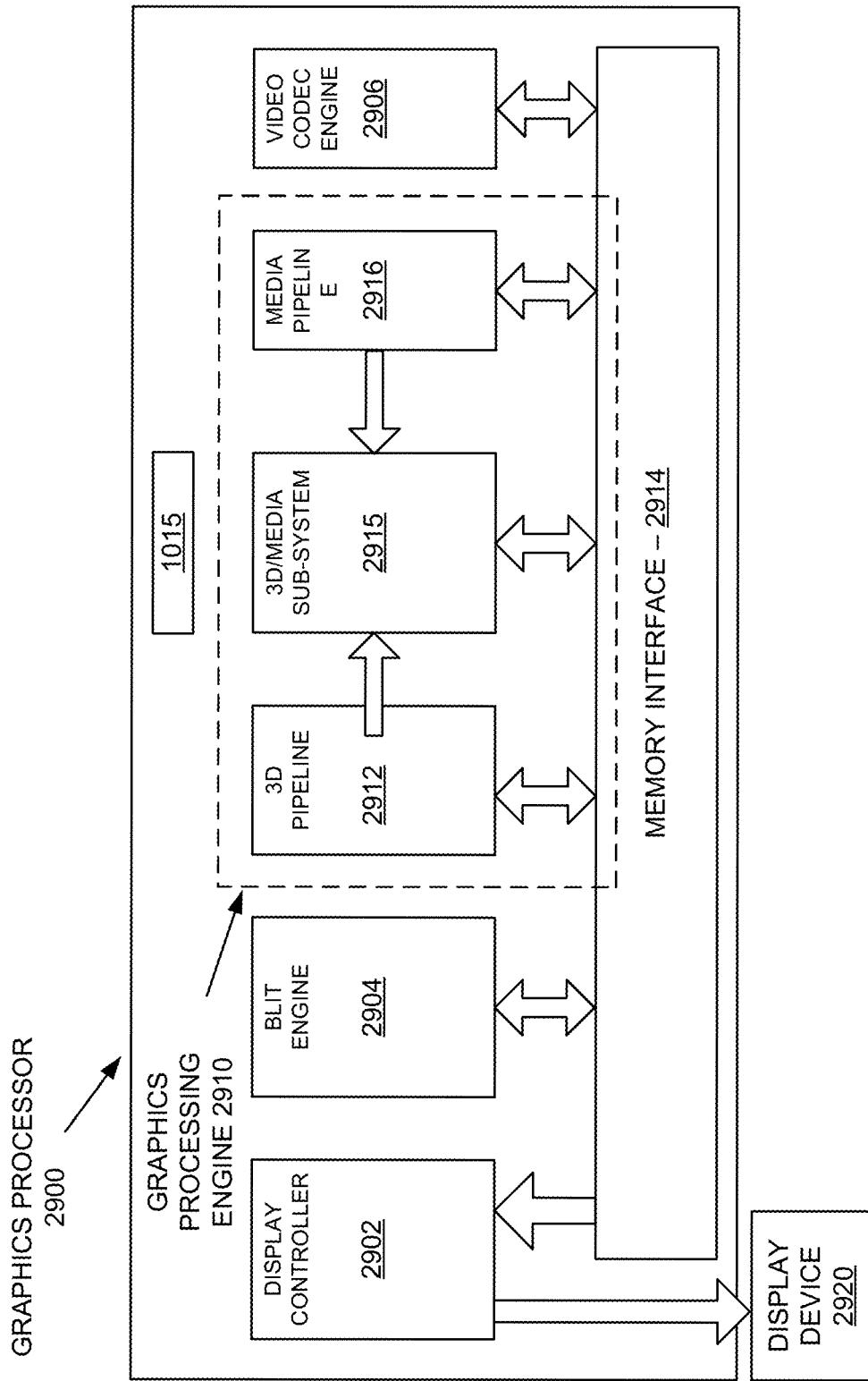
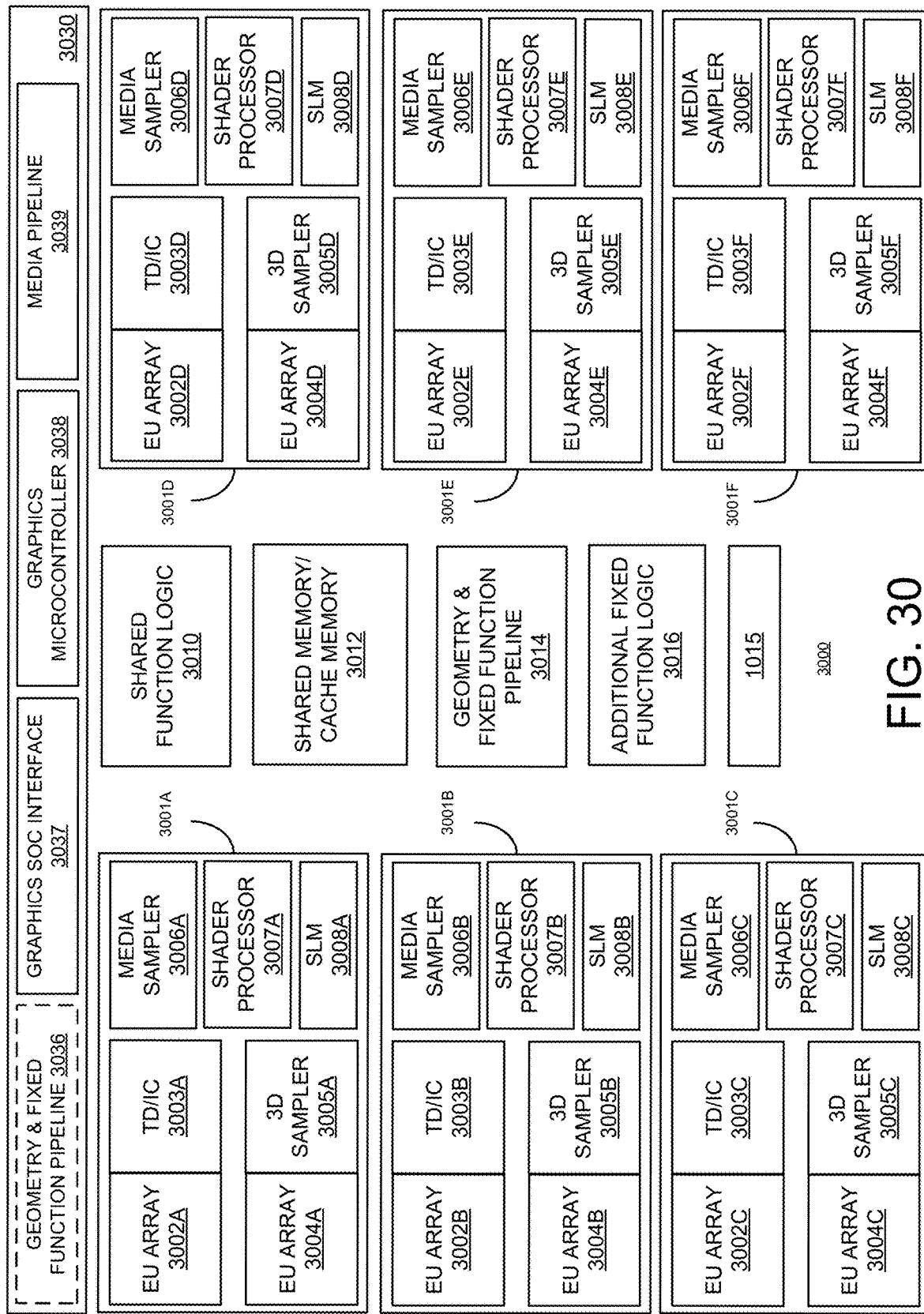


FIG. 29


FIG. 30

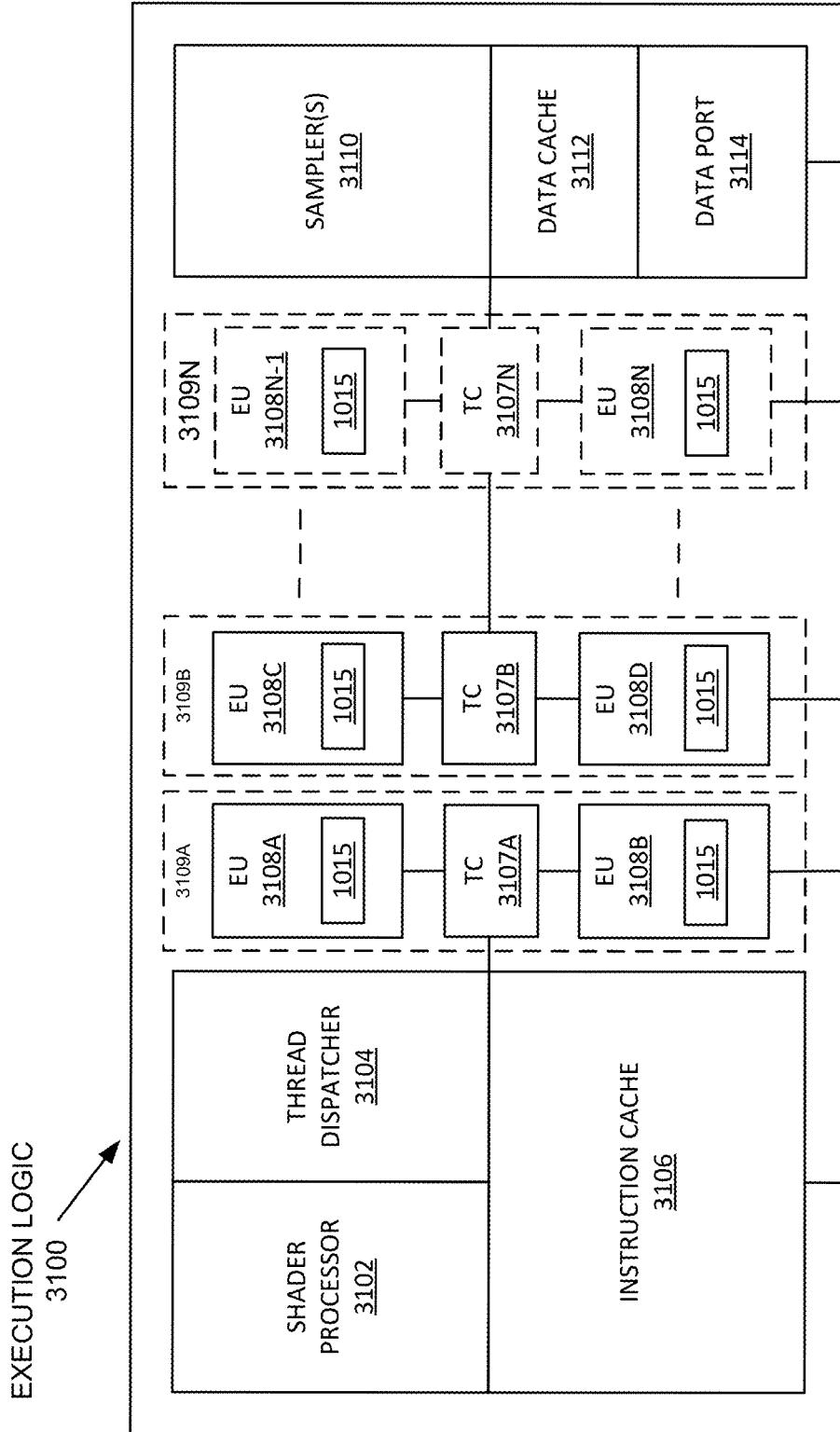


FIG. 31A

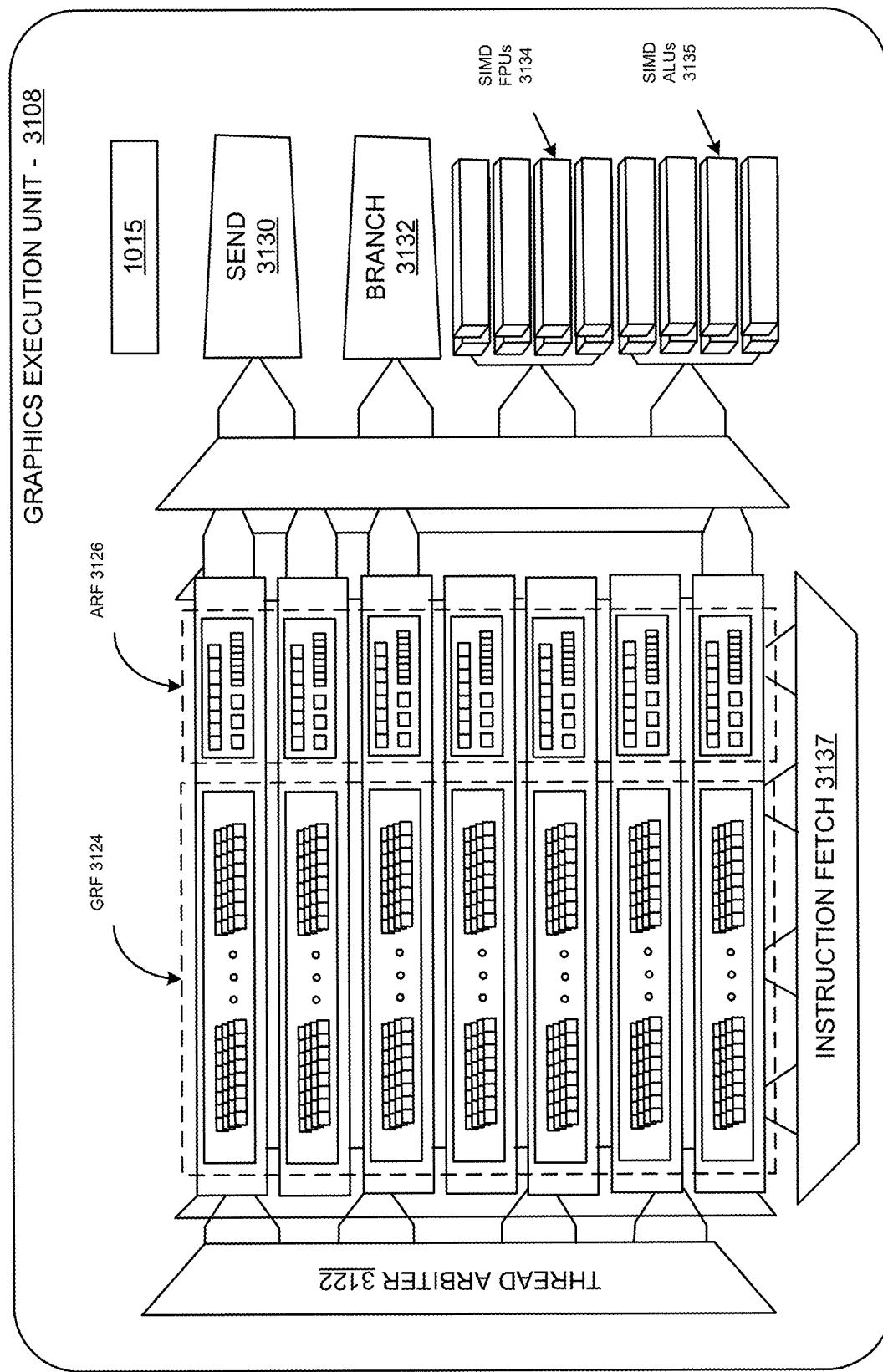


FIG. 31B

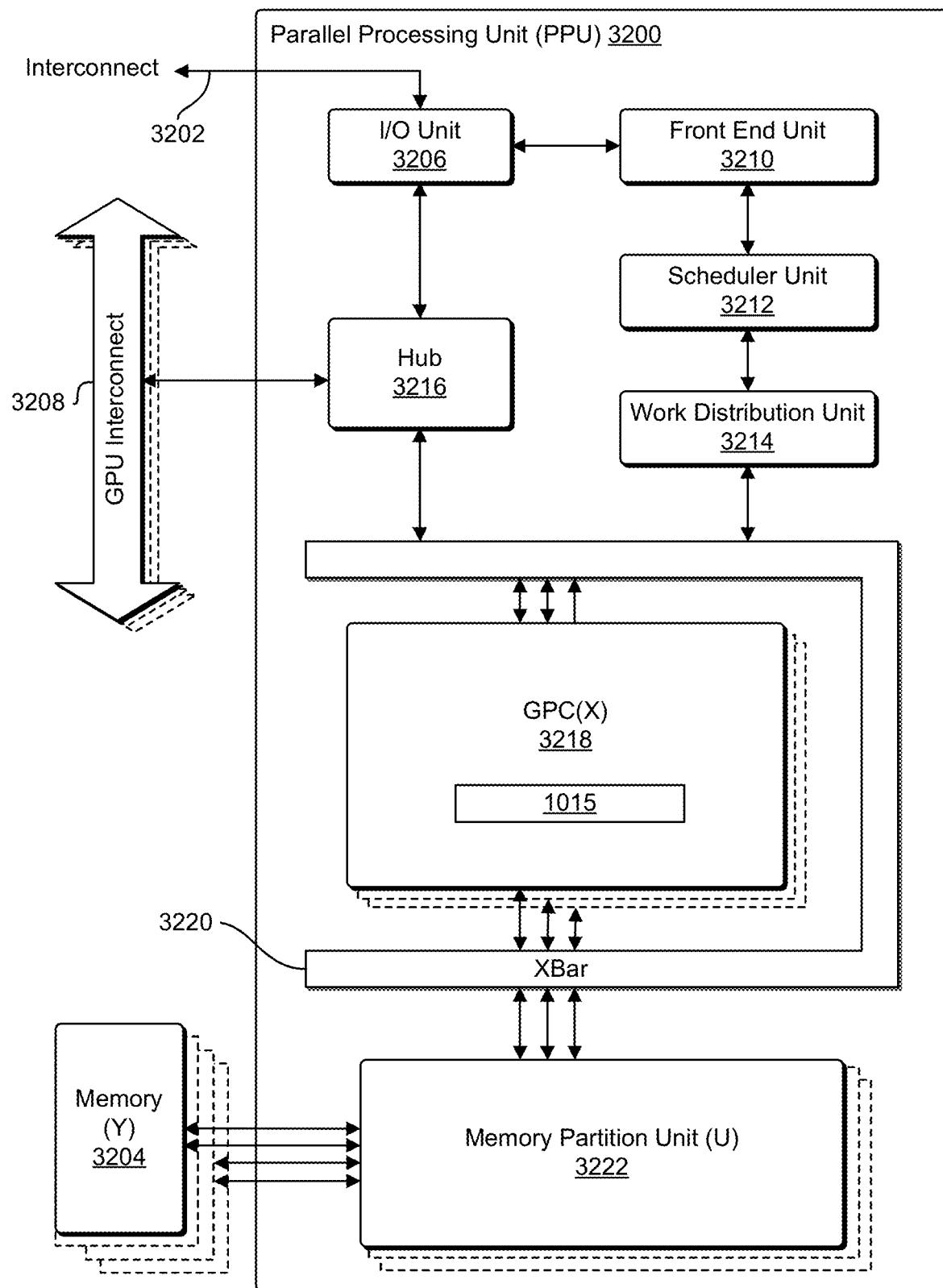


FIG. 32

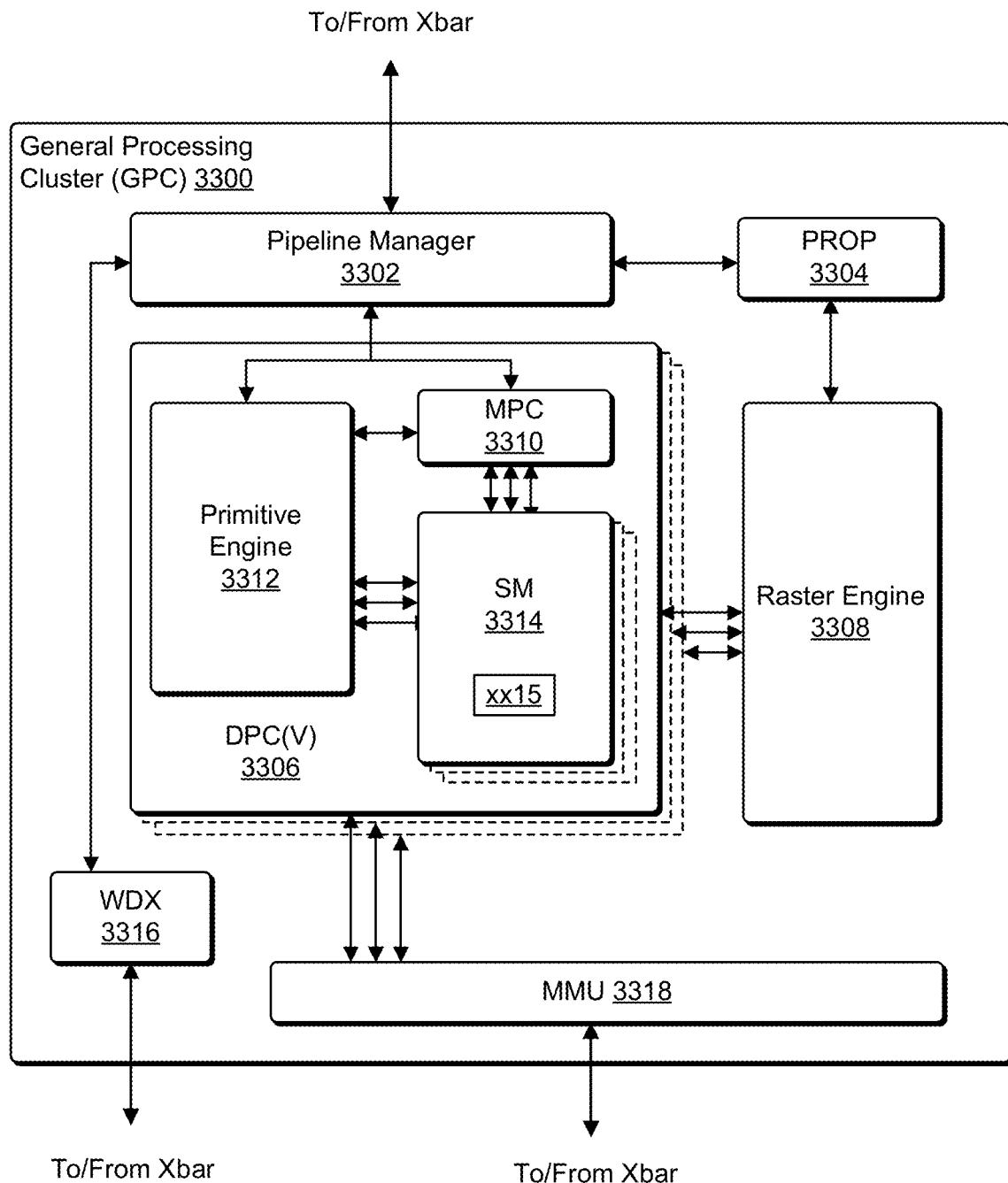


FIG. 33

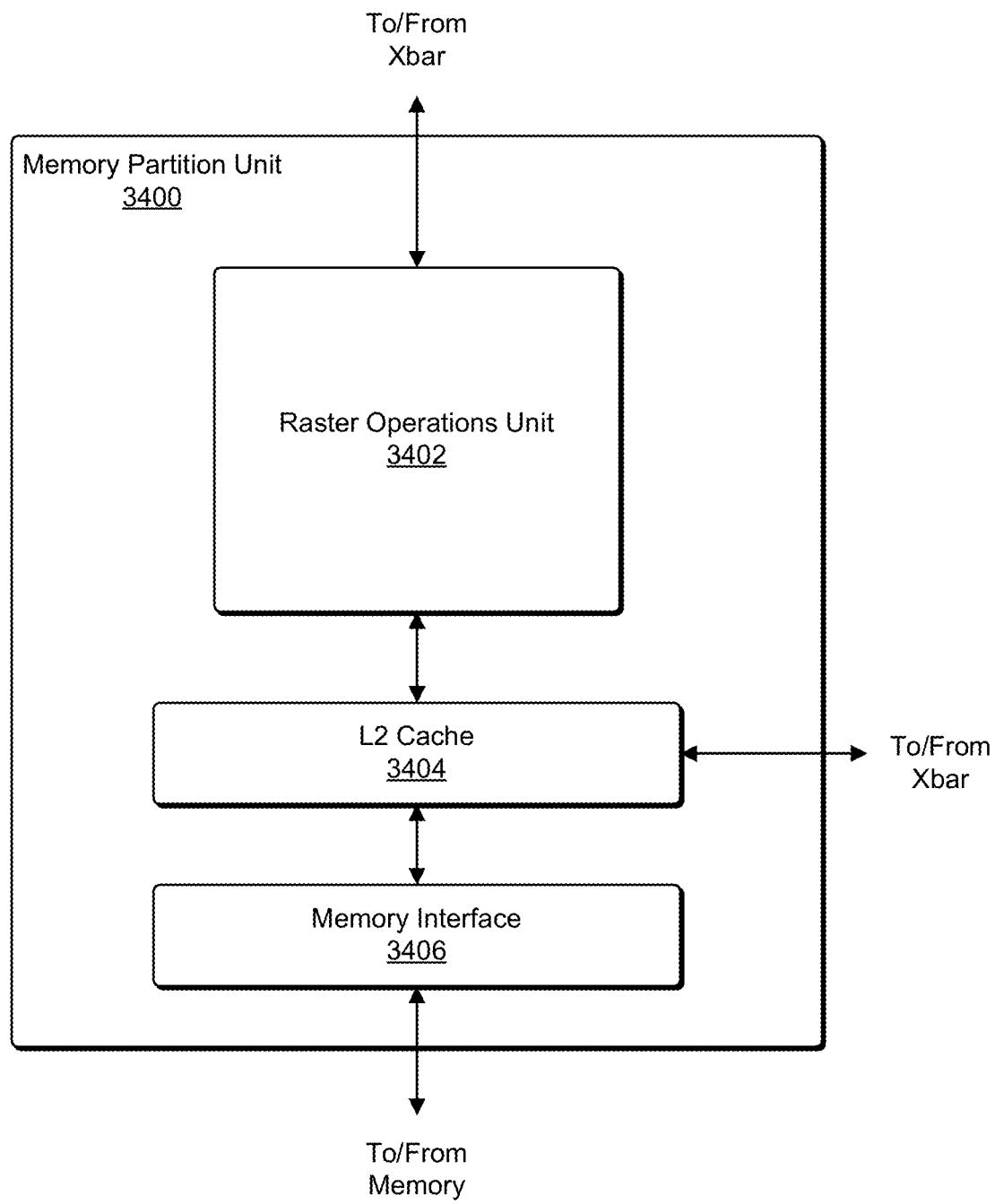


FIG. 34

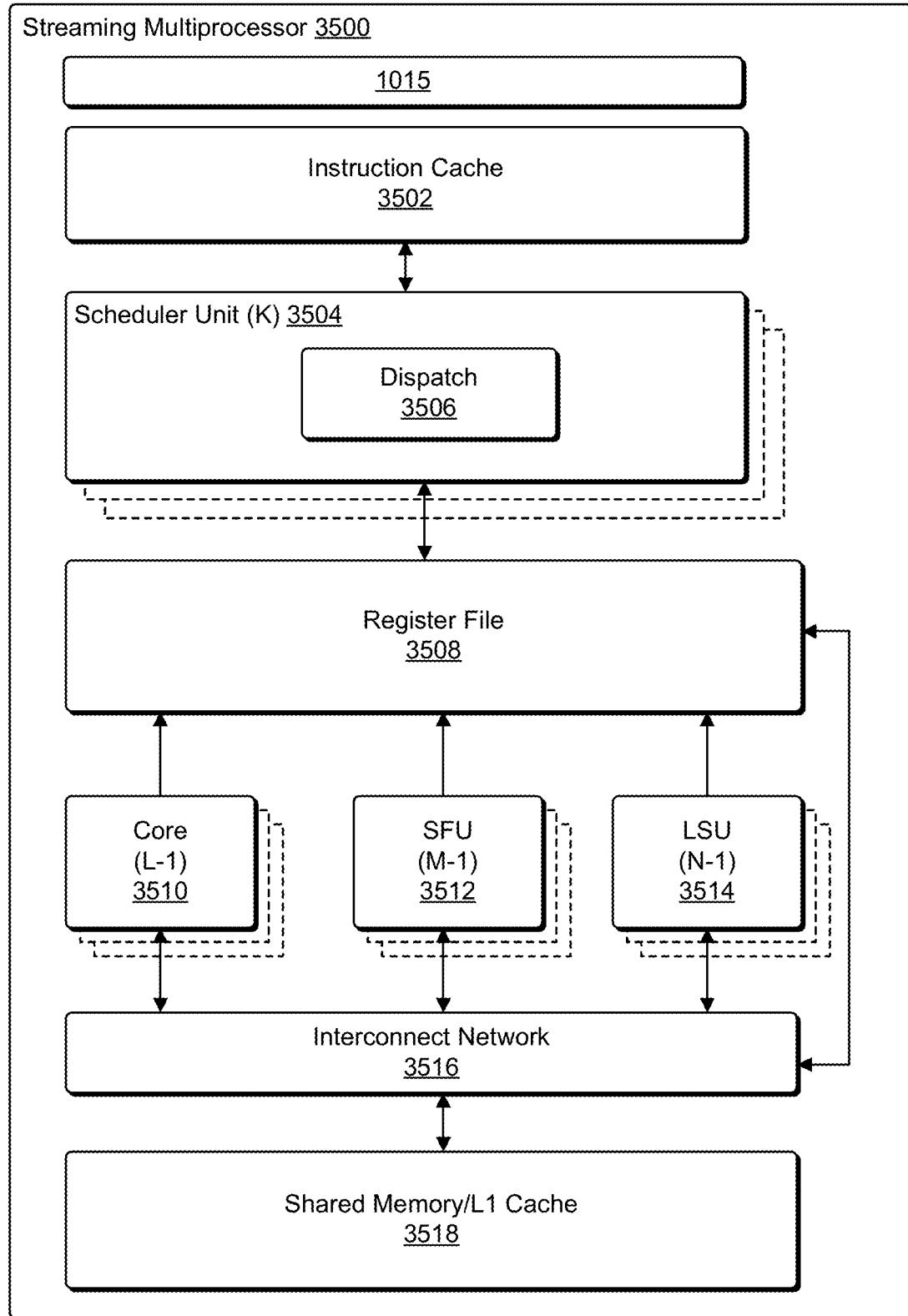


FIG. 35

**WEAKLY-SUPERVISED OBJECT
DETECTION USING ONE OR MORE
NEURAL NETWORKS**

BACKGROUND

[0001] Advances in computer technology have resulted in improved capabilities in object identification and analysis. Machine learning has been used as a tool for detecting objects in image data for purposes of such analysis. In order to train machine learning, a significant amount of labeled training data is needed. Creating this training data can be a long and complicated process.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] The present disclosure will be described with reference to the drawings, in which:

[0003] FIGS. 1A, 1B, and 1C illustrates example images that can be processed or generated using one or more neural networks, in at least one embodiment;

[0004] FIG. 2 illustrates components of an example multi-branch network, in at least one embodiment;

[0005] FIGS. 3A, 3B, 3C, and 3D illustrate components of an example network, in at least one embodiment;

[0006] FIGS. 4A and 4B illustrates example drop block approach, in at least one embodiment;

[0007] FIG. 5 illustrates example object detections that can be inferred, in at least one embodiment;

[0008] FIGS. 6A and 6B illustrate portions of an example process for training a neural network to infer object representations, and using such a network for inferencing, in at least one embodiment;

[0009] FIG. 7 illustrates an example environment in which aspects of at least one embodiment can be implemented;

[0010] FIG. 8 illustrates an example system for training an image synthesis network that can be utilized, in at least one embodiment;

[0011] FIG. 9 illustrates layers of an example statistical model that can be utilized, in at least one embodiment;

[0012] FIG. 10 illustrates inference and/or training logic, in at least one embodiment;

[0013] FIG. 11 illustrates inference and/or training logic, in at least one embodiment;

[0014] FIG. 13 illustrates an example data center system, in at least one embodiment;

[0015] FIG. 14 is a block diagram illustrating a computer system, in at least one embodiment;

[0016] FIG. 15 is a block diagram illustrating computer system, in at least one embodiment;

[0017] FIG. 16 illustrates a computer system, in at least one embodiment;

[0018] FIG. 17 illustrates a computer system, according at least one embodiment;

[0019] FIG. 18 illustrates exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, in at least one embodiment described herein;

[0020] FIGS. 19A-19B illustrate exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, in at least one embodiment;

[0021] FIGS. 20A-20B illustrate additional exemplary graphics processor logic in at least one embodiment;

[0022] FIG. 21 illustrates a computer system, in at least one embodiment;

[0023] FIG. 22 illustrates a parallel processor, in at least one embodiment;

[0024] FIG. 23 illustrates a partition unit, in at least one embodiment;

[0025] FIG. 24 illustrates a processing cluster, in at least one embodiment;

[0026] FIG. 25 illustrates a graphics multiprocessor, in at least one embodiment;

[0027] FIG. 26 is a block diagram illustrating a processor micro-architecture for a processor, in at least one embodiment;

[0028] FIG. 27 illustrates a deep learning application processor, in at least one embodiment;

[0029] FIG. 28 is a block diagram illustrating an example neuromorphic processor, in at least one embodiment;

[0030] FIG. 29 illustrates at least portions of a graphics processor, according to one or more embodiments;

[0031] FIG. 30 is a block diagram of at least portions of a graphics processor core, in at least one embodiment;

[0032] FIGS. 31A and 31B illustrates thread execution logic, in at least one embodiment;

[0033] FIG. 32 illustrates a parallel processing unit (“PPU”), in at least one embodiment;

[0034] FIG. 33 illustrates a general processing cluster (“GPC”), in at least one embodiment;

[0035] FIG. 34 illustrates a memory partition unit of a parallel processing unit (“PPU”), in at least one embodiment; and

[0036] FIG. 35 illustrates a streaming multi-processor, in at least one embodiment.

DETAILED DESCRIPTION

[0037] FIG. 1A illustrates an example image 100 that can be analyzed in at least one embodiment. This example image includes representations of multiple objects 102, including a man and a dog attempting to herd a group of sheep. In at least one embodiment, such an image used for training would have specific labels attached, which would include a type of object and location of an object in an image. In at least one embodiment location information could include coordinates for bounding boxes 142 indicating a portion of an image corresponding to an object as illustrated in an example image 140 of FIG. 1B. As mentioned, however, in at least one embodiment such annotations are not required for each training image, and can determine locations of various objects without specific labeling. At least one embodiment can determine proposals for objects of different classes, then use one or more refinements to arrive that locations of each object, such as illustrated in FIG. 1B. In at least one embodiment classification and location information can be used to detect and/or segment objects 182 represented in an image, as illustrated in an example image 180 of FIG. 1C. It should be understood that location information other than bounding box coordinates can be used as well in at least one embodiment.

[0038] In at least one embodiment a framework is utilized for instance-aware and context-focused weakly-supervised object detection that can accept image-level tags, without other spatial information. In at least one embodiment such a framework can differentiate between multiple instances of a same type of object rather than obtaining a single combined detection. In at least one embodiment an example frame-

work can also obtain accurate results for classes with large intra-class variance, and can be memory friendly. An example framework 200 in at least one embodiment is illustrated in FIG. 2, with a framework including an image branch 206 and at least one refinement branch 208, 210, 212. In at least one embodiment an image branch can accept as input an image-level label and an image, and can supervise a detection model that produces output and region of interest (ROI) scores. In at least one embodiment an ROI score can be an indication of confidence in at least one embodiment that an object is represented in that region. In at least one embodiment, one or more refinement branches can take at least some of these outputs, such as ROI scores, and attempt to refine or improve accuracy of a score, in order to make a more accurate object detection determination. In at least one embodiment, a refinement branch can help to solve instance ambiguity and part domination.

[0039] In at least one embodiment, detection of objects of interest can be performed using a multi-label image classification framework. In one embodiment, object categories of interest can be collected in a set C. Given an input image I and a set of pre-computed region proposals R, in at least one embodiment convolutional layers from a set of pre-trained DNNs ('Base' 202 in FIG. 2) can be used to extract features, which can then be fed into an ROI pooling component to produce region features for every region in a set R. In at least one embodiment these features can be passed through a set of intermediate layers ('Neck' 204 in FIG. 2), then passed through two different fully-connected layers, illustrated via boxes FC_{cls}, FC_{det} in FIG. 2, to produce classification logits f_w(c, r) ∈ ℝ and detection logits g_w(c, r) ∈ ℝ for every object category c ∈ C and for every region r ∈ R. Vector w in this example subserves all trainable parameters, including Base, Neck, and Head. In at least one embodiment a score s(c|r) is obtained for a region r being classified as category c when using a softmax on classification logits, as may be given by:

$$s_w(c | r) = \frac{e^{f_w(c,r)}}{\sum_{c \in C} e^{f_w(c,r)}}$$

[0040] Similarly, in at least one embodiment a score s(c|r) can be obtained for detecting category c with region r when using a softmax on detection logits:

$$s_w(r | c) = \frac{e^{g_w(c,r)}}{\sum_{r \in R} e^{g_w(c,r)}}$$

[0041] In at least one embodiment a set of region of interest (ROI) scores s_w(c, r) for assigning category c ∈ C to region r ∈ R can be computed via an element-wise product, such as may correspond to s_w(c, r) = s_w(c|r)s_w(r|c) ∈ [0, 1]. For inferencing, in at least one embodiment these ROI scores can be used to independently rank all regions for each category. In at least one embodiment non-maximum suppression and thresholding can be applied to remove redundant boxes in at least one embodiment.

[0042] Since only image-level labels are available, in at least one embodiment a loss function can be constructed by summing ROI scores s_w(c, r) for all regions r ∈ R to obtain image evidence:

$$\phi_w(c) = \sum_{r \in R} s_w(c, r)$$

for class c in an image. In at least one embodiment a ground-truth class label y(c) ∈ {0, 1} can be used to indicate image-level existence of category c in a given image, and multi-label binary cross entropy loss can be used, as may be given by:

$$\mathcal{L}_{img}(w) = - \sum_{c \in C} y(c) \log(\phi_w(c))$$

to train parameters w. In at least one embodiment this computation corresponds to an image branch 206 illustrated in FIG. 2.

[0043] Instead of computing detections from these ROI scores s_w(c, r), in at least one embodiment scores can be used to generate instance-level self-supervision for a refinement branch. In at least one embodiment an online refinement module can be used to encourage highly-overlapping proposals to be assigned a same classification. For example, in at least one embodiment an ROI score s_w(c, r) can be used to generate a pseudo-label ŷ¹(c, r) ∈ {0, 1}, where every region r can only be assigned to one class.

[0044] With pseudo-label ŷ¹(c, r) as self-supervision, in at least one embodiment a refinement branch can address instance-level classification in at least one embodiment by:

$$\mathcal{L}_{roi}^1(w) = - \frac{1}{R} \sum_{r \in R} \sum_{c \in C} \lambda_r \hat{s}_w^1(c, r) \log \hat{s}_w^1(c, r)$$

where λ_r are hyper-parameters and ŷ¹(c, r) is output of a first refinement branch, in this example an output after FC_{ref} in FIG. 2 passed through a class-wise softmax. In at least one embodiment hyper-parameters λ_r can be used to adaptively balance ℝ_{img} and ℝ_{roi}¹ so that more attention can be put on ℝ_{roi}¹ when pseudo-label ŷ¹ gradually improves during training. In at least one embodiment, multiple refinement branches ℝ_{roi}^k(w) can be stacked and current ROI scores ŷ^k(c, r) used to compute pseudo-labels ŷ^{k+1} as illustrated in FIG. 2. In at least one embodiment a final score can be obtained after averaging all ŷ^k, not including ŷ^k.

[0045] In at least one embodiment, such a framework can yield compelling results for instance-level object detection. It is possible, however, to obtain more accurate instance-level pseudo-labels, such as by providing adequate focus on context. In at least one embodiment, to obtain pseudo-labels ŷ^{k+1}(c, r), top scoring proposals can be a point of focus. In at least one embodiment for every class c, a top ranking can be located, such as by setting:

$$r^{*,k}(c) = \operatorname{argmax}_{r \in R} \hat{s}_w^k(c, r)$$

$$\hat{y}^{(k+1)}(c, r) = 1$$

if an intersection over union ("IoU") between region r and region r^{*k}(c) is larger than 0.5. It should be noted that in this example ŷ⁰(c|r) = s_w(c, r).

[0046] To find pseudo-labels $\hat{y}^{k+1}(c, r)$ that are more suitable for instance-level WSOD, in at least one embodiment a spatial relationship of multiple instances in an image can be considered. For example, instead of only operating with a top-1 ROI $r^{*,k}(c)$, in at least one embodiment all proposals r can be sorted according to their ROI score $\hat{s}^k(c|r)$ for each class c . In at least one embodiment a top p percent of ranked ROIs for every category c can be selected, and $R'(c)$ can be used to refer to a resulting set. Subsequently, in at least one embodiment a greedy algorithm can be utilized to select instances from $R'(c)$, driven at least in part by a motivation that two non-overlapping and confident proposals have a high probability of corresponding to different objects. In at least one embodiment, a top proposal can be added to a pseudo label pool. Then, in at least one embodiment starting from a second highest to a last one, an ROI can be added if it does not overlap with another one in a pool. In at least one embodiment, since two instances can be relatively close together, a very small threshold τ can be used to reject ROIs, rather than strictly enforcing a zero-overlap rule. Similarly, in at least one embodiment proposals with an IoU bigger than 0.5 with any pseudo-label in a pool can be treated as positive examples during training.

[0047] In at least one embodiment memory-efficient sequential batch forward and backward computation can be used, which can be tailored for WSOD models. In at least one embodiment, basic training via backpropagation can store all intermediate activations during a forward pass, which are reused when computing gradients of network parameters. In at least one embodiment such a method can be computationally efficient due in part to memorization, yet memory-demanding for a same reason. In at least one embodiment only a subset of intermediate activations are saved during a forward pass at key layers. In at least one embodiment, an entire model can be divided into smaller sub-networks at these key layers. In at least one embodiment when computing gradients for a sub-network, a forward pass can be applied to obtain intermediate representations for this subnetwork, starting from stored activations at an input key layer of a sub-network. Combined with gradients propagated from earlier sub-networks, in at least one embodiment gradients of sub-network weights can be computed and gradients propagated to outputs of earlier sub-networks.

[0048] In at least one embodiment this can be particularly beneficial for extremely deep networks in at least one embodiment, where a memory cost may be roughly evenly distributed along layers. In at least one embodiment, when these DNNs are adapted for detection, however, activations (after ROI-Pooling) can grow from $1 \times \text{CHW}$ (image feature) to $N \times \text{CHW}$ (ROI-features), where N is in thousands for weakly-supervised models. In at least one embodiment, without ground-truth boxes, these proposals can be maintained to keep good recall and thus good performance.

[0049] In at least one embodiment this training challenge can be addressed at least in part by using a sequential batch back-propagation. In at least one embodiment this can be performed in a Head sub-module as illustrated in an example 300 of FIG. 3A, in a Neck sub-module, as illustrated in examples 320 and 340 of FIGS. 3B and 3C, and in a Base sub-module in example 360 of FIG. 3D. In these figures illustrating sequential batch back-propagation in one iteration, an activation, gradients, and a module are shown that are being updated in that step. In at least one embodiment, during a forward pass, an input image is first passed through

a Base and Neck, with only activations A_b after Base being stored. In at least one embodiment, output of a Neck then goes into a Head for its first forward and backward pass to update weights of a Head and gradients G_h as illustrated. In at least one embodiment, to update parameters of a Neck, ROI-features can be split into batches and backpropagation run sequentially on small batches. Such an approach can avoid storing memory-consuming feature maps and their gradients within a Neck. In at least one embodiment an example of such a sequential method is shown in FIG. 3C, where 2,000 proposals are split into two batches of 1000 proposals each. In at least one embodiment a gradient G_b is accumulated and used to update parameters of a Base network via regular back-propagation as illustrated in FIG. 4D. For testing, in at least one embodiment a similar strategy can be applied if either a number of ROIs or a size of a Neck is too large.

[0050] In at least one embodiment, due at least in part to intra-category variation, existing WSOD methods often mistakenly only predict discriminative parts of an object rather than its full extent. In at least one embodiment a DNN can be encouraged to focus on a context that can be achieved by dropping some most discriminative parts, referred to herein as performing spatial dropout. Since discriminative parts of objects differ in location and size, in at least one embodiment object detection can utilize a structured drop block approach for each region of interest, such as is illustrated in an example of FIGS. 4A and 4B. In at least one embodiment, during training of points on ROI-feature maps can be sampled, such as with a Bernoulli distribution of probability y as blob centers. In at least one embodiment object regions can be divided into arrays of cells 402, 402 as illustrated in an example of FIG. 4A. In at least one embodiment square or rectangular regions 452, 454 around these centers of size $w \times w$ can be zeroed out as illustrated in an example 450 of FIG. 4B, such as by having relevant pixel values set to zero or another set value, across all channels on a feature map. In at least one embodiment feature values can then be re-scaled by a factor of an area of ROI over an area of an un-dropped region, for example, such that no normalization has to be applied for inference where all features are kept. In at least one embodiment, discriminative parts such as faces, car fronts, and heads are zeroed out.

[0051] In at least one embodiment a model for object detection and/or segmentation can be trained using data that does not contain very detailed labels, and in some instances might only indicate classes of objects represented in an image or video frame, without any location or occurrence information being provided as well. In at least one embodiment, labels might only state that there is at least one sheep and at least one dog represented in an image, but without indication of how many sheep or dogs are represented, or where those representations might be located in an image. In at least one embodiment these types of labels are referred to herein as image-level labels. In at least one embodiment a fully supervised method would need bounding box coordinates, or other location information, to determine a size of an object boundary and location of an object representation in an image, in order to accurately train a model.

[0052] In at least one embodiment, there can be different numbers and/or selections of refinement branches and functionality. In at least one embodiment an image detection branch can receive and/or provide proposals for a plurality of bounding boxes that each corresponds to a potential

object representation, or at least a representation of a type of object or object satisfying specific criteria, etc. In at least one embodiment a separate object region proposal network or algorithm will generate an initial set of proposals. In at least one embodiment an image detection branch outputs a score, or set of scores, for object proposals, along with one or more classifications, and in at least one embodiment can reduce a number of proposals based at least in part upon scores and/or redundant proposals. In at least one embodiment output of an image branch for a given table is a matrix of dimension n (for a number of proposals) by c (for a number of identified classes). In at least one embodiment each element of a matrix is then a confidence score (or other such score) indicating a confidence that a given object proposal is of a specific class. In at least one embodiment, these results can include FC_{cls} and FC_{det} values that correspond to post-processing results from this table. FC_{cls} and FC_{det} can each be each a different nxc matrix derived from an original nxc matrix for initial image proposals. In at least one embodiment each matrix can be normalized, such as by utilizing a softmax function along both a proposal dimension and a classification dimension. In at least one embodiment normalization can provide for easier comparisons. In at least one embodiment a higher element score can be indicative of a higher likelihood of a correctly identified object and classification. In at least one embodiment tables can then be multiplied back as illustrated in FIG. 2. In at least one embodiment a result is a new nxc matrix that is output of a network. In at least one embodiment a global, adaptive, and/or other such threshold can be applied, and any score falling below that threshold can be determined to not correspond to a correct actual object representation. In at least one embodiment elements with scores at or above a threshold in at least one embodiment can be determined to be correct object representations in at least at least one embodiment.

[0053] Since image-level labels are available, in at least one embodiment a sum can be performed across various instances along n dimension of a matrix. In at least one embodiment a result is a c-dimensional pool vector, which is supervised by image-level labels. In at least one embodiment such an approach provides a multi-label loss. In at least one embodiment a loss can supervise a vector, which can back-propagate a gradient to be used to train a network. In at least one embodiment a sum pooling process can help to reduce a large, initial table of proposals to something more like an image-level representation, which can then be more easily refined. In at least one embodiment a refinement branch follows a teacher/student model. In at least one embodiment it takes labels from initial predictions as a set of proposals with corresponding detection scores. In at least one embodiment, for each object class from image-level labels, proposals can be analyzed and their scores sorted. In at least one embodiment all scores can be normalized to a value between 0 and 1. In at least one embodiment a global threshold can be applied whereby any proposals with values that meet or exceed thresholds have their value set to 1, and any proposals with values less than a threshold can have their value set to 0. In at least one embodiment this process results in a quantization of all scores. In at least one embodiment input is a score from a previous branch, but because of this quantization a set of pseudo-labels is produced for each refinement branch. An example refinement branch can produce an nxc matrix with all element values

being either 0 or 1. In at least one embodiment pseudo-labels can be used to define a loss, which can help to refine this branch as well.

[0054] In at least one embodiment initial seeds, or positions of various object proposals, are prevented from becoming very close to each other. In at least one embodiment, a sorting of top windows could be performed, where a “top window” can correspond to a bounding box for a seed or proposal that has a high confidence score. In at least one embodiment many of these top windows will likely be very close together in an image, as multiple windows may correspond to a different view or approximation of an object represented in an image. In at least one embodiment top windows that are determined to be very close to another top window, or similar in size and location, can be excluded from consideration. In at least one embodiment, a greedy method with maximum suppression can be used to exclude redundant windows or proposals from consideration, as there is a high probability that they correspond to a same object representation. In at least one embodiment a quantization of a nxc matrix can again be performed, where an initial (highest value similar) window is set to a value of 1, and redundant or similar windows are set to a value of 0. In at least one embodiment a large set of diverse seeds can then be selected or identified that can be used as starting points. In at least one embodiment elements of a matrix, or pseudo-labels, for various proposals can be adjusted so that any window that significantly overlaps another window will drop out of consideration. In at least one embodiment a process is a form of label propagation, which can help to prevent a part domination problem.

[0055] In at least one embodiment a refinement approach is referred to herein as a drop-block refinement. In at least one embodiment such a refinement approach can remove one or more sub-regions or portions of an object proposal region from consideration. FIGS. 4A and 4B illustrate an example drop-block approach that can be utilized in accordance with at least one embodiment. In at least one embodiment, a set of “blocks” (where blocks represent pixels or groups of pixels in an image) are dropped, or removed from consideration for an image proposal region. An example situation 400 in FIG. 4A illustrates a segmentation of an object proposal region into blocks, and FIG. 4B illustrates a situation 450 wherein a subset of these blocks has been removed from consideration in at least one embodiment. In at least one embodiment, blocks, and sizes of blocks, can be selected at random or according to a block selection algorithm. In at least one embodiment a remaining portion of proposal data can be analyzed to determine whether a similar confidence score is obtained, indicating that a region likely represents an object of that class. In at least one embodiment this can also help train a network to infer a correct classification for an object for various different views or portions of an object. In at least one embodiment drops can be selected to drop from a region such as a face that would otherwise provide a high confidence region for a type of object. In at least one embodiment a process of randomly dropping blocks can also effectively introduce some noise into an image region, which can help to improve an ability of a network, once trained, to recognize different object representations.

[0056] In at least one embodiment further advantages can be obtained that can be at least partially independent of selected refinement approaches. In at least one embodiment

an issue of memory consumption can be addressed by utilizing a batch back-propagation technique. In at least one embodiment data can be segmented into smaller batches that can be processed separately and/or in parallel, such that less data needs to be stored in memory for any given back-propagation task. In at least one embodiment processing of various batches can also be staggered such that all data does not need to be processed and/or stored in processor memory concurrently. In at least one embodiment this reduction in memory consumption can also enable object detection and/or segmentation to be performed on video or other large data sets, streams, or sources.

[0057] In at least one embodiment refinement branches all operate in parallel, taking as input proposal data from a detection branch. In at least one embodiment results from various refinement branches can then be aggregated with results from an object detection branch, such as by averaging or using a selected aggregation function that calculates a weighted or other function of various values for proposals. In at least one embodiment a proposal that does not satisfy requirements of various refinements can have its score substantially reduced, which may cause that proposal value to fall below a threshold and therefore be considered to not correspond to an object of a relevant class.

[0058] In at least one embodiment, refinement branches can each take output (e.g., ROI scores) from an earlier branch in a specified order. In at least one embodiment such an approach can save an extra processing step, as an ROI score from an image branch will be refined as it passes through various refinement branches. In at least one embodiment some accurate object representations might have a corresponding ROI score improperly lowered by a refinement branch to an extent where a proposal would be removed from consideration. In at least one embodiment performing refinements separately allows an object proposal to remain under consideration if, for example, all but one refinement branch indicate a high confidence in object determination for a given class.

[0059] In at least one embodiment, such a process is adaptive in that it can be performed for multiple instances of a type of object, as well as multiple types of objects, represented in a single image or video frame. In at least one embodiment pseudo-labels that are generated serve as a type of ground truth data, providing at least some constraint to learning. In at least one embodiment, a number of pseudo-labels with scores above a threshold value are determined, which are used to refine scores for object proposals from an initial object detection branch or process. In at least one embodiment, initial proposal scores are averaged with refinement scores to generate a final score, which can then be compared against a threshold or otherwise processed to determine which object proposals are accepted as being accurate.

[0060] FIG. 5 illustrates an example set of images 500 including object detection regions that can be inferred in accordance with at least one embodiment. In at least one embodiment, for each pair of images, results of a different approach are illustrated on left, and results of an approach in accordance with at least one embodiment are illustrated on right. In at least one embodiment each column of pairs also illustrates that an approach in accordance with at least one embodiment improves results for specific problems. For example, in at least one embodiment a left column illustrates that an approach does not miss instances of a type of object

in an image. In at least one embodiment a middle column illustrates that an approach in accordance with at least one embodiment does not have problems detecting objects when an object proposal regions are initially grouped in an image. Further, in at least one embodiment a right column illustrates that an approach in accordance with at least one embodiment does not determine object regions in an image based upon part dominations, or regions of a face or grill that dominate a detection and may miss other parts of an object.

[0061] FIG. 6A illustrates an example process 600 for training a neural network, such as a generative adversarial network (GAN), to infer object detections and/or segmentations in at least one embodiment. It should be understood for this and other processes discussed herein that there can be additional, alternative, or fewer steps performed in similar or alternative orders, or in parallel, in at least one embodiment unless otherwise stated. Further, this example discusses training a generative adversarial network (GAN) using alphanumeric semantic description data, but as discussed elsewhere herein there can be various types of models trained using a variety of different types of data within a scope of at least one embodiment. In at least one embodiment, an image (or video frame, etc.) is obtained 602 that includes at least one representation of an object type of interest, with an image including one or more image-level labels indicating one or more classes of objects represented in an image. In at least one embodiment, labeling may not include any location or instance data, etc. In at least one embodiment a set of object proposals can be determined, either external or internal to a neural network, such as a GAN in this example. In at least one embodiment object proposal regions can be input to a detection branch of a GAN, which as discussed elsewhere herein can determine a region of interest (ROI) score or similar such value for various proposals, and in at least one embodiment can reduce a number of proposals based at least in part upon determined scores. In at least one embodiment ROI scores for various proposals can be passed to one or more refinement branches of a network that can determine 606 pseudo-labels for various proposals. In at least one embodiment these pseudo-labels can be associated with new or updated ROI scores for various proposals as determined using criteria or approaches for relevant refinement branches. In at least one embodiment, an ROI score from an object detection branch is combined 608 with pseudo-label scores from refinement branches to generate a set of final ROI scores for various proposals and classes. In at least one embodiment a loss function can be determined 610 using a set of final ROI scores. Network parameters for a GAN can then be updated 612 based at least in part upon a determined loss function.

[0062] FIG. 6B illustrates an example process 650 for inferring object representations in an image using such a trained model in at least one embodiment. In at least one embodiment, an image or video frame is obtained 652 that is to be used for inferencing. In at least one embodiment an image can be provided 654 as input to a trained model. In at least one embodiment a trained model can process data and infer 656 one or more classes and locations of objects represented in an image. In at least one embodiment an inferred location can correspond to segmentation for an object in at least one embodiment.

[0063] An increasing variety of industries and applications are taking advantage of machine learning. As an example, deep neural networks (DNNs) developed on processors have

been used for diverse use cases, from self-driving cars to faster drug development, from automatic image analysis for security systems to smart real-time language translation in video chat applications. Deep learning is a technique that models a neural learning process of a human brain, continually learning, continually getting smarter, and delivering more accurate results more quickly over time. A child is initially taught by an adult to correctly identify and classify various shapes, eventually being able to identify shapes without any coaching. Similarly, a deep learning or neural learning system designed to accomplish a similar task would need to be trained for it to get smarter and more efficient at identifying basic objects, occluded objects; etc., while also assigning context to those objects.

[0064] At a simplest level, neurons in a human brain look at various inputs that are received, importance levels are assigned to each of these inputs, and output is passed on to other neurons to act upon. An artificial neuron or perceptron is a most basic model off neural network. In one example, a perceptron may receive one or more inputs that represent various features of an object that a perceptron is being trained to recognize and classify, and each of these features is assigned a certain weight based on an importance of that feature in defining a shape of an object.

[0065] A deep neural network (DNN) model includes multiple layers of many connected perceptrons (e.g., nodes) that can be trained with enormous amounts of input data to quickly solve complex problems with high accuracy. In one example, a first layer of a DNN model breaks down an input image of an automobile into various sections and looks for basic patterns such as lines and angles. A second layer assembles lines to look for higher-level patterns such as wheels, windshields, and mirrors. A next layer identifies a type of vehicle, and a final few layers generate a label for an input image, identifying a model of a specific automobile brand. Once a DNN is trained, this DNN can be deployed and used to identify and classify objects or patterns in a process known as inference. Examples of inference (a process through which a DNN extracts useful information from a given input) include identifying handwritten numbers on checks deposited into ATM machines, identifying images of friends in photos, delivering movie recommendations, identifying and classifying different types of automobiles, pedestrians, and road hazards in driverless cars, or translating human speech in near real-time.

[0066] During training, data flows through a DNN in a forward propagation phase until a prediction is produced that indicates a label corresponding to this input. If a neural network does not correctly label this input, then errors between a correct label and a predicted label are analyzed, and weights are adjusted for each feature during a backward propagation phase until this DNN correctly labels this input and other inputs in a training dataset. Training complex neural networks requires massive amounts of parallel computing performance, including floating-point multiplications and additions that are supported. Inferencing is less compute-intensive than training, being a latency-sensitive process where a trained neural network is applied to new inputs it has not seen before to classify images, translate speech, and infer new information.

[0067] Neural networks rely heavily on matrix math operations, and complex multi-layered networks require tremendous amounts of floating-point performance and bandwidth for both efficiency and speed. With thousands of

processing cores, optimized for matrix math operations, and delivering tens to hundreds of TFLOPS of performance, a computing platform can deliver performance required for deep neural network-based artificial intelligence and machine learning applications.

[0068] FIG. 7 illustrates components of an example system 700 that can be used to train and utilize machine learning in accordance with at least one embodiment. As will be discussed, various components can be provided by various combinations of computing devices and resources, or a single computing system, which may be under control of a single entity or multiple entities. Further, various aspects may be triggered, initiated, or requested by different entities. For example, in at least one embodiment training of a neural network might be instructed by a provider associated with a provider environment 706, while in at least one embodiment training might be requested by a customer or other user having access to a provider environment through a client device 702 or other such resource. Training data (or data to be analyzed by a trained neural network) can be provided by a provider, a user, or a third party content provider 724. In at least one embodiment, client device 702 may be a vehicle or object that is to be navigated on behalf of a user, for example, which can submit requests and/or receive instructions that assist in navigation of this device.

[0069] In this example, requests are able to be submitted across at least one network 704 to be received to a provider environment 706. A client device may be any appropriate electronic and/or computing devices enabling a user to generate and send such requests, as may include desktop computers, notebook computers, computer servers, smartphones, tablet computers, gaming consoles (portable or otherwise), computer processors, computing logic, and set-top boxes. Network(s) 704 can include any appropriate network for transmitting a request or other such data, as may include an Internet, an intranet, an Ethernet, a cellular network, a local area network (LAN), a network of direct wireless connections among peers, and so on.

[0070] Requests can be received to an interface layer 708, which can forward data to a training and inference manager 710 in this example. This manager can be a system or service including hardware and software for managing requests and service corresponding data or content in at least one embodiment. This manager can receive a request to train a neural network, and can provide data for this request to a training manager 712. Training manager 712 can select an appropriate model or network to be used, if not specified by this request, and can train this model using relevant training data. In at least one embodiment training data can be a batch of data stored to a training data repository 714, received from client device 702 or obtained from a third party provider 724. Training manager 712 can be responsible for training this data, such as by using a LARC-based approach as discussed herein. A network can be any appropriate network, such as a recurrent neural network (RNN) or convolutional neural network (CNN). Once a network is trained and successfully evaluated, this trained network can be stored to a model repository 716, for example, that may store different models or networks for users, applications, or services, etc. As mentioned, in at least one embodiment there may be multiple models for a single application or entity, as may be utilized based on a number of different factors.

[0071] At a subsequent point in time, a request may be received from a client device 702 (or another such device)

for content (e.g., path determinations) or data that is at least partially determined or impacted by a trained neural network. A request can include, for example, input data to be processed using a neural network to obtain one or more inferences or other output values, classifications, or predictions. Input data can be received to interface layer 708 and directed to inference module 718, although a different system or service can be used as well in at least one embodiment. Inference module 718 can obtain an appropriate trained network, such as a trained deep neural network (DNN) as discussed herein, from model repository 716 if not already stored locally to inference module 718. Inference module 718 can provide data as input to a trained network, which can then generate one or more inferences as output. This may include, for example, a classification of an instance of input data. Inferences can then be transmitted to client device 702 for display or other communication to a user. Context data for a user may also be stored to a user context data repository 722, which may include data about a user which may be useful as input to a network in generating inferences, or determining data to return to a user after obtaining instances. Relevant data, which may include at least some of input or inference data, may also be stored to a local database 720 for processing future requests. In at least one embodiment, a user can use account or other information to access resources or functionality of a provider environment. If permitted and available, user data may also be collected and used to further train models, in order to provide more accurate inferences for future requests. Requests may be received through a user interface to a machine learning application 726 executing on client device 702 in at least one embodiment, and results displayed through a same interface. A client device can include resources such as a processor 728 and memory 730 for generating this request and processing results or a response, as well as at least one data storage element 732 for storing data for machine learning application 726.

[0072] In at least one embodiment a processor 728 (or a processor of training manager 712 or inference module 718) will be a central processing unit (CPU). As mentioned, however, resources in such environments can utilize GPUs to process data for at least certain types of requests. With thousands of cores, GPUs are designed to handle substantial parallel workloads and, therefore, have become popular in deep learning for training neural networks and generating predictions. While use of GPUs for offline builds has enabled faster training of larger and more complex models, generating predictions offline implies that either request-time input features cannot be used or predictions must be generated for all permutations of features and stored in a lookup table to serve real-time requests. If a deep learning framework supports a CPU-mode and model is small and simple enough to perform a feed-forward on a CPU with a reasonable latency, then a service on a CPU instance could host a model. In this case, training can be done offline on a GPU and inference done in real-time on this CPU. If a CPU approach is not a viable choice, then a service can run on a GPU instance. Because GPUs have different performance and cost characteristics than CPUs, however, running a service that offloads a runtime algorithm to a GPU can require it to be designed differently from a CPU based service.

[0073] FIG. 8 illustrates an example system 800 that can be used to classify data, or generate inferences, in accor-

dance with at least one embodiment. Various types of predictions, labels, or other outputs can be generated for input data as well, as should be apparent in light of teachings and suggestions contained herein. Further, both supervised and unsupervised training can be used in at least one embodiment discussed herein. In this example, a set of training data 802 (e.g., classified or labeled data) is provided as input to function as training data. Training data can include instances of at least one type of object for which a neural network is to be trained, as well as information that identifies that type of object. For example, training data might include a set of images that each includes a representation of a type of object, where each image also includes, or is associated with, a label, metadata, classification, or other piece of information identifying a type of object represented in a respective image. Various other types of data may be used as training data as well, as may include text data, audio data, video data, and so on. Training data 802 in this example is provided as training input to a training manager 804. Training manager 804 can be a system or service that includes hardware and software, such as one or more computing devices executing a training application, for training a neural network (or other model or algorithm, etc.). In this example, training manager 804 receives an instruction or request indicating a type of model to be used for training. A model can be any appropriate statistical model, network, or algorithm useful for such purposes, as may include an artificial neural network, deep learning algorithm, learning classifier, Bayesian network, and so on. Training manager 804 can select an initial model, or other untrained model, from an appropriate repository 806 and utilize training data 802 to train this model, generating a trained model 808 (e.g., trained deep neural network) that can be used to classify similar types of data, or generate other such inferences. In at least one embodiment where training data is not used, an appropriate initial model can still be selected for training on input data per training manager 804.

[0074] A model can be trained in a number of different ways, as may depend in part upon a type of model selected. For example, in at least one embodiment a machine learning algorithm can be provided with a set of training data, where this model is a model artifact created by a training process. Each instance of training data contains a correct answer (e.g., classification), which can be referred to as a target or target attribute. A learning algorithm finds patterns in training data that map input data attributes to a target, an answer to be predicted, and a machine learning model is output that captures these patterns. A machine learning model can then be used to obtain predictions on new data for which a target is not specified.

[0075] In one example, a training manager 804 can select from a set of machine learning models including binary classification, multiclass classification, and regression models. A type of model to be used can depend at least in part upon a type of target to be predicted. Machine learning models for binary classification problems predict a binary outcome, such as one of two possible classes. A learning algorithm such as logistic regression can be used to train binary classification models. Machine learning models for multiclass classification problems allow predictions to be generated for multiple classes, such as to predict one of more than two outcomes. Multinomial logistic regression can be useful for training multiclass models. Machine learning

models for regression problems predict a numeric value. Linear regression can be useful for training regression models.

[0076] In order to train a machine learning model in accordance with at least one embodiment, a training manager must determine an input training data source, as well as other information such as a name of a data attribute that contains a target to be predicted, required data transformation instructions, and training parameters to control a learning algorithm. During a training process, a training manager **804** in at least one embodiment may automatically select an appropriate learning algorithm based on a type of target specified in a training data source. Machine learning algorithms can accept parameters used to control certain properties of a training process and of a resulting machine learning model. These are referred to herein as training parameters. If no training parameters are specified, a training manager can utilize default values that are known to work well for a large range of machine learning tasks. Examples of training parameters for which values can be specified include maximum model size, maximum number of passes over training data, shuffle type, regularization type, learning rate, and regularization amount. Default settings may be specified, with choices to adjust values to fine-tune performance.

[0077] A maximum model size is a total size, in units of bytes, of patterns that are created during training of a model. A model may be created of a specified size by default, such as a model of 100 MB. If a training manager is unable to determine enough patterns to fill model size, a smaller model may be created. If a training manager finds more patterns than will fit into a specified size, a maximum cut-off may be enforced by trimming patterns that least affect quality of a learned model. Choosing a model size provides for control of a trade-off between a predictive quality of a model and a cost of use. Smaller models can cause a training manager to remove many patterns to fit within a maximum size limit, affecting quality of predictions. Larger models may cost more to query for real-time predictions. Larger input data sets do not necessarily result in larger models because models store patterns, not input data. If patterns are few and simple, a resulting model will be small. Input data that has a large number of raw attributes (input columns) or derived features (outputs of data transformations) will likely have more patterns found and stored during a training process.

[0078] In at least one embodiment, training manager **804** can make multiple passes or iterations over training data to attempt to discover patterns. There may be a default number of passes, such as ten passes, while in at least one embodiment up to a maximum number of passes may be set, such as up to one hundred passes. In at least one embodiment there may be no maximum set, or there may be a convergence criterion or other factor set that will trigger an end to a training process. In at least one embodiment training manager **804** can monitor quality of patterns during training, and can automatically stop training when there are no more data points or patterns to discover. Data sets with only a few observations may require more passes over data to obtain sufficiently high model quality. Larger data sets may contain many similar data points, which can reduce a need for a large number of passes. Potential impact of choosing more data passes over data is that model training can take longer and cost more in terms of resources and system utilization.

[0079] In at least one embodiment training data is shuffled before training, or between passes of training. Shuffling in at least one embodiment is a random or pseudo-random shuffling to generate a truly random ordering, although there may be some constraints in place to ensure that there is no grouping of certain types of data, or shuffled data may be reshuffled if such grouping exists, etc. Shuffling changes an order or arrangement in which data is utilized for training so that a training algorithm does not encounter groupings of similar types of data, or a single type of data for too many observations in succession. For example, a model might be trained to predict an object. Data might be sorted by object type before uploading. An algorithm can then process data alphabetically by object type, encountering only data for a certain object type first. A model will begin to learn patterns for that type of object. A model will then encounter only data for a second object type, and will try to adjust a model to fit that object type, which can degrade patterns that fit that a first object type. This sudden switch from between object types can produce a model that does not learn how to predict object types accurately. Shuffling can be performed in at least one embodiment before a training data set is split into training and evaluation subsets, such that a relatively even distribution of data types is utilized for both stages. In at least one embodiment training manager **804** can automatically shuffle data using, for example, a pseudo-random shuffling technique.

[0080] When creating a machine learning model, training manager **804** in at least one embodiment can enable a user to specify settings. For example, a user may specify one or more evaluation settings, indicating a portion of input data to be reserved for evaluating predictive quality of a machine learning model. A user may specify a policy that indicates which attributes and attribute transformations are available for model training. A user may also specify various training parameters that control certain properties of a training process and of a resulting model.

[0081] Once a training manager has determined that training of a model is complete, such as by using at least one end criterion discussed herein, trained model **808** can be provided for use by a classifier **814** in classifying (or otherwise generating inferences for) validation data **812**. As illustrated, this involves a logical transition between a training mode for a model and an inference mode for a model. In at least one embodiment, however, trained model **808** will first be passed to an evaluator **810**, which may include an application, process, or service executing on at least one computing resource (e.g., a CPU or GPU of at least one server) for evaluating quality (or another such aspect) of a trained model. A model is evaluated to determine whether this model will provide at least a minimum acceptable or threshold level of performance in predicting a target on new and future data. If not, training manager **804** can continue to train a model. Since future data instances will often have unknown target values, it can be desirable to check an accuracy metric of machine learning on data for which a target answer is known, and use this assessment as a proxy for predictive accuracy on future data.

[0082] In at least one embodiment, a model is evaluated using a subset of training data **802** that was provided for training. A subset can be determined using a shuffle and split approach as discussed above. This evaluation data subset will be labeled with a target, and thus can act as a source of ground truth for evaluation. Evaluating a predictive accu-

racy of a machine learning model with same data that was used for training is not useful, as positive evaluations might be generated for models that remember training data instead of generalizing from it. Once training has completed, an evaluation data subset is processed using trained model **808** and evaluator **810** can determine accuracy of a model by comparing ground truth data against corresponding output (or predictions/observations) of a model. Evaluator **810** in at least one embodiment can provide a summary or performance metric indicating how well predicted and true values match. If a trained model does not satisfy at least a minimum performance criterion, or other such accuracy threshold, then training manager **804** can be instructed to perform further training, or in some instances try training a new or different model. If trained model **808** satisfies relevant criteria, then a trained model can be provided for use by classifier **814**.

[0083] When creating and training a machine learning model, it can be desirable in at least one embodiment to specify model settings or training parameters that will result in a model capable of making most accurate predictions. Example parameters include a number of passes to be performed (forward and/or backward), regularization, model size, and shuffle type. As mentioned, however, selecting model parameter settings that produce best predictive performance on evaluation data might result in an overfitting of a model. Overfitting occurs when a model has memorized patterns that occur in training and evaluation data sources, but has failed to generalize patterns in data. Overfitting often occurs when training data includes all data used in evaluation. A model that has been over fit may perform well during evaluation, but may fail to make accurate predictions on new or otherwise validation data. To avoid selecting an over fitted model as a best model, a training manager can reserve additional data to validate performance of this model. For example, a training data set might be divided into 60 percent for training, and 40 percent for evaluation or validation, which may be divided into two or more stages. After selecting model parameters that work well for evaluation data, leading to convergence on a subset of validation data, such as half validation data, a second validation may be executed with a remainder of validation data to ensure performance of this model. If this model meets expectations on validation data, then this model is not overfitting data. Alternatively, a test set or held-out set may be used for testing parameters. Using a second validation or testing step helps to select appropriate model parameters to prevent overfitting. However, holding out more data from a training process for validation makes less data available for training. This may be problematic with smaller data sets as there may not be sufficient data available for training. One approach in such a situation is to perform cross-validation as discussed elsewhere herein.

[0084] There are many metrics or insights that can be used to review and evaluate a predictive accuracy of a given model. One example evaluation outcome contains a prediction accuracy metric to report on overall success of a model, as well as visualizations to help explore accuracy of this model beyond a prediction accuracy metric. An outcome can also provide an ability to review impact of setting a score threshold, such as for binary classification, and can generate alerts on criteria to check validity of an evaluation. Choice of a metric and visualization can depend at least in part upon a type of model being evaluated.

[0085] Once trained and evaluated satisfactorily, a trained machine learning model can be used to build or support a machine learning application. In at least one embodiment building a machine learning application is an iterative process that involves a sequence of steps. A core machine learning problem(s) can be framed in terms of what is observed and what answer this model is to predict. Data can then be collected, cleaned, and prepared to make this data suitable for consumption by machine learning model training algorithms. This data can be visualized and analyzed to run sanity checks to validate quality of this data and to understand this data. Raw data (e.g., input variables) and answer data (e.g., target) may not be represented in a way that can be used to train a highly predictive model. Therefore, it may be desirable to construct more predictive input representations or features from raw variables. Resulting features can be fed to a learning algorithm to build models and evaluate quality of models on data that was held out from model building. A model can then be used to generate predictions of a target answer for new data instances.

[0086] In an example system **800** of FIG. 8, trained model **810** after evaluation is provided, or made available, to a classifier **814** that is able to use this trained model to process validation data. This may include, for example, data received from users or third parties that are not classified, such as query images that are looking for information about what is represented in those images. This validation data can be processed by a classifier using this trained model, and results **816** that are produced can be sent back to respective sources or otherwise processed or stored. In at least one embodiment, and where such usage is permitted, now-classified data instances can be stored to a training data repository, which can be used for further training of trained model **808** by a training manager. In at least one embodiment a model will be continually trained as new data is available, but in at least one embodiment models will be retrained periodically, such as once a day or week, depending upon factors such as a size of a data set or complexity of a model.

[0087] Classifier **814** can include appropriate hardware and software for processing validation data **812** using a trained model. In at least one embodiment a classifier will include one or more computer servers each having one or more graphics processing units (GPUs) that are able to process data. A configuration and design of GPUs can make them more desirable to use in processing machine learning data than CPUs or other such components. A trained model in at least one embodiment can be loaded into GPU memory and a received data instance provided to a GPU for processing. GPUs can have a much larger number of cores than CPUs, and GPU cores can also be much less complex. Accordingly, a given GPU may be able to process thousands of data instances concurrently via different hardware threads. A GPU can also be configured to maximize floating point throughput, which can provide significant additional processing advantages for a large data set.

[0088] Even when using GPUs, accelerators, and other such hardware to accelerate tasks such as training of a model or classification of data using such a model, such tasks can still require significant time, resource allocation, and cost. For example, if a machine learning model is to be trained using 800 passes, and a data set includes 1,000,000 data instances to be used for training, then all million instances would need to be processed for each pass. Different portions

of an architecture can also be supported by different types of devices. For example, training may be performed using a set of servers at a logically centralized location, as may be offered as a service, while classification of raw data may be performed by such a service or on a client device. These devices may also be owned, operated, or controlled by a same entity or multiple entities in at least one embodiment.

[0089] FIG. 9 illustrates an example neural network 900 that can be trained or otherwise utilized in accordance with at least one embodiment. In this example a statistical model is an artificial neural network (ANN) that includes a multiple layers of nodes, including an input layer 902, an output layer 906, and multiple layers 904 of intermediate nodes, often referred to as “hidden” layers, as internal layers and nodes are typically not visible or accessible in various neural networks. Although only a few intermediate layers are illustrated for purposes of explanation, it should be understood that there is no limit to a number of intermediate layers that can be utilized, and any limit on layers will often be a factor of resources or time required for processed using this model. As discussed elsewhere herein, there can be additional types of models, networks, algorithms, or processes used as well, as may include other numbers or selections of nodes and layers. Validation data can be processed by layers of this network to generate a set of inferences, or inference scores, which can then be fed to a loss function 908.

[0090] In this example network 900, all nodes of a given layer are interconnected to all nodes of an adjacent layer. As illustrated, nodes of an intermediate layer will then each be connected to nodes of two adjacent layers. Nodes are also referred to as neurons or connected units in some models, and connections between nodes are referred to as edges. Each node can perform a function for inputs received, such as by using a specified function. Nodes and edges can obtain different weightings during training, and individual layers of nodes can perform specific types of transformations on received input, where those transformations can also be learned or adjusted during training. Learning can be supervised or unsupervised learning, as may depend at least in part upon a type of information contained in a training data set. Various types of neural networks can be utilized, as may include a convolutional neural network (CNN) that includes a number of convolutional layers and a set of pooling layers, and have proven to be beneficial for applications such as image recognition. CNNs can also be easier to train than other networks due to a relatively small number of parameters to be determined.

[0091] In at least one embodiment, such a complex machine learning model can be trained using various tuning parameters. Choosing parameters, fitting a model, and evaluating this model are parts of a model tuning process, often referred to as hyperparameter optimization. Such tuning can involve introspecting an underlying model or data in at least one embodiment. In a training or production setting, a robust workflow can be important to avoid overfitting of hyperparameters as discussed elsewhere herein. Cross-validation and adding Gaussian noise to a training dataset are techniques that can be useful for avoiding overfitting to any one dataset. For hyperparameter optimization it may be desirable in at least one embodiment to keep training and validation sets fixed. In at least one embodiment, hyperparameters can be tuned in certain categories, as may include data preprocessing, CNN architecture definition (for example, filter sizes, number of filters), stochastic gradient

descent (SGD) parameters (for example, learning rate), and regularization (for example, dropout probability).

[0092] In an example pre-processing step, instances of a dataset can be embedded into a lower dimensional space of a certain size. A size of this space is a parameter to be tuned. An architecture of this CNN contains many tunable parameters. A parameter for filter sizes can represent an interpretation of information that corresponds to size of an instance that will be analyzed. In computational linguistics, this is known as an n-gram size. An example CNN uses three different filter sizes, which represent potentially different n-gram sizes. A number of filters per filter size can correspond to a depth of this filter. Each filter attempts to learn something different from a structure of an instance, such as a sentence structure for textual data. In a convolutional layer, an activation function can be a rectified linear unit and pooling type set as max pooling. Results can then be concatenated into a single dimensional vector, and a last layer is fully connected onto a two-dimensional output. This corresponds to binary classification to which an optimization function can be applied. One such function is an implementation of a Root Mean Square (RMS) propagation method of gradient descent, where example hyperparameters can include learning rate, batch size, maximum gradient normal, and epochs. With neural networks, regularization can be an extremely important consideration. As mentioned, in at least one embodiment input data may be relatively sparse. A main hyperparameter in such a situation can be a dropout at a penultimate layer, which represents a proportion of nodes that will not “fire” at each training cycle. An example training process can suggest different hyperparameter configurations based on feedback for performance of previous configurations. A model can be trained with a proposed configuration, evaluated on a designated validation set, and performance reporting. This process can be repeated to, for example, trade off exploration (learning more about different configurations) and exploitation (leveraging previous knowledge to achieve better results).

[0093] As training CNNs can be parallelized and GPU-enabled computing resources can be utilized, multiple optimization strategies can be attempted for different scenarios. A complex scenario allows tuning model architecture and preprocessing and stochastic gradient descent parameters. This expands a model configuration space. In a basic scenario, only preprocessing and stochastic gradient descent parameters are tuned. There can be a greater number of configuration parameters in this complex scenario than in a basic scenario. Tuning in a joint space can be performed using a linear or exponential number of steps, iteration through an optimization loop for models. A cost for such a tuning process can be significantly less than for tuning processes such as random search and grid search, without any significant performance loss.

[0094] In at least one embodiment backpropagation can be utilized to calculate a gradient used for determining weights for a neural network. Backpropagation is a form of differentiation, and can be used by a gradient descent optimization algorithm to adjust weights applied to various nodes or neurons as discussed above. Weights can be determined in at least one embodiment using gradient of a relevant loss function. Backpropagation can utilize a derivative of a loss function with respect to an output generated by a statistical model. As mentioned, various nodes can have associated activation functions that define output of respective nodes.

Various activation functions can be used as appropriate, as may include radial basis functions (RBFs) and sigmoids, which can be utilized by various support vector machines (SVMs) for transformation of data. An activation function of an intermediate layer of nodes is referred to herein as an inner product kernel. These functions can include, for example, identity functions, step functions, sigmoidal functions, ramp functions, and so on. Activation functions can also be linear or non-linear.

Inference and Training Logic

[0095] FIG. 10 illustrates inference and/or training logic 1015 used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 1015 are provided below in conjunction with FIGS. 10 and/or 11.

[0096] In at least one embodiment, inference and/or training logic 1015 may include, without limitation, a data storage 1001 to store forward 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 data storage 1001 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 data storage 1001 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0097] In at least one embodiment, any portion of data storage 1001 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, data storage 1001 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, choice of whether data storage 1001 is internal or external to a processor, for example, or comprised of 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.

[0098] In at least one embodiment, inference and/or training logic 1015 may include, without limitation, a data storage 1005 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, data storage 1005 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, any portion of data storage 1005 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 data storage 1005 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, data

storage 1005 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., Flash memory), or other storage. In at least one embodiment, choice of whether data storage 1005 is internal or external to a processor, for example, or comprised of 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.

[0099] In at least one embodiment, data storage 1001 and data storage 1005 may be separate storage structures. In at least one embodiment, data storage 1001 and data storage 1005 may be same storage structure. In at least one embodiment, data storage 1001 and data storage 1005 may be partially same storage structure and partially separate storage structures. In at least one embodiment, any portion of data storage 1001 and data storage 1005 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0100] In at least one embodiment, inference and/or training logic 1015 may include, without limitation, one or more arithmetic logic unit(s) ("ALU(s)") 1010 to perform logical and/or mathematical operations based, at least in part on, or indicated by, training and/or inference code, result of which may result in activations (e.g., output values from layers or neurons within a neural network) stored in an activation storage 1020 that are functions of input/output and/or weight parameter data stored in data storage 1001 and/or data storage 1005. In at least one embodiment, activations stored in activation storage 1020 are generated according to linear algebraic and or matrix-based mathematics performed by ALU(s) 1010 in response to performing instructions or other code, wherein weight values stored in data storage 1005 and/or data 1001 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 data storage 1005 or data storage 1001 or another storage on or off-chip. In at least one embodiment, ALU(s) 1010 are included within one or more processors or other hardware logic devices or circuits, whereas in another embodiment, ALU(s) 1010 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 1010 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, data storage 1001, data storage 1005, and activation storage 1020 may be on same 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 1020 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.

[0101] In at least one embodiment, activation storage **1020** may be cache memory, DRAM, SRAM, non-volatile memory (e.g., Flash memory), or other storage. In at least one embodiment, activation storage **1020** may be completely or partially within or external to one or more processors or other logical circuits. In at least one embodiment, choice of whether activation storage **1020** is internal or external to a processor, for example, or comprised of 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. In at least one embodiment, inference and/or training logic **1015** illustrated in FIG. **10** may be used in conjunction with an application-specific integrated circuit (“ASIC”), such as Tensorflow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, inference and/or training logic **1015** illustrated in FIG. **10** 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”).

[0102] FIG. **11** illustrates inference and/or training logic **1015**, in at least one embodiment various. In at least one embodiment, inference and/or training logic **1015** may include, without limitation, hardware logic in which computational resources are dedicated or otherwise exclusively used in conjunction with weight values or other information corresponding to one or more layers of neurons within a neural network. In at least one embodiment, inference and/or training logic **1015** illustrated in FIG. **11** may be used in conjunction with an application-specific integrated circuit (ASIC), such as Tensorflow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., “Lake Crest”) processor from Intel Corp. In at least one embodiment, inference and/or training logic **1015** illustrated in FIG. **11** may be used in conjunction with central processing unit (CPU) hardware, graphics processing unit (GPU) hardware or other hardware, such as field programmable gate arrays (FPGAs). In at least one embodiment, inference and/or training logic **1015** includes, without limitation, data storage **1001** and data storage **1005**, which may be used to store 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. **10.B**, each of data storage **1001** and data storage **1005** is associated with a dedicated computational resource, such as computational hardware **1002** and computational hardware **1006**, respectively. In at least one embodiment, each of computational hardware **1002** and computational hardware **1006** comprises one or more ALUs that perform mathematical functions, such as linear algebraic functions, only on information stored in data storage **1001** and data storage **1005**, respectively, result of which is stored in activation storage **1020**.

[0103] In at least one embodiment, each of data storage **1001** and **1005** and corresponding computational hardware **1002** and **1006**, respectively, correspond to different layers of a neural network, such that resulting activation from one “storage/computational pair **1001/1002**” of data storage **1001** and computational hardware **1002** is provided as an

input to next “storage/computational pair **1005/1006**” of data storage **1005** and computational hardware **1006**, in order to mirror conceptual organization of a neural network. In at least one embodiment, each of storage/computational pairs **1001/1002** and **1005/1006** 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 **1001/1002** and **1005/1006** may be included in inference and/or training logic **1015**.

Neural Network Training and Deployment

[0104] FIG. **12** illustrates training and deployment of a deep neural network, in at least one embodiment. In at least one embodiment, untrained neural network **1206** is trained using a training dataset **1202**. In at least one embodiment, training framework **1104** is a PyTorch framework, whereas in other embodiments, training framework **1104** is a Tensorflow, Boost, Caffe, Microsoft Cognitive Toolkit/CNTK, MXNet, Chainer, Keras, Deeplearning4j, or other training framework. In at least one embodiment training framework **1104** trains an untrained neural network **1106** and enables it to be trained using processing resources described herein to generate a trained neural network **1108**. 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.

[0105] In at least one embodiment, untrained neural network **1106** is trained using supervised learning, wherein training dataset **1102** includes an input paired with a desired output for an input, or where training dataset **1102** includes input having known output and output of a neural network is manually graded. In at least one embodiment, untrained neural network **1106** is trained in a supervised manner processes inputs from training dataset **1102** 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 **1106**. In at least one embodiment, training framework **1104** adjusts weights that control untrained neural network **1106**. In at least one embodiment, training framework **1104** includes tools to monitor how well untrained neural network **1106** is converging towards a model, such as trained neural network **1108**, suitable to generating correct answers, such as in result **1114**, based on known input data, such as new data **1112**. In at least one embodiment, training framework **1104** trains untrained neural network **1106** repeatedly while adjust weights to refine an output of untrained neural network **1106** using a loss function and adjustment algorithm, such as stochastic gradient descent. In at least one embodiment, training framework **1104** trains untrained neural network **1106** until untrained neural network **1106** achieves a desired accuracy. In at least one embodiment, trained neural network **1108** can then be deployed to implement any number of machine learning operations.

[0106] In at least one embodiment, untrained neural network **1106** is trained using unsupervised learning, wherein untrained neural network **1106** attempts to train itself using unlabeled data. In at least one embodiment, unsupervised learning training dataset **1102** will include input data without any associated output data or “ground truth” data. In at least one embodiment, untrained neural network **1106** can learn groupings within training dataset **1102** and can determine

how individual inputs are related to untrained dataset 1102. In at least one embodiment, unsupervised training can be used to generate a self-organizing map, which is a type of trained neural network 1108 capable of performing operations useful in reducing dimensionality of new data 1112. In at least one embodiment, unsupervised training can also be used to perform anomaly detection, which allows identification of data points in a new dataset 1112 that deviate from normal patterns of new dataset 1112.

[0107] In at least one embodiment, semi-supervised learning may be used, which is a technique in which in training dataset 1102 includes a mix of labeled and unlabeled data. In at least one embodiment, training framework 1104 may be used to perform incremental learning, such as through transferred learning techniques. In at least one embodiment, incremental learning enables trained neural network 1108 to adapt to new data 1112 without forgetting knowledge instilled within network during initial training.

Data Center

[0108] FIG. 13 illustrates an example data center 1300, in which at least one embodiment may be used. In at least one embodiment, data center 1300 includes a data center infrastructure layer 1310, a framework layer 1320, a software layer 1330 and an application layer 1340.

[0109] In at least one embodiment, as shown in FIG. 13, data center infrastructure layer 1310 may include a resource orchestrator 1312, grouped computing resources 1314, and node computing resources (“node C.R.s”) 1316(1)-1316(N), where “N” represents any whole, positive integer. In at least one embodiment, node C.R.s 1316(1)-1316(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 devices (e.g., dynamic read-only memory), storage devices (e.g., solid state 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 1316(1)-1316(N) may be a server having one or more of above-mentioned computing resources.

[0110] In at least one embodiment, grouped computing resources 1314 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). separate groupings of node C.R.s within grouped computing resources 1314 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.

[0111] In at least one embodiment, resource orchestrator 1322 may configure or otherwise control one or more node C.R.s 1316(1)-1316(N) and/or grouped computing resources 1314. In at least one embodiment, resource orchestrator 1322 may include a software design infrastructure (“SDI”) management entity for data center 1300. In at least one embodiment, resource orchestrator may include hardware, software or some combination thereof.

[0112] In at least one embodiment, as shown in FIG. 13, framework layer 1320 includes a job scheduler 1332, a configuration manager 1334, a resource manager 1336 and a distributed file system 1338. In at least one embodiment, framework layer 1320 may include a framework to support software 1332 of software layer 1330 and/or one or more application(s) 1342 of application layer 1340. In at least one embodiment, software 1332 or application(s) 1342 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 1320 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 1338 for large-scale data processing (e.g., “big data”). In at least one embodiment, job scheduler 1332 may include a Spark driver to facilitate scheduling of workloads supported by various layers of data center 1300. In at least one embodiment, configuration manager 1334 may be capable of configuring different layers such as software layer 1330 and framework layer 1320 including Spark and distributed file system 1338 for supporting large-scale data processing. In at least one embodiment, resource manager 1336 may be capable of managing clustered or grouped computing resources mapped to or allocated for support of distributed file system 1338 and job scheduler 1332. In at least one embodiment, clustered or grouped computing resources may include grouped computing resource 1314 at data center infrastructure layer 1310. In at least one embodiment, resource manager 1336 may coordinate with resource orchestrator 1312 to manage these mapped or allocated computing resources.

[0113] In at least one embodiment, software 1332 included in software layer 1330 may include software used by at least portions of node C.R.s 1316(1)-1316(N), grouped computing resources 1314, and/or distributed file system 1338 of framework layer 1320. 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.

[0114] In at least one embodiment, application(s) 1342 included in application layer 1340 may include one or more types of applications used by at least portions of node C.R.s 1316(1)-1316(N), grouped computing resources 1314, and/or distributed file system 1338 of framework layer 1320. one or more types of applications may include, but are not limited to, any number of a genomics application, a cognitive compute, 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.

[0115] In at least one embodiment, any of configuration manager 1334, resource manager 1336, and resource orchestrator 1312 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 1300 from making possibly bad configuration decisions and possibly avoiding underutilized and/or poor performing portions of a data center.

[0116] In at least one embodiment, data center 1300 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 1300. 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 1300 by using weight parameters calculated through one or more training techniques described herein.

[0117] 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.

[0118] Inference and/or training logic 1015 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 1015 are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, inference and/or training logic 1015 may be used in system FIG. 13 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0119] In accordance with at least one embodiment, a data center infrastructure 1310 can receive input text and cause that input to be directed to corresponding components of application layer 1340 and software layer 1330 for purposes of training and/or inferencing as discussed herein.

Computer Systems

[0120] FIG. 14 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 1400 formed with a processor that may include execution units to execute an instruction, in at least one embodiment. In at least one embodiment, computer system 1400 may include, without limitation, a component, such as a processor 1402 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 1400 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, Calif., 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 1400 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.

[0121] 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.

[0122] In at least one embodiment, computer system 1400 may include, without limitation, processor 1402 that may include, without limitation, one or more execution units 1408 to perform machine learning model training and/or inferencing according to techniques described herein. In at least one embodiment, a system is a single processor desktop or server system, but in another embodiment this system may be a multiprocessor system. In at least one embodiment, processor 1402 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 1402 may be coupled to a processor bus 1410 that may transmit data signals between processor 1402 and other components in computer system 1400.

[0123] In at least one embodiment, processor 1402 may include, without limitation, a Level 1 (“L1”) internal cache memory (“cache”) 1404. In at least one embodiment, processor 1402 may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor 1402. Other embodiments may also include a combination of both internal and external caches depending on particular implementation and needs. In at least one embodiment, register file 1406 may store different types of data in various registers including, without limitation, integer registers, floating point registers, status registers, and instruction pointer register.

[0124] In at least one embodiment, execution unit 1408, including, without limitation, logic to perform integer and floating point operations, also resides in processor 1402. Processor 1402 may also include a microcode (“ucode”) read only memory (“ROM”) that stores microcode for certain macro instructions. In at least one embodiment, execution unit 1408 may include logic to handle a packed instruction set 1409. In at least one embodiment, by including packed instruction set 1409 in instruction set of a general-purpose processor 1402, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using packed data in a general-purpose processor 1402. In one or more embodiments, many multimedia applications may be accelerated and executed more efficiently by using full width of a processor’s data bus for performing operations on packed data, which may eliminate need to transfer smaller units of data across processor’s data bus to perform one or more operations one data element at a time.

[0125] In at least one embodiment, execution unit 1408 may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. In at least one embodiment, computer system 1400 may

include, without limitation, a memory **1420**. In at least one embodiment, memory **1420** may be implemented as a Dynamic Random Access Memory (“DRAM”) device, a Static Random Access Memory (“SRAM”) device, flash memory device, or other memory device. Memory **1420** may store instruction(s) **1419** and/or data **1421** represented by data signals that may be executed by processor **1402**.

[0126] In at least one embodiment, system logic chip may be coupled to processor bus **1410** and memory **1420**. In at least one embodiment, system logic chip may include, without limitation, a memory controller hub (“MCH”) **1416**, and processor **1402** may communicate with MCH **1416** via processor bus **1410**. In at least one embodiment, MCH **1416** may provide a high bandwidth memory path **1418** to memory **1420** for instruction and data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH **1416** may direct data signals between processor **1402**, memory **1420**, and other components in computer system **1400** and to bridge data signals between processor bus **1410**, memory **1420**, and a system I/O **1422**. In at least one embodiment, system logic chip may provide a graphics port for coupling to a graphics controller. In at least one embodiment, MCH **1416** may be coupled to memory **1420** through a high bandwidth memory path **1418** and graphics/video card **1412** may be coupled to MCH **1416** through an Accelerated Graphics Port (“AGP”) interconnect **1414**.

[0127] In at least one embodiment, computer system **1400** may use system I/O **1422** that is a proprietary hub interface bus to couple MCH **1416** to I/O controller hub (“ICH”) **1430**. In at least one embodiment, ICH **1430** may provide direct connections to some I/O devices via a local I/O bus. In at least one embodiment, local I/O bus may include, without limitation, a high-speed I/O bus for connecting peripherals to memory **1420**, chipset, and processor **1402**. Examples may include, without limitation, an audio controller **1429**, a firmware hub (“flash BIOS”) **1428**, a wireless transceiver **1426**, a data storage **1424**, a legacy I/O controller **1423** containing user input and keyboard interfaces, a serial expansion port **1427**, such as Universal Serial Bus (“USB”), and a network controller **1434**. Data storage **1424** may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

[0128] In at least one embodiment, FIG. **14** illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. **14A** may illustrate an exemplary System on a Chip (“SoC”). In at least one embodiment, devices illustrated in FIG. **14A** 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 system **1400** are interconnected using compute express link (CXL) interconnects.

[0129] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** may be used in system FIG. **14A** 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.

[0130] In some embodiments, a video data stream may be received over an expansion port **1427** or wireless transceiver **1426**, for example, then directed to processor **1402** and/or video graphics card **1412** for processing. Depending on whether components are part of a device such as an autonomous vehicle or a separate device, output could then go to a control system via I/O or transmitted to vehicle via wireless transceiver.

[0131] FIG. **15** is a block diagram illustrating an electronic device **1500** for utilizing a processor **1510**, in at least one embodiment. In at least one embodiment, electronic device **1500** 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.

[0132] In at least one embodiment, system **1500** may include, without limitation, processor **1510** communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor **1510** coupled using a bus or interface, such as a 1° C. 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), or a Universal Asynchronous Receiver/Transmitter (“UART”) bus. In at least one embodiment, FIG. **15** illustrates a system, which includes interconnected hardware devices or “chips”, whereas in other embodiments, FIG. **15** may illustrate an exemplary System on a Chip (“SoC”). In at least one embodiment, devices illustrated in FIG. **15** 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. **15** are interconnected using compute express link (CXL) interconnects.

[0133] In at least one embodiment, FIG. **15** may include a display **1524**, a touch screen **1525**, a touch pad **1530**, a Near Field Communications unit (“NFC”) **1545**, a sensor hub **1540**, a thermal sensor **1546**, an Express Chipset (“EC”) **1535**, a Trusted Platform Module (“TPM”) **1538**, BIOS/firmware/flash memory (“BIOS, FW Flash”) **1522**, a DSP **1560**, a drive “SSD or HDD”) **1520** such as a Solid State Disk (“SSD”) or a Hard Disk Drive (“HDD”), a wireless local area network unit (“WLAN”) **1550**, a Bluetooth unit **1552**, a Wireless Wide Area Network unit (“WWAN”) **1556**, a Global Positioning System (GPS) **1555**, a camera (“USB 3.0 camera”) **1554** such as a USB 3.0 camera, or a Low Power Double Data Rate (“LPDDR”) memory unit (“LPDDR3”) **1515** implemented in, for example, LPDDR3 standard. These components may each be implemented in any suitable manner.

[0134] In at least one embodiment, other components may be communicatively coupled to processor **1510** through components discussed above. In at least one embodiment, an accelerometer **1541**, Ambient Light Sensor (“ALS”) **1542**, compass **1543**, and a gyroscope **1544** may be communicatively coupled to sensor hub **1540**. In at least one embodiment, thermal sensor **1539**, a fan **1537**, a keyboard **1546**, and a touch pad **1530** may be communicatively coupled to EC **1535**. In at least one embodiment, speaker **1563**, a headphones **1564**, and a microphone (“mic”) **1565** may be communicatively coupled to an audio unit (“audio codec and class d amp”) **1564**, which may in turn be communicatively

coupled to DSP **1560**. In at least one embodiment, audio unit **1564** may include, for example and without limitation, an audio coder/decoder (“codec”) and a class D amplifier. In at least one embodiment, SIM card (“SIM”) **1557** may be communicatively coupled to WWAN unit **1556**. In at least one embodiment, components such as WLAN unit **1550** and Bluetooth unit **1552**, as well as WWAN unit **1556** may be implemented in a Next Generation Form Factor (“NGFF”).

[0135] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** may be used in system FIG. **15** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0136] FIG. **16** illustrates a computer system **1600**, in at least one embodiment. In at least one embodiment, computer system **1600** is configured to implement various processes and methods described throughout this disclosure.

[0137] In at least one embodiment, computer system **1600** comprises, without limitation, at least one central processing unit (“CPU”) **1602** that is connected to a communication bus **1610** 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 **1600** includes, without limitation, a main memory **1604** and control logic (e.g., implemented as hardware, software, or a combination thereof) and data are stored in main memory **1604** which may take form of random access memory (“RAM”). In at least one embodiment, a network interface subsystem (“network interface”) **1622** provides an interface to other computing devices and networks for receiving data from and transmitting data to other systems from computer system **1600**.

[0138] In at least one embodiment, computer system **1600**, in at least one embodiment, includes, without limitation, input devices **1608**, parallel processing system **1612**, and display devices **1606** which can be implemented using a cathode ray tube (“CRT”), liquid crystal display (“LCD”), light emitting diode (“LED”), plasma display, or other suitable display technologies. In at least one embodiment, user input is received from input devices **1608** such as keyboard, mouse, touchpad, microphone, and more. In at least one embodiment, each of foregoing modules can be situated on a single semiconductor platform to form a processing system.

[0139] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** may be used in system FIG. **16** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

[0140] FIG. **17** illustrates a computer system **1700**, in at least one embodiment. In at least one embodiment, computer system **1700** includes, without limitation, a computer **1710** and a USB stick **1720**. In at least one embodiment, computer **1710** may include, without limitation, any number and type of processor(s) (not shown) and a memory (not shown). In at least one embodiment, computer **1710** includes, without limitation, a server, a cloud instance, a laptop, and a desktop computer.

[0141] In at least one embodiment, USB stick **1720** includes, without limitation, a processing unit **1730**, a USB interface **1740**, and USB interface logic **1750**. In at least one embodiment, processing unit **1730** may be any instruction execution system, apparatus, or device capable of executing instructions. In at least one embodiment, processing unit **1730** may include, without limitation, any number and type of processing cores (not shown). In at least one embodiment, processing core **1730** 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 core **1730** is a tensor processing unit (“TPC”) that is optimized to perform machine learning inference operations. In at least one embodiment, processing core **1730** is a vision processing unit (“VPU”) that is optimized to perform machine vision and machine learning inference operations.

[0142] In at least one embodiment, USB interface **1740** may be any type of USB connector or USB socket. For instance, in at least one embodiment, USB interface **1740** is a USB 3.0 Type-C socket for data and power. In at least one embodiment, USB interface **1740** is a USB 3.0 Type-A connector. In at least one embodiment, USB interface logic **1750** may include any amount and type of logic that enables processing unit **1730** to interface with or devices (e.g., computer **1710**) via USB connector **1740**.

[0143] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** may be used in system FIG. **17** 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.

[0144] 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, in 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**, UART controller **1830**, an SPI/SDIO controller **1835**, and an I.sup.2S/I.sup.2C 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, 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**.

[0145] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** 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.

[0146] For example, inference and/or training logic **1015** can accept an input video stream and generate inferences for objects represented in video stream as discussed herein. Image processor **1815** can be used to process video frames as they are received, in at least some embodiments.

[0147] 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.

[0148] 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, in 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, in 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**.

[0149] 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.

[0150] 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 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.

[0151] In at least one embodiment, graphics processor **1940** includes one or more MMU(s) **1920A-1920B**, caches **1925A-1925B**, and circuit interconnects **1930A-1930B** of graphics processor **1910** of FIG. **19A**. 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**), 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.

[0152] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** may be used in integrated circuit **19A** and/or **19B** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein. For example, inference and/or training logic **1015** can accept an input video stream and generate inferences for objects represented in video stream as discussed herein.

[0153] FIGS. **20A-20B** illustrate additional exemplary graphics processor logic according to embodiments described herein. FIG. **20A** illustrates a graphics core **2000** that may be included within graphics processor **1810** of FIG. **18**, in at least one embodiment, and may be a unified shader core **1955A-1955N** as in FIG. **19B** in at least one embodiment.

ment. FIG. 20B illustrates a highly-parallel general-purpose graphics processing unit 2030 suitable for deployment on a multi-chip module in at least one embodiment.

[0154] In one embodiment, graphics core 2000 includes a shared instruction cache 2002, a texture unit 2018, and a cache/shared memory 2020 that may be shared with execution resources within graphics core 2000. In one embodiment, graphics core 2000 can include multiple slices 2001A-2001N or partition for each core, and a graphics processor can include multiple instances of graphics core 2000. Slices 2001A-2001N can include support logic including a local instruction cache 2004A-2004N, a thread scheduler 2006A-2006N, a thread dispatcher 2008A-2008N, and a set of registers 2010A-2010N. In one embodiment, slices 2001A-2001N can include a set of additional function units (AFUs 2012A-2012N), floating-point units (FPU 2014A-2014N), integer arithmetic logic units (ALUs 2016-2016N), address computational units (ACU 2013A-2013N), double-precision floating-point units (DPFPUs 2015A-2015N), and matrix processing units (MPU 2017A-2017N).

[0155] In 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 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 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 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 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.).

[0156] Inference and/or training logic 1015 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 1015 are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, inference and/or training logic 1015 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.

[0157] FIG. 20B illustrates a general-purpose processing unit (GPGPU) 2030 that can be configured to enable highly-parallel compute operations to be performed by an array of graphics processing units, in at least one embodiment. In at least one embodiment, GPGPU 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 to distribute execution threads asso-

ciated 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.

[0158] In at least one embodiment, GPGPU 2030 includes memory 2044A-2044B coupled with compute clusters 2036A-2036H via a set of memory controllers 2042A-2042B. In at least one embodiment, memory 2044A-2044B can include various types of memory devices including dynamic random access memory (DRAM) or graphics random access memory, such as synchronous graphics random access memory (SGRAM), including graphics double data rate (GDDR) memory.

[0159] 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.

[0160] 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.

[0161] 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 a inferencing platform. In at least one embodiment, in which GPGPU 2030 is used for inferencing, GPGPU may include fewer compute clusters 2036A-2036H relative to when GPGPU 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, 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.

[0162] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, inference and/or training logic **1015** 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.

[0163] FIG. 21 is a block diagram illustrating a computing system **2100** in 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.

[0164] 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 be 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, one or more 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, one or more 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**.

[0165] 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 wireless network adapter **2119** that may be integrated into a 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.

[0166] 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 so on, 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.

[0167] In at least one embodiment, one or more 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). In at least one embodiment, one or more parallel processor(s) **2112** incorporate circuitry optimized for general purpose processing. In at least embodiment, components of computing system **2100** may be integrated with one or more other system elements on a single integrated circuit. For example, in at least one embodiment, one or more 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.

[0168] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, inference and/or training logic **1015** may be used in system FIG. 2100 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

Processors

[0169] FIG. 22 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.

[0170] 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 memory hub **2105**. In at least one embodiment, connections between memory hub **2105** and I/O unit **2204** form a communication link **2113**. 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.

[0171] In at least one embodiment, when host interface **2206** receives a command buffer via I/O unit **2204**, host interface **2206** can direct work operations to perform those commands to a front end **2208**. In at least one embodiment, front end **2208** couples with a scheduler **2210**, which is configured to distribute commands or other work items to a processing cluster array **2212**. In at least one embodiment, scheduler **2210** ensures that processing cluster array **2212** is properly configured, and in a valid state, before tasks are distributed to processing cluster array **2212** 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 prove workloads for scheduling on processing array **2212** via one of multiple graphics processing doorbells. In at least one embodiment, workloads can then be automatically distributed across processing array **2212** by scheduler **2210** logic within a microcontroller including scheduler **2210**.

[0172] 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**). 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.

[0173] 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.

[0174] 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.

[0175] 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.

[0176] 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, such as may include 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.

[0177] In at least one embodiment, each of one or more instances of parallel processing unit **2202** can couple with 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 Nth partition unit **2220N** has a corresponding Nth memory unit **2224N**. In at least one embodiment, a number of partition units **2220A-2220N** may not be equal to a number of memory devices.

[0178] 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). 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.

[0179] 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**.

[0180] 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 inter-operate 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.

[0181] FIG. 23 is a block diagram of a partition unit **2320** in at least one embodiment. In at least one embodiment,

partition unit **2320** is an instance of one of partition units **2220A-2220N** of FIG. 22. In at least one embodiment, partition unit **2320** includes an L2 cache **2321**, a frame buffer interface **2325**, and a ROP **2326** (raster operations unit). L2 cache **2321** is a read/write cache that is configured to perform load and store operations received from memory crossbar **2316** and ROP **2326**. In at least one embodiment, read misses and urgent write-back requests are output by L2 cache **2321** to frame buffer interface **2325** for processing. In at least one embodiment, updates can also be sent to a frame buffer via frame buffer interface **2325** for processing. In at least one embodiment, frame buffer interface **2325** interfaces with one of memory units in parallel processor memory, such as memory units **2224A-2224N** of FIG. 22 (e.g., within parallel processor memory **2222**).

[0182] In at least one embodiment, ROP **2326** is a processing unit that performs raster operations such as stencil, z test, blending, and like. In at least one embodiment, ROP **2326** then outputs processed graphics data that is stored in graphics memory. In at least one embodiment, ROP **2326** 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. A type of compression that is performed by ROP **2326** 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.

[0183] In at least one embodiment, ROP **2326** is included within each processing cluster (e.g., cluster **2214A-2214N** of FIG. 22) instead of within partition unit **2320**. In at least one embodiment, read and write requests for pixel data are transmitted over memory crossbar **2316** 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. 22.

[0184] FIG. 24 is a block diagram of a processing cluster **2414** within a parallel processing unit in at least one embodiment. In at least one embodiment, a processing cluster is an instance of one of processing clusters **2214A-2214N** of FIG. 22. In at least one embodiment, processing cluster **2414** can be configured to execute many threads in parallel, where a term “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 synchronized threads, using a instruction unit configured to issue instructions to a set of processing engines within each one of processing clusters.

[0185] In at least one embodiment, operation of processing cluster **2414** can be controlled via a pipeline manager **2432** that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager **2432** receives instructions from scheduler **2210** of FIG. 22 and manages execution of those instructions via a graphics

multiprocessor **2434** and/or a texture unit **2436**. In at least one embodiment, graphics multiprocessor **2434** is an exemplary instance of a SIMD parallel processor. However, in at least one embodiment, various types of SIMD parallel processors of differing architectures may be included within processing cluster **2414**. In at least one embodiment, one or more instances of graphics multiprocessor **2434** can be included within a processing cluster **2414**. In at least one embodiment, graphics multiprocessor **2434** can process data and a data crossbar **2440** can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager **2432** can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data crossbar **2440**.

[0186] In at least one embodiment, each graphics multiprocessor **2434** within processing cluster **2414** 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, a same functional-unit hardware can be leveraged to perform different operations and any combination of functional units may be present.

[0187] In at least one embodiment, instructions transmitted to processing cluster **2414** 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 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 **2434**. In at least one embodiment, a thread group may include fewer threads than a number of processing engines within graphics multiprocessor **2434**. 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 **2434**. In at least one embodiment, when a thread group includes more threads than a number of processing engines within graphics multiprocessor **2434**, processing can be performed over consecutive clock cycles. In at least one embodiment, multiple thread groups can be executed concurrently on a graphics multiprocessor **2434**.

[0188] In at least one embodiment, graphics multiprocessor **2434** includes an internal cache memory to perform load and store operations. In at least one embodiment, graphics multiprocessor **2434** can forego an internal cache and use a cache memory (e.g., L1 cache **2448**) within processing cluster **2414**. In at least one embodiment, each graphics multiprocessor **2434** also has access to L2 caches within partition units (e.g., partition units **2220A-2220N** of FIG. **22**) that are shared among all processing clusters **2414** and may be used to transfer data between threads. In at least one embodiment, graphics multiprocessor **2434** 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 **2402** may be used as global memory. In at least one embodiment, processing cluster **2414** includes multiple instances of graphics multiprocessor **2434** can share instructions and data, which may be stored in L1 cache **2448**.

[0189] In at least one embodiment, each processing cluster **2414** may include an MMU **2445** (memory management unit) that is configured to map virtual addresses into physical addresses. In at least one embodiment, one or more instances of MMU **2445** may reside within memory interface **2218** of FIG. **22**. In at least one embodiment, MMU **2445** includes a set of page table entries (PTEs) used to map a virtual address to a physical address of a tile (talk more about tiling) and a cache line index in at least one embodiment. In at least one embodiment, MMU **2445** may include address translation lookaside buffers (TLB) or caches that may reside within graphics multiprocessor **2434** or L1 cache or processing cluster **2414**. In at least one embodiment, physical address is processed to distribute surface data access locality to allow efficient request interleaving among partition units. In at least one embodiment, cache line index may be used to determine whether a request for a cache line is a hit or miss.

[0190] In at least one embodiment, a processing cluster **2414** may be configured such that each graphics multiprocessor **2434** is coupled to a texture unit **2436** for performing texture mapping operations, such as may involve 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 **2434** and is fetched from an L2 cache, local parallel processor memory, or system memory, as needed. In at least one embodiment, each graphics multiprocessor **2434** outputs processed tasks to data crossbar **2440** to provide processed task to another processing cluster **2414** for further processing or to store processed task in an L2 cache, local parallel processor memory, or system memory via memory crossbar **2416**. In at least one embodiment, preROP **2442** (pre-raster operations unit) is configured to receive data from graphics multiprocessor **2434**, direct data to ROP units, which may be located with partition units as described herein (e.g., partition units **2220A-2220N** of FIG. **22**). In at least one embodiment, PreROP **2442** unit can perform optimizations for color blending, organize pixel color data, and perform address translations.

[0191] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, inference and/or training logic **1015** 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.

[0192] FIG. **25** shows a graphics multiprocessor **2534** in at least one embodiment. In at least one embodiment, graphics multiprocessor **2534** couples with pipeline manager **2532** of processing cluster **2514**. In at least one embodiment, graphics multiprocessor **2534** has an execution pipeline including but not limited to an instruction cache **2552**, an instruction unit **2554**, an address mapping unit **2556**, a register file

2558, one or more general purpose graphics processing unit (GPGPU) cores **2562**, and one or more load/store units **2566**. GPGPU cores **2562** and load/store units **2566** are coupled with cache memory **2572** and shared memory **2570** via a memory and cache interconnect **2568**.

[0193] In at least one embodiment, instruction cache **2552** receives a stream of instructions to execute from pipeline manager **2532**. In at least one embodiment, instructions are cached in instruction cache **2552** and dispatched for execution by instruction unit **2554**. In at least one embodiment, instruction unit **2554** can dispatch instructions as thread groups (e.g., warps), with each thread of thread group assigned to a different execution unit within GPGPU core **2562**. 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 **2556** can be used to translate addresses in a unified address space into a distinct memory address that can be accessed by load/store units **2566**.

[0194] In at least one embodiment, register file **2558** provides a set of registers for functional units of graphics multiprocessor **2534**. In at least one embodiment, register file **2558** provides temporary storage for operands connected to data paths of functional units (e.g., GPGPU cores **2562**, load/store units **2566**) of graphics multiprocessor **2534**. In at least one embodiment, register file **2558** is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file **2558**. In one embodiment, register file **2558** is divided between different warps being executed by graphics multiprocessor **2534**.

[0195] In at least one embodiment, GPGPU cores **2562** can each include floating point units (FPUs) and/or integer arithmetic logic units (ALUs) that are used to execute instructions of graphics multiprocessor **2534**. GPGPU cores **2562** can be similar in architecture or can differ in architecture. In at least one embodiment, a first portion of GPGPU cores **2562** 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 for floating point arithmetic or enable variable precision floating point arithmetic. In at least one embodiment, graphics multiprocessor **2534** 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 GPGPU cores can also include fixed or special function logic.

[0196] In at least one embodiment, GPGPU cores **2562** include SIMD logic capable of performing a single instruction on multiple sets of data. In one embodiment GPGPU cores **2562** 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.

[0197] In at least one embodiment, memory and cache interconnect **2568** is an interconnect network that connects each functional unit of graphics multiprocessor **2534** to register file **2558** and to shared memory **2570**. In at least one embodiment, memory and cache interconnect **2568** is a crossbar interconnect that allows load/store unit **2566** to implement load and store operations between shared memory **2570** and register file **2558**. In at least one embodiment, register file **2558** can operate at a same frequency as GPGPU cores **2562**, thus data transfer between GPGPU cores **2562** and register file **2558** is very low latency. In at least one embodiment, shared memory **2570** can be used to enable communication between threads that execute on functional units within graphics multiprocessor **2534**. In at least one embodiment, cache memory **2572** can be used as a data cache for example, to cache texture data communicated between functional units and texture unit **2536**. In at least one embodiment, shared memory **2570** can also be used as a program managed cache. In at least one embodiment, threads executing on GPGPU cores **2562** can programmatically store data within shared memory in addition to automatically cached data that is stored within cache memory **2572**.

[0198] 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, GPU may be communicatively coupled to host processor/cores over a bus or other interconnect (e.g., a high speed interconnect such as PCIe or NVLink). In at least one embodiment, a GPU may be integrated on same package or chip as cores and communicatively coupled to cores over an internal processor bus/interconnect, as may be internal to a package or chip. In at least one embodiment, regardless of manner in which GPU is connected, processor cores may allocate work to a GPU in form of sequences of commands/instructions contained in a work descriptor. In at least one embodiment, a GPU then uses dedicated circuitry/logic for efficiently processing these commands/instructions.

[0199] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, inference and/or training logic **1015** 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.

[0200] FIG. 26 is a block diagram illustrating micro-architecture for a processor **2600** that may include logic circuits to perform instructions, in at least one embodiment. In at least one embodiment, processor **2600** may perform instructions, including x86 instructions, ARM instructions, specialized instructions for application-specific integrated circuits (ASICs), etc. In at least one embodiment, processor **2610** 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, processors **2610** may perform instructions to accelerate machine learning or deep learning algorithms, training, or inferencing.

[0201] In at least one embodiment, processor **2600** includes an in-order front end (“front end”) **2601** to fetch instructions to be executed and prepare instructions to be used later in processor pipeline. In at least one embodiment, front end **2601** may include several units. In at least one embodiment, an instruction prefetcher **2626** fetches instructions from memory and feeds instructions to an instruction decoder **2628** which in turn decodes or interprets instructions. For example, in at least one embodiment, instruction decoder **2628** decodes a received instruction into one or more operations called “micro-instructions” or “micro-operations” (also called “micro ops” or “uops”) that machine may execute. In at least one embodiment, instruction decoder **2628** parses 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 **2630** may assemble decoded uops into program ordered sequences or traces in a uop queue **2634** for execution. In at least one embodiment, when trace cache **2630** encounters a complex instruction, a microcode ROM **2632** provides uops needed to complete operation.

[0202] 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 **2628** may access microcode ROM **2632** to perform instruction. In at least one embodiment, an instruction may be decoded into a small number of micro-ops for processing at instruction decoder **2628**. In at least one embodiment, an instruction may be stored within microcode ROM **2632** should a number of micro-ops be needed to accomplish operation. In at least one embodiment, trace cache **2630** 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 **2632** in accordance with at least one embodiment. In at least one embodiment, after microcode ROM **2632** finishes sequencing micro-ops for an instruction, front end **2601** of a machine may resume fetching micro-ops from trace cache **2630**.

[0203] In at least one embodiment, out-of-order execution engine (“out of order engine”) **2603** 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 pipeline and get scheduled for execution. Out-of-order execution engine **2603** includes, without limitation, an allocator/register renamer **2640**, a memory uop queue **2642**, an integer/floating point uop queue **2644**, a memory scheduler **2646**, a fast scheduler **2602**, a slow/general floating point scheduler (“slow/general FP scheduler”) **2604**, and a simple floating point scheduler (“simple FP scheduler”) **2606**. In at least one embodiment, fast schedule **2602**,

slow/general floating point scheduler **2604**, and simple floating point scheduler **2606** are also collectively referred to herein as “uop schedulers **2602, 2604, 2606**.” allocator/register renamer **2640** allocates machine buffers and resources that each uop needs in order to execute. In at least one embodiment, allocator/register renamer **2640** renames logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer **2640** also allocates an entry for each uop in one of two uop queues, memory uop queue **2642** for memory operations and integer/floating point uop queue **2644** for non-memory operations, in front of memory scheduler **2646** and uop schedulers **2602, 2604, 2606**. In at least one embodiment, uop schedulers **2602, 2604, 2606**, 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 **2602** of at least one embodiment may schedule on each half of main clock cycle while slow/general floating point scheduler **2604** and simple floating point scheduler **2606** may schedule once per main processor clock cycle. In at least one embodiment, uop schedulers **2602, 2604, 2606** arbitrate for dispatch ports to schedule uops for execution.

[0204] In at least one embodiment, execution block b **11** includes, without limitation, an integer register file/bypass network **2608**, a floating point register file/bypass network (“FP register file/bypass network”) **2610**, address generation units (“AGUs”) **2612** and **2614**, fast Arithmetic Logic Units (ALUs) (“fast ALUs”) **2616** and **2618**, a slow Arithmetic Logic Unit (“slow ALU”) **2620**, a floating point ALU (“FP”) **2622**, and a floating point move unit (“FP move”) **2624**. In at least one embodiment, integer register file/bypass network **2608** and floating point register file/bypass network **2610** are also referred to herein as “register files **2608, 2610**.” In at least one embodiment, AGUs **2612** and **2614**, fast ALUs **2616** and **2618**, slow ALU **2620**, floating point ALU **2622**, and floating point move unit **2624** are also referred to herein as “execution units **2612, 2614, 2616, 2618, 2620, 2622**, and **2624**.” In at least one embodiment, execution block b11 may include, without limitation, any number (including zero) and type of register files, bypass networks, address generation units, and execution units, in any combination.

[0205] In at least one embodiment, register files **2608, 2610** may be arranged between uop schedulers **2602, 2604, 2606**, and execution units **2612, 2614, 2616, 2618, 2620, 2622**, and **2624**. In at least one embodiment, integer register file/bypass network **2608** performs integer operations. In at least one embodiment, floating point register file/bypass network **2610** performs floating point operations. In at least one embodiment, each of register files **2608, 2610** may include, without limitation, a bypass network that may bypass or forward just completed results that have not yet been written into register file to new dependent uops. In at least one embodiment, register files **2608, 2610** may communicate data with each other. In at least one embodiment, integer register file/bypass network **2608** may include, without limitation, two separate register files, one register file for low-order thirty-two bits of data and a second register file for high order thirty-two bits of data. In at least one embodiment, floating point register file/bypass network **2610** may include, without limitation, 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

[0206] In at least one embodiment, execution units **2612**, **2614**, **2616**, **2618**, **2620**, **2622**, **2624** may execute instructions. In at least one embodiment, register files **2608**, **2610** store integer and floating point data operand values that micro-instructions need to execute. In at least one embodiment, processor **2600** may include, without limitation, any number and combination of execution units **2612**, **2614**, **2616**, **2618**, **2620**, **2622**, **2624**. In at least one embodiment, floating point ALU **2622** and floating point move unit **2624**, 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 **2622** 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 **2616**, **2618**. In at least one embodiment, fast ALUs **2616**, **2618** 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 **2620** as slow ALU **2620** 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 **2612**, **2614**. In at least one embodiment, fast ALU **2616**, fast ALU **2618**, and slow ALU **2620** may perform integer operations on 64-bit data operands. In at least one embodiment, fast ALU **2616**, fast ALU **2618**, and slow ALU **2620** 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 **2622** and floating point move unit **2624** may be implemented to support a range of operands having bits of various widths. In at least one embodiment, floating point ALU **2622** and floating point move unit **2624** may operate on 128-bit wide packed data operands in conjunction with SIMD and multimedia instructions.

[0207] In at least one embodiment, uop schedulers **2602**, **2604**, **2606**, dispatch dependent operations before parent load has finished executing. In at least one embodiment, as uops may be speculatively scheduled and executed in processor **2600**, processor **2600** may also include logic to handle memory misses. In at least one embodiment, if a data load misses in data cache, there may be dependent operations in flight in pipeline that have left 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 replay mechanism of at least one embodiment of a processor may also be designed to catch instruction sequences for text string comparison operations.

[0208] In at least one embodiment, term “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 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.

[0209] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment portions or all of inference and/or training logic **1015** may be incorporated into EXE Block **2611** 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 EXE Block **2611**. Moreover, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of EXE Block **2611** to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0210] FIG. **27** illustrates a deep learning application processor **2700**, in at least one embodiment. In at least one embodiment, deep learning application processor **2700** uses instructions that, if executed by deep learning application processor **2700**, cause deep learning application processor **2700** to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, deep learning application processor **2700** is an application-specific integrated circuit (ASIC). In at least one embodiment, application processor **2700** 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 **2700** includes, without limitation, processing clusters **2710** (1)-**2710**(12), Inter-Chip Links (“ICLs”) **2720**(1)-**2720**(12), Inter-Chip Controllers (“ICCs”) **2730**(1)-**2730**(2), high bandwidth memory second generation (“HBM2”) **2740**(1)-**2740**(4), memory controllers (“Mem Ctrlrs”) **2742**(1)-**2742**(4), high bandwidth memory physical layer (“HBM PHY”) **2744**(1)-**2744**(4), a management-controller central processing unit (“management-controller CPU”) **2750**, a Serial Peripheral Interface, Inter-Integrated Circuit, and General Purpose Input/Output block (“SPI, I2C, GPIO”) **2760**, a peripheral component interconnect express controller and direct memory access block (“PCIe Controller and DMA”) **2770**, and a sixteen-lane peripheral component interconnect express port (“PCI Express \times 16”) **2780**.

[0211] In at least one embodiment, processing clusters **2710** 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 **2710** may include, without limitation, any number and type of processors. In at least one embodiment, deep learning application processor **2700** may include any number and type of processing clusters **2700**. In at least one embodiment, Inter-Chip Links **2720** are bi-directional. In at least one embodiment, Inter-Chip Links **2720** and Inter-Chip Controllers **2730** enable multiple deep learning application

processors **2700** 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 **2700** may include any number (including zero) and type of ICLs **2720** and ICCs **2730**.

[0212] In at least one embodiment, HBM2s **2740** provide a total of 32 Gigabytes (GB) of memory. HBM2 **2740(i)** is associated with both memory controller **2742(i)** and HBM PHY **2744(i)**. In at least one embodiment, any number of HBM2s **2740** 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 **2742** and HBM PHYs **2744**. In at least one embodiment, SPI, I2C, GPIO **2760**, PCIe Controller and DMA **2770**, and/or PCIe **2780** may be replaced with any number and type of blocks that enable any number and type of communication standards in any technically feasible fashion.

[0213] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. 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 **2700**. In at least one embodiment, deep learning application processor **2700** 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 **2700**. In at least one embodiment, processor **2700** may be used to perform one or more neural network use cases described herein.

[0214] FIG. 28 is a block diagram of a neuromorphic processor **2800**, in at least one embodiment. In at least one embodiment, neuromorphic processor **2800** may receive one or more inputs from sources external to neuromorphic processor **2800**. In at least one embodiment, these inputs may be transmitted to one or more neurons **2802** within neuromorphic processor **2800**. In at least one embodiment, neurons **2802** 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 **2800** may include, without limitation, thousands or millions of instances of neurons **2802**, but any suitable number of neurons **2802** may be used. In at least one embodiment, each instance of neuron **2802** may include a neuron input **2804** and a neuron output **2806**. In at least one embodiment, neurons **2802** may generate outputs that may be transmitted to inputs of other instances of neurons **2802**. For example, in at least one embodiment, neuron inputs **2804** and neuron outputs **2806** may be interconnected via synapses **2808**.

[0215] In at least one embodiment, neurons **2802** and synapses **2808** may be interconnected such that neuromorphic processor **2800** operates to process or analyze information received by neuromorphic processor **2800**. In at least one embodiment, neurons **2802** may transmit an output pulse (or “fire” or “spike”) when inputs received through neuron input **2804** exceed a threshold. In at least one embodiment, neurons **2802** may sum or integrate signals received at neuron inputs **2804**. For example, in at least one embodiment, neurons **2802** may be implemented as leaky integrate-and-fire neurons, wherein if a sum (referred to as

a “membrane potential”) exceeds a threshold value, neuron **2802** 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 **2804** 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 **2804** rapidly enough to exceed a threshold value, such as before a membrane potential decays too low to fire. In at least one embodiment, neurons **2802** 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 **2802** may include, without limitation, comparator circuits or logic that generate an output spike at neuron output **2806** when result of applying a transfer function to neuron input **2804** exceeds a threshold. In at least one embodiment, once neuron **2802** 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 **2802** may resume normal operation after a suitable period of time (or refractory period).

[0216] In at least one embodiment, neurons **2802** may be interconnected through synapses **2808**. In at least one embodiment, synapses **2808** may operate to transmit signals from an output of a first neuron **2802** to an input of a second neuron **2802**. In at least one embodiment, neurons **2802** may transmit information over more than one instance of synapse **2808**. In at least one embodiment, one or more instances of neuron output **2806** may be connected, via an instance of synapse **2808**, to an instance of neuron input **2804** in same neuron **2802**. In at least one embodiment, an instance of neuron **2802** generating an output to be transmitted over an instance of synapse **2808** may be referred to as a “pre-synaptic neuron” with respect to that instance of synapse **2808**. In at least one embodiment, an instance of neuron **2802** receiving an input transmitted over an instance of synapse **2808** may be referred to as a “post-synaptic neuron” with respect to that instance of synapse **2808**. Because an instance of neuron **2802** may receive inputs from one or more instances of synapse **2808**, and may also transmit outputs over one or more instances of synapse **2808**, a single instance of neuron **2802** may therefore be both a “pre-synaptic neuron” and “post-synaptic neuron,” with respect to various instances of synapses **2808**, in at least one embodiment.

[0217] In at least one embodiment, neurons **2802** may be organized into one or more layers. Each instance of neuron **2802** may have one neuron output **2806** that may fan out through one or more synapses **2808** to one or more neuron inputs **2804**. In at least one embodiment, neuron outputs **2806** of neurons **2802** in a first layer **2810** may be connected to neuron inputs **2804** of neurons **2802** in a second layer **2812**. In at least one embodiment, layer **2810** may be referred to as a “feed-forward layer.” In at least one embodiment, each instance of neuron **2802** in an instance of first layer **2810** may fan out to each instance of neuron **2802** in second layer **2812**. In at least one embodiment, first layer **2810** may be referred to as a “fully connected feed-forward layer.” In at least one embodiment, each instance of neuron

2802 in an instance of second layer **2812** may fan out to fewer than all instances of neuron **2802** in a third layer **2814**. In at least one embodiment, second layer **2812** may be referred to as a “sparsely connected feed-forward layer.” In at least one embodiment, neurons **2802** in second layer **2812** may fan out to neurons **2802** in multiple other layers, including to neurons **2802** in (same) second layer **2812**. In at least one embodiment, second layer **2812** may be referred to as a “recurrent layer.” Neuromorphic processor **2800** 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.

[0218] In at least one embodiment, neuromorphic processor **2800** may include, without limitation, a reconfigurable interconnect architecture or dedicated hard wired interconnects to connect synapse **2808** to neurons **2802**. In at least one embodiment, neuromorphic processor **2800** may include, without limitation, circuitry or logic that allows synapses to be allocated to different neurons **2802** as needed based on neural network topology and neuron fan-in/out. For example, in at least one embodiment, synapses **2808** may be connected to neurons **2802** 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.

[0219] FIG. 29 is a block diagram of a graphics processor **2900**, 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 **2900** communicates via a memory mapped I/O interface to registers on graphics processor **2900** and with commands placed into memory. In at least one embodiment, graphics processor **2900** includes a memory interface **2914** to access memory. In at least one embodiment, memory interface **2914** is an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0220] In at least one embodiment, graphics processor **2900** also includes a display controller **2902** to drive display output data to a display device **2920**. In at least one embodiment, display controller **2902** includes hardware for one or more overlay planes for display device **2920** and composition of multiple layers of video or user interface elements. In at least one embodiment, display device **2920** can be an internal or external display device. In at least one embodiment, display device **2920** 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 **2900** includes a video codec engine **2906** 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 a 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.

[0221] In at least one embodiment, graphics processor **2900** includes a block image transfer (BLIT) engine **2904** to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in at least one embodiment, 2D graphics operations are per-

formed using one or more components of graphics processing engine (GPE) **2910**. In at least one embodiment, GPE **2910** is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0222] In at least one embodiment, GPE **2910** includes a 3D pipeline **2912** 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.). 3D pipeline **2912** includes programmable and fixed function elements that perform various tasks and/or spawn execution threads to a 3D/Media subsystem **2915**. While 3D pipeline **2912** can be used to perform media operations, in at least one embodiment, GPE **2910** also includes a media pipeline **2916** that is used to perform media operations, such as video post-processing and image enhancement.

[0223] In at least one embodiment, media pipeline **2916** 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 **2906**. In at least one embodiment, media pipeline **2916** additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system **2915**. 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 **2915**.

[0224] In at least one embodiment, 3D/Media subsystem **2915** includes logic for executing threads spawned by 3D pipeline **2912** and media pipeline **2916**. In at least one embodiment, 3D pipeline **2912** and media pipeline **2916** send thread execution requests to 3D/Media subsystem **2915**, 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 **2915** includes one or more internal caches for thread instructions and data. In at least one embodiment, subsystem **2915** also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

[0225] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment portions or all of inference and/or training logic **1015** may be incorporated into graphics processor **2900**. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in 3D pipeline **2912**. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. 10 or 11. 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 **2900** to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0226] FIG. 30 is a block diagram of hardware logic of a graphics processor core **3000**, in at least one embodiment described herein. In at least one embodiment, graphics

processor core **3000** is included within a graphics core array. In at least one embodiment, graphics processor core **3000**, sometimes referred to as a core slice, can be one or multiple graphics cores within a modular graphics processor. In at least one embodiment, graphics processor core **3000** is exemplary of one graphics core slice, and a graphics processor as described herein may include multiple graphics core slices based on target power and performance envelopes. In at least one embodiment, each graphics core **3000** can include a fixed function block **3030** coupled with multiple sub-cores **3001A-3001F**, also referred to as sub-slices, that include modular blocks of general-purpose and fixed function logic.

[0227] In at least one embodiment, fixed function block **3030** includes a geometry/fixed function pipeline **3036** that can be shared by all sub-cores in graphics processor **3000**, for example, in lower performance and/or lower power graphics processor implementations. In at least one embodiment, geometry/fixed function pipeline **3036** includes a 3D fixed function pipeline, a video front-end unit, a thread spawner and thread dispatcher, and a unified return buffer manager, which manages unified return buffers.

[0228] In at least one embodiment fixed function block **3030** also includes a graphics SoC interface **3037**, a graphics microcontroller **3038**, and a media pipeline **3039**. Graphics SoC interface **3037** provides an interface between graphics core **3000** and other processor cores within a system on a chip integrated circuit. In at least one embodiment, graphics microcontroller **3038** is a programmable sub-processor that is configurable to manage various functions of graphics processor **3000**, including thread dispatch, scheduling, and pre-emption. In at least one embodiment, media pipeline **3039** includes logic to facilitate decoding, encoding, pre-processing, and/or post-processing of multimedia data, including image and video data. In at least one embodiment, media pipeline **3039** implement media operations via requests to compute or sampling logic within sub-cores **3001-3001F**.

[0229] In at least one embodiment, SoC interface **3037** enables graphics core **3000** to communicate with general-purpose application processor cores (e.g., CPUs) and/or other components within an SoC, including memory hierarchy elements such as a shared last level cache memory, system RAM, and/or embedded on-chip or on-package DRAM. In at least one embodiment, SoC interface **3037** can also enable communication with fixed function devices within an SoC, such as camera imaging pipelines, and enables use of and/or implements global memory atomics that may be shared between graphics core **3000** and CPUs within an SoC. In at least one embodiment, SoC interface **3037** can also implement power management controls for graphics core **3000** and enable an interface between a clock domain of graphic core **3000** and other clock domains within an SoC. In at least one embodiment, SoC interface **3037** enables receipt of command buffers from a command streamer and global thread dispatcher that are configured to provide commands and instructions to each of one or more graphics cores within a graphics processor. In at least one embodiment, commands and instructions can be dispatched to media pipeline **3039**, when media operations are to be performed, or a geometry and fixed function pipeline (e.g., geometry and fixed function pipeline **3036**, geometry and fixed function pipeline **3014**) when graphics processing operations are to be performed.

[0230] In at least one embodiment, graphics microcontroller **3038** can be configured to perform various scheduling and management tasks for graphics core **3000**. In at least one embodiment, graphics microcontroller **3038** can perform graphics and/or compute workload scheduling on various graphics parallel engines within execution unit (EU) arrays **3002A-3002F**, **3004A-3004F** within sub-cores **3001A-3001F**. In at least one embodiment, host software executing on a CPU core of an SoC including graphics core **3000** can submit workloads one of multiple graphic processor doorbells, 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, preempting existing workloads running on an engine, monitoring progress of a workload, and notifying host software when a workload is complete. In at least one embodiment, graphics microcontroller **3038** can also facilitate low-power or idle states for graphics core **3000**, providing graphics core **3000** with an ability to save and restore registers within graphics core **3000** across low-power state transitions independently from an operating system and/or graphics driver software on a system.

[0231] In at least one embodiment, graphics core **3000** may have greater than or fewer than illustrated sub-cores **3001A-3001F**, up to N modular sub-cores. For each set of N sub-cores, in at least one embodiment, graphics core **3000** can also include shared function logic **3010**, shared and/or cache memory **3012**, a geometry/fixed function pipeline **3014**, as well as additional fixed function logic **3016** to accelerate various graphics and compute processing operations. In at least one embodiment, shared function logic **3010** can include logic units (e.g., sampler, math, and/or inter-thread communication logic) that can be shared by each N sub-cores within graphics core **3000**. Shared and/or cache memory **3012** can be a last-level cache for N sub-cores **3001A-3001F** within graphics core **3000** and can also serve as shared memory that is accessible by multiple sub-cores. In at least one embodiment, geometry/fixed function pipeline **3014** can be included instead of geometry/fixed function pipeline **3036** within fixed function block **3030** and can include same or similar logic units.

[0232] In at least one embodiment, graphics core **3000** includes additional fixed function logic **3016** that can include various fixed function acceleration logic for use by graphics core **3000**. In at least one embodiment, additional fixed function logic **3016** includes an additional geometry pipeline for use in position-only shading. In position-only shading, at least two geometry pipelines exist, whereas in a full geometry pipeline within geometry/fixed function pipeline **3016**, **3036**, and a cull pipeline, which is an additional geometry pipeline which may be included within additional fixed function logic **3016**. In at least one embodiment, cull pipeline is a trimmed down version of a full geometry pipeline. In at least one embodiment, a full pipeline and a cull pipeline can execute different instances of an application, each instance having a separate context. In at least one embodiment, position only shading can hide long cull runs of discarded triangles, enabling shading to be completed earlier in some instances. For example, in at least one embodiment, cull pipeline logic within additional fixed function logic **3016** can execute position shaders in parallel with a main application and generates critical results faster than a full pipeline, as cull pipeline fetches and shades

position attribute of vertices, without performing rasterization and rendering of pixels to a frame buffer. In at least one embodiment, 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, 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.

[0233] In at least one embodiment, additional fixed function logic **3016** can also include machine-learning acceleration logic, such as fixed function matrix multiplication logic, for implementations including optimizations for machine learning training or inferencing.

[0234] In at least one embodiment, within each graphics sub-core **3001A-3001F** includes a set of execution resources that may be used to perform graphics, media, and compute operations in response to requests by graphics pipeline, media pipeline, or shader programs. In at least one embodiment, graphics sub-cores **3001A-3001F** include multiple EU arrays **3002A-3002F**, **3004A-3004F**, thread dispatch and inter-thread communication (TD/IC) logic **3003A-3003F**, a 3D (e.g., texture) sampler **3005A-3005F**, a media sampler **3006A-3006F**, a shader processor **3007A-3007F**, and shared local memory (SLM) **3008A-3008F**. EU arrays **3002A-3002F**, **3004A-3004F** each include multiple execution units, which are general-purpose graphics processing units capable of performing floating-point and integer/fixed-point logic operations in service of a graphics, media, or compute operation, including graphics, media, or compute shader programs. In at least one embodiment, TD/IC logic **3003A-3003F** performs local thread dispatch and thread control operations for execution units within a sub-core and facilitate communication between threads executing on execution units of a sub-core. In at least one embodiment, 3D sampler **3005A-3005F** can read texture or other 3D graphics related data into memory. In at least one embodiment, 3D sampler 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 sampler **3006A-3006F** can perform similar read operations based on a type and format associated with media data. In at least one embodiment, each graphics sub-core **3001A-3001F** can alternately include a unified 3D and media sampler. In at least one embodiment, threads executing on execution units within each of sub-cores **3001A-3001F** can make use of shared local memory **3008A-3008F** within each sub-core, to enable threads executing within a thread group to execute using a pool of on-chip memory.

[0235] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, portions or all of inference and/or training logic **1015** may be incorporated into graphics processor **3010**. 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 **3010**, graphics microcontroller **3038**, geometry & fixed function pipeline **3014** and **3036**, or other logic in FIG. 29B. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. 10 or 11. 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.

[0236] FIGS. 31A-31B illustrate thread execution logic **3100** including an array of processing elements of a graphics processor core in at least one embodiment. FIG. 31A illustrates at least one embodiment, in which thread execution logic **3100** is used. FIG. 31B illustrates exemplary internal details of an execution unit, in at least one embodiment.

[0237] As illustrated in FIG. 31A, in at least one embodiment, thread execution logic **3100** includes a shader processor **3102**, a thread dispatcher **3104**, instruction cache **3106**, a scalable execution unit array including a plurality of execution units **3108A-3108N**, a sampler **3110**, a data cache **3112**, and a data port **3114**. In at least one embodiment a scalable execution unit array can dynamically scale by enabling or disabling one or more execution units (e.g., any of execution unit **3108A**, **3108B**, **3108C**, **3108D**, through **3108N-1** and **3108N**) based on computational requirements of a workload, for example. In at least one embodiment, scalable execution units are interconnected via an interconnect fabric that links to each execution unit. In at least one embodiment, thread execution logic **3100** includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache **3106**, data port **3114**, sampler **3110**, and execution units **3108A-3108N**. In at least one embodiment, each execution unit (e.g., **3108A**) 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 **3108A-3108N** is scalable to include any number individual execution units.

[0238] In at least one embodiment, execution units **3108A-3108N** are primarily used to execute shader programs. In at least one embodiment, shader processor **3102** can process various shader programs and dispatch execution threads associated with shader programs via a thread dispatcher **3104**. In at least one embodiment, thread dispatcher **3104** includes logic to arbitrate thread initiation requests from graphics and media pipelines and instantiate requested threads on one or more execution units in execution units **3108A-3108N**. For example, in at least one embodiment, a geometry pipeline can dispatch vertex, tessellation, or geometry shaders to thread execution logic for processing. In at least one embodiment, thread dispatcher **3104** can also process runtime thread spawning requests from executing shader programs.

[0239] In at least one embodiment, execution units **3108A-3108N** 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 minimal translation. In at least one embodiment, execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, 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 **3108A-3108N**, 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 or more shared functions, dependency logic within execution units **3108A-3108N** causes a waiting thread to sleep until requested data has been returned. In at least one embodiment, hardware resources may be devoted to processing other threads while a specific, waiting thread is sleeping. 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.

[0240] In at least one embodiment, each execution unit in execution units **3108A-3108N** operates on arrays of data elements. In at least one embodiment, a number of data elements is “execution size,” or a 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 **3108A-3108N** support integer and floating-point data types.

[0241] 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 an execution unit will process various elements based on a data size of those 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.

[0242] In at least one embodiment, one or more execution units can be combined into a fused execution unit **3109A-3109N** having thread control logic (**3107A-3107N**) that is used for fused EUs. In at least one embodiment, multiple EUs can be fused into an EU group. In at least one embodiment, each EU in fused EU group can be configured to execute a separate SIMD hardware thread. A number of EUs in a fused EU group can vary according to various embodiments. In at least one embodiment, various SIMD widths can be performed per-EU, including but not limited to SIMD8, SIMD16, and SIMD32. In at least one embodiment, each fused graphics execution unit **3109A-3109N** includes at least two execution units. For example, in at least one embodiment, fused execution unit **3109A** includes a first EU **3108A**, second EU **3108B**, and thread control logic **3107A** that is shared with first EU **3108A** and second EU **3108B**. In at least one embodiment, thread control logic **3107A** controls threads executed on fused graphics execu-

tion unit **3109A**, allowing each EU within fused execution units **3109A-3109N** to execute using an instruction pointer register.

[0243] In at least one embodiment, one or more internal instruction caches (e.g., **3106**) are included in thread execution logic **3100** to cache thread instructions for execution units. In at least one embodiment, one or more data caches (e.g., **3112**) are included to cache thread data during thread execution. In at least one embodiment, a sampler **3110** is included to provide texture sampling for 3D operations and media sampling for media operations. In at least one embodiment, sampler **3110** includes specialized texture or media sampling functionality to process texture or media data during a sampling process before providing sampled data to an execution unit.

[0244] During execution, in at least one embodiment, graphics and media pipelines send thread initiation requests to thread execution logic **3100** via thread spawning and dispatch logic. In at least one embodiment, once a group of geometric objects has been processed and rasterized into pixel data, pixel processor logic (e.g., pixel shader logic, fragment shader logic, etc.) within shader processor **3102** is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In at least one embodiment, a pixel shader or fragment shader calculates values of various vertex attributes that are to be interpolated across a rasterized object. In at least one embodiment, pixel processor logic within shader processor **3102** then executes an application programming interface (API)-supplied pixel or fragment shader program. In at least one embodiment, to execute a shader program, shader processor **3102** dispatches threads to an execution unit (e.g., **3108A**) via thread dispatcher **3104**. In at least one embodiment, shader processor **3102** uses texture sampling logic in sampler **3110** to access texture data in texture maps stored in memory. In at least one embodiment, arithmetic operations on texture data and input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

[0245] In at least one embodiment, data port **3114** provides a memory access mechanism for thread execution logic **3100** to output processed data to memory for further processing on a graphics processor output pipeline. In at least one embodiment, data port **3114** includes or couples to one or more cache memories (e.g., data cache **3112**) to cache data for memory access via a data port.

[0246] As illustrated in FIG. 31B, in at least one embodiment, a graphics execution unit **3108** can include an instruction fetch unit **3137**, a general register file array (GRF) **3124**, an architectural register file array (ARF) **3126**, a thread arbiter **3122**, a send unit **3130**, a branch unit **3132**, a set of SIMD floating point units (FPUs) **3134**, and in at least one embodiment a set of dedicated integer SIMD ALUs **3135**. In at least one embodiment, GRF **3124** and ARF **3126** includes a set of general register files and architecture register files associated with each simultaneous hardware thread that may be active in graphics execution unit **3108**. In at least one embodiment, per thread architectural state is maintained in ARF **3126**, while data used during thread execution is stored in GRF **3124**. In at least one embodiment, execution state of each thread, including instruction pointers for each thread, can be held in thread-specific registers in ARF **3126**.

[0247] In at least one embodiment, graphics execution unit **3108** has an architecture that is a combination of Simultaneous Multi-Threading (SMT) and fine-grained Interleaved Multi-Threading (IMT). In at least one embodiment, architecture has a modular configuration that can be fine-tuned at design time based on a target number of simultaneous threads and number of registers per execution unit, where execution unit resources are divided across logic used to execute multiple simultaneous threads.

[0248] In at least one embodiment, graphics execution unit **3108** can co-issue multiple instructions, which may each be different instructions. In at least one embodiment, thread arbiter **3122** of graphics execution unit thread **3108** can dispatch instructions to one of send unit **3130**, branch unit **3142**, or SIMD FPU(s) **3134** for execution. In at least one embodiment, each execution thread can access 128 general-purpose registers within GRF **3124**, where each register can store 32 bytes, accessible as a SIMD 8-element vector of 32-bit data elements. In at least one embodiment, each execution unit thread has access to 4 Kbytes within GRF **3124**, although embodiments are not so limited, and greater or fewer register resources may be provided in other embodiments. In at least one embodiment, up to seven threads can execute simultaneously, although a number of threads per execution unit can also vary according to embodiments. In at least one embodiment, in which seven threads may access 4 Kbytes, GRF **3124** can store a total of 28 Kbytes. 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.

[0249] 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 send unit **3130**. In at least one embodiment, branch instructions are dispatched to a dedicated branch unit **3132** to facilitate SIMD divergence and eventual convergence.

[0250] In at least one embodiment graphics execution unit **3108** includes one or more SIMD floating point units (FPU(s)) **3134** to perform floating-point operations. In at least one embodiment, FPU(s) **3134** also support integer computation. In at least one embodiment FPU(s) **3134** can SIMD execute up to M number of 32-bit floating-point (or integer) operations, or SIMD execute up to 2M 16-bit integer or 16-bit floating-point operations. In at least one embodiment, at least one of FPU(s) provides extended math capability to support high-throughput transcendental math functions and double precision 64-bit floating-point. In at least one embodiment, a set of 8-bit integer SIMD ALUs **3135** are also present, and may be specifically optimized to perform operations associated with machine learning computations.

[0251] In at least one embodiment, arrays of multiple instances of graphics execution unit **3108** can be instantiated in a graphics sub-core grouping (e.g., a sub-slice). In at least one embodiment execution unit **3108** can execute instructions across a plurality of execution channels. In at least one embodiment, each thread executed on graphics execution unit **3108** is executed on a different channel.

[0252] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction

with FIGS. **10** and/or **11**. In at least one embodiment, portions or all of inference and/or training logic **1015** may be incorporated into execution logic **3100**. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. **10** or **11**. 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 execution logic **3100** to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

[0253] FIG. **32** illustrates a parallel processing unit (“PPU”) **3200**, in at least one embodiment. In at least one embodiment, PPU **3200** is configured with machine-readable code that, if executed by PPU **3200**, causes PPU **3200** to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, PPU **3200** is a multi-threaded processor that is implemented on one or more integrated circuit devices and that utilizes multithreading as a latency-hiding technique designed to process computer-readable instructions (also referred to as machine-readable instructions or simply instructions) on multiple threads in parallel. In at least one embodiment, a thread refers to a thread of execution and is an instantiation of a set of instructions configured to be executed by PPU **3200**. In at least one embodiment, PPU **3200** is a graphics processing unit (“GPU”) configured to implement a graphics rendering pipeline for processing three-dimensional (“3D”) graphics data in order to generate two-dimensional (“2D”) image data for display on a display device such as a liquid crystal display (“LCD”) device. In at least one embodiment, PPU **3200** is utilized to perform computations such as linear algebra operations and machine-learning operations. FIG. **32** illustrates an example parallel processor for illustrative purposes only and should be construed as a non-limiting example of processor architectures contemplated within scope of this disclosure and that any suitable processor may be employed to supplement and/or substitute for same.

[0254] In at least one embodiment, one or more PPUs **3200** are configured to accelerate High Performance Computing (“HPC”), data center, and machine learning applications. In at least one embodiment, PPU **3200** is configured to accelerate deep learning systems and applications including following non-limiting examples: autonomous vehicle platforms, deep learning, high-accuracy speech, image, text recognition systems, intelligent video analytics, molecular simulations, drug discovery, disease diagnosis, weather forecasting, big data analytics, astronomy, molecular dynamics simulation, financial modeling, robotics, factory automation, real-time language translation, online search optimizations, and personalized user recommendations, and more.

[0255] In at least one embodiment, PPU **3200** includes, without limitation, an Input/Output (“I/O”) unit **3206**, a front-end unit **3210**, a scheduler unit **3212**, a work distribution unit **3214**, a hub **3216**, a crossbar (“Xbar”) **3220**, one or more general processing clusters (“GPCs”) **3218**, and one or more partition units (“memory partition units”) **3222**. In at least one embodiment, PPU **3200** is connected to a host processor or other PPUs **3200** via one or more high-speed GPU interconnects (“GPU interconnects”) **3208**. In at least one embodiment, PPU **3200** is connected to a host processor or other peripheral devices via an interconnect **3202**. In at least one embodiment, PPU **3200** is connected to a local memory comprising one or more memory devices

(“memory”) **3204**. In at least one embodiment, memory devices **3204** include, without limitation, one or more dynamic random access memory (“DRAM”) devices. In at least one embodiment, one or more DRAM devices are configured and/or configurable as high-bandwidth memory (“HBM”) subsystems, with multiple DRAM dies stacked within each device.

[0256] In at least one embodiment, high-speed GPU interconnect **3208** may refer to a wire-based multi-lane communications link that is used by systems to scale and include one or more PPUs **3200** combined with one or more central processing units (“CPUs”), supports cache coherence between PPUs **3200** and CPUs, and CPU mastering. In at least one embodiment, data and/or commands are transmitted by high-speed GPU interconnect **3208** through hub **3216** to/from other units of PPU **3200** such as one or more copy engines, video encoders, video decoders, power management units, and other components which may not be explicitly illustrated in FIG. 32.

[0257] In at least one embodiment, I/O unit **3206** is configured to transmit and receive communications (e.g., commands, data) from a host processor (not illustrated in FIG. 32) over system bus **3202**. In at least one embodiment, I/O unit **3206** communicates with host processor directly via system bus **3202** or through one or more intermediate devices such as a memory bridge. In at least one embodiment, I/O unit **3206** may communicate with one or more other processors, such as one or more of PPUs **3200** via system bus **3202**. In at least one embodiment, I/O unit **3206** implements a Peripheral Component Interconnect Express (“PCIe”) interface for communications over a PCIe bus. In at least one embodiment, I/O unit **3206** implements interfaces for communicating with external devices.

[0258] In at least one embodiment, I/O unit **3206** decodes packets received via system bus **3202**. In at least one embodiment, at least some packets represent commands configured to cause PPU **3200** to perform various operations. In at least one embodiment, I/O unit **3206** transmits decoded commands to various other units of PPU **3200** as specified by commands. In at least one embodiment, commands are transmitted to front-end unit **3210** and/or transmitted to hub **3216** or other units of PPU **3200** such as one or more copy engines, a video encoder, a video decoder, a power management unit, etc. (not explicitly illustrated in FIG. 32). In at least one embodiment, I/O unit **3206** is configured to route communications between and among various logical units of PPU **3200**.

[0259] In at least one embodiment, a program executed by host processor encodes a command stream in a buffer that provides workloads to PPU **3200** for processing. In at least one embodiment, a workload comprises instructions and data to be processed by those instructions. In at least one embodiment, buffer is a region in a memory that is accessible (e.g., read/write) by both host processor and PPU **3200**. A host interface unit may be configured to access buffer in a system memory connected to system bus **3202** via memory requests transmitted over system bus **3202** by I/O unit **3206**. In at least one embodiment, host processor writes command stream to buffer and then transmits a pointer to start of command stream to PPU **3200** such that front-end unit **3210** receives pointers to one or more command streams and manages one or more command streams, reading commands from command streams and forwarding commands to various units of PPU **3200**.

[0260] In at least one embodiment, front-end unit **3210** is coupled to scheduler unit **3212** that configures various GPCs **3218** to process tasks determined by one or more command streams. In at least one embodiment, scheduler unit **3212** is configured to track state information related to various tasks managed by scheduler unit **3212** where state information may indicate which of GPCs **3218** a task is assigned to, whether task is active or inactive, a priority level associated with task, and so forth. In at least one embodiment, scheduler unit **3212** manages execution of a plurality of tasks on one or more of GPCs **3218**.

[0261] In at least one embodiment, scheduler unit **3212** is coupled to work distribution unit **3214** that is configured to dispatch tasks for execution on GPCs **3218**. In at least one embodiment, work distribution unit **3214** tracks a number of scheduled tasks received from scheduler unit **3212** and work distribution unit **3214** manages a pending task pool and an active task pool for each of GPCs **3218**. In at least one embodiment, pending task pool comprises a number of slots (e.g., 32 slots) that contain tasks assigned to be processed by a particular GPC **3218**; active task pool may comprise a number of slots (e.g., 4 slots) for tasks that are actively being processed by GPCs **3218** such that as one of GPCs **3218** completes execution of a task, that task is evicted from active task pool for GPC **3218** and one of other tasks from pending task pool is selected and scheduled for execution on GPC **3218**. In at least one embodiment, if an active task is idle on GPC **3218**, such as while waiting for a data dependency to be resolved, then active task is evicted from GPC **3218** and returned to pending task pool while another task in pending task pool is selected and scheduled for execution on GPC **3218**.

[0262] In at least one embodiment, work distribution unit **3214** communicates with one or more GPCs **3218** via XBar **3220**. In at least one embodiment, XBar **3220** is an interconnect network that couples many of units of PPU **3200** to other units of PPU **3200** and can be configured to couple work distribution unit **3214** to a particular GPC **3218**. In at least one embodiment, one or more other units of PPU **3200** may also be connected to XBar **3220** via hub **3216**.

[0263] In at least one embodiment, tasks are managed by scheduler unit **3212** and dispatched to one of GPCs **3218** by work distribution unit **3214**. GPC **3218** is configured to process task and generate results. In at least one embodiment, results may be consumed by other tasks within GPC **3218**, routed to a different GPC **3218** via XBar **3220**, or stored in memory **3204**. In at least one embodiment, results can be written to memory **3204** via partition units **3222**, which implement a memory interface for reading and writing data to/from memory **3204**. In at least one embodiment, results can be transmitted to another PPU **3200** or CPU via high-speed GPU interconnect **3208**. In at least one embodiment, PPU **3200** includes, without limitation, a number U of partition units **3222** that is equal to number of separate and distinct memory devices **3204** coupled to PPU **3200**. In at least one embodiment, partition unit **3222** will be described in more detail below in conjunction with FIG. 34.

[0264] 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 host processor to schedule operations for execution on PPU **3200**. In at least one embodiment, multiple compute applications are simultaneously executed by PPU **3200** and PPU **3200** provides isolation,

quality of service (“QoS”), and independent address spaces for multiple compute applications. In at least one embodiment, an application generates instructions (e.g., in form of API calls) that cause driver kernel to generate one or more tasks for execution by PPU **3200** and driver kernel outputs tasks to one or more streams being processed by PPU **3200**. In at least one embodiment, each task comprises one or more groups of related threads, which may be referred to as a warp. In at least one embodiment, a warp comprises a plurality of related threads (e.g., 32 threads) that can be executed in parallel. In at least one embodiment, cooperating threads can refer to a plurality of threads including instructions to perform task and that exchange data through shared memory. In at least one embodiment, threads and cooperating threads are described in more detail, in accordance with at least one embodiment, in conjunction with FIG. 34.

[0265] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. 10 and/or 11. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to PPU **3200**. In at least one embodiment, deep learning application processor **3200** is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by PPU **3200**. In at least one embodiment, PPU **3200** may be used to perform one or more neural network use cases described herein.

[0266] FIG. 33 illustrates a general processing cluster (“GPC”) **3300**, in at least one embodiment. In at least one embodiment, GPC **3300** is GPC **3218** of FIG. 32. In at least one embodiment, each GPC **3300** includes, without limitation, a number of hardware units for processing tasks and each GPC **3300** includes, without limitation, a pipeline manager **3302**, a pre-raster operations unit (“PROP”) **3304**, a raster engine **3308**, a work distribution crossbar (“WDX”) **3316**, a memory management unit (“MMU”) **3318**, one or more Data Processing Clusters (“DPCs”) **3306**, and any suitable combination of parts.

[0267] In at least one embodiment, operation of GPC **3300** is controlled by pipeline manager **3302**. In at least one embodiment, pipeline manager **3302** manages configuration of one or more DPCs **3306** for processing tasks allocated to GPC **3300**. In at least one embodiment, pipeline manager **3302** configures at least one of one or more DPCs **3306** to implement at least a portion of a graphics rendering pipeline. In at least one embodiment, DPC **3306** is configured to execute a vertex shader program on a programmable streaming multi-processor (“SM”) **3314**. In at least one embodiment, pipeline manager **3302** is configured to route packets received from a work distribution unit to appropriate logical units within GPC **3300**, in at least one embodiment, and some packets may be routed to fixed function hardware units in PROP **3304** and/or raster engine **3308** while other packets may be routed to DPCs **3306** for processing by a primitive engine **3312** or SM **3314**. In at least one embodiment, pipeline manager **3302** configures at least one of DPCs **3306** to implement a neural network model and/or a computing pipeline.

[0268] In at least one embodiment, PROP unit **3304** is configured, in at least one embodiment, to route data generated by raster engine **3308** and DPCs **3306** to a Raster

Operations (“ROP”) unit in partition unit **3222**, described in more detail above in conjunction with FIG. 32. In at least one embodiment, PROP unit **3304** is configured to perform optimizations for color blending, organize pixel data, perform address translations, and more. In at least one embodiment, raster engine **3308** includes, without limitation, a number of fixed function hardware units configured to perform various raster operations, in at least one embodiment, and raster engine **3308** includes, without limitation, a setup engine, a coarse raster engine, a culling engine, a clipping engine, a fine raster engine, a tile coalescing engine, and any suitable combination thereof. In at least one embodiment, setup engine receives transformed vertices and generates plane equations associated with geometric primitive defined by vertices; plane equations are transmitted to coarse raster engine to generate coverage information (e.g., an x, y coverage mask for a tile) for primitive; output of coarse raster engine is transmitted to culling engine where fragments associated with 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 fine raster engine to generate attributes for pixel fragments based on plane equations generated by setup engine. In at least one embodiment, output of raster engine **3308** comprises fragments to be processed by any suitable entity such as by a fragment shader implemented within DPC **3306**.

[0269] In at least one embodiment, each DPC **3306** included in GPC **3300** comprise, without limitation, an M-Pipe Controller (“MPC”) **3310**; primitive engine **3312**; one or more SMs **3314**; and any suitable combination thereof. In at least one embodiment, MPC **3310** controls operation of DPC **3306**, routing packets received from pipeline manager **3302** to appropriate units in DPC **3306**. In at least one embodiment, packets associated with a vertex are routed to primitive engine **3312**, which is configured to fetch vertex attributes associated with vertex from memory; in contrast, packets associated with a shader program may be transmitted to SM **3314**.

[0270] In at least one embodiment, SM **3314** comprises, without limitation, a programmable streaming processor that is configured to process tasks represented by a number of threads. In at least one embodiment, SM **3314** is multi-threaded and configured to execute a plurality of threads (e.g., 32 threads) from a particular group of threads concurrently and implements a Single-Instruction, Multiple-Data (“SIMD”) architecture where each thread in a group of threads (e.g., a warp) is configured to process a different set of data based on same set of instructions. In at least one embodiment, all threads in group of threads execute same instructions. In at least one embodiment, SM **3314** implements a Single-Instruction, Multiple Thread (“SIMT”) architecture wherein each thread in a group of threads is configured to process a different set of data based on same set of instructions, but where individual threads in group of threads are allowed to diverge during execution. In at least one embodiment, a program counter, call stack, and execution state is maintained for each warp, enabling concurrency between warps and serial execution within warps when threads within warp diverge. In another embodiment, a program counter, call stack, and execution state are 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 same instructions may be converged and executed in parallel for better efficiency. At least one embodiment of SM **3314** is described in more detail below.

[0271] In at least one embodiment, MMU **3318** provides an interface between GPC **3300** and memory partition unit (e.g., partition unit **3222** of FIG. **32**) and MMU **3318** provides translation of virtual addresses into physical addresses, memory protection, and arbitration of memory requests. In at least one embodiment, MMU **3318** provides one or more translation lookaside buffers ("TLBs") for performing translation of virtual addresses into physical addresses in memory.

[0272] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to GPC **3300**. In at least one embodiment, GPC **3300** is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by GPC **3300**. In at least one embodiment, GPC **3300** may be used to perform one or more neural network use cases described herein.

[0273] FIG. **34** illustrates a memory partition unit **3400** of a parallel processing unit ("PPU"), in accordance with at least one embodiment. In at least one embodiment, memory partition unit **3400** includes, without limitation, a Raster Operations ("ROP") unit **3402**; a level two ("L2") cache **3404**; a memory interface **3406**; and any suitable combination thereof. Memory interface **3406** is coupled to memory. Memory interface **3406** 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 **3406**, one memory interface **3406** per pair of partition units **3400**, where each pair of partition units **3400** is connected to a corresponding memory device. For example, in at least one embodiment, PPU may be connected to up to Y memory devices, such as high bandwidth memory stacks or graphics double-data-rate, version 5, synchronous dynamic random access memory ("GDDR5 SDRAM").

[0274] In at least one embodiment, memory interface **3406** implements a high bandwidth memory second generation ("HBM2") memory interface and Y equals half U. In at least one embodiment, HBM2 memory stacks are located on same physical package as PPU, providing substantial power and area savings compared with GDDR5 SDRAM systems. In at least one embodiment, each HBM2 stack includes, without limitation, four memory dies and Y equals 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, memory supports Single-Error Correcting Double-Error Detecting ("SECDED") Error Correction Code ("ECC") to protect data. ECC provides higher reliability for compute applications that are sensitive to data corruption.

[0275] In at least one embodiment, PPU implements a multi-level memory hierarchy. In at least one embodiment, memory partition unit **3400** supports a unified memory to provide a single unified virtual address space for central

processing unit ("CPU") and PPU memory, enabling data sharing between virtual memory systems. In at least one embodiment frequency of accesses by a PPU to memory located on other processors is traced to ensure that memory pages are moved to physical memory of PPU that is accessing pages more frequently. In at least one embodiment, high-speed GPU interconnect **3208** supports address translation services allowing PPU to directly access a CPU's page tables and providing full access to CPU memory by PPU.

[0276] In at least one embodiment, copy engines transfer data between multiple PPUs or between PPUs and CPUs. In at least one embodiment, copy engines can generate page faults for addresses that are not mapped into page tables and memory partition unit **3400** then services page faults, mapping addresses into page table, after which copy engine performs transfer. In at least one embodiment, memory is pinned (or 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 copy process is transparent.

[0277] Data from memory **3204** of FIG. **32** or other system memory is fetched by memory partition unit **3400** and stored in L2 cache **3404**, which is located on-chip and is shared between various GPCs, in accordance with at least one embodiment. Each memory partition unit **3400**, in at least one embodiment, includes, without limitation, at least a portion of L2 cache associated with a corresponding memory device. In at least one embodiment, lower level caches are implemented in various units within GPCs. In at least one embodiment, each of SMs **3314** may implement a level one ("L1") cache wherein L1 cache is private memory that is dedicated to a particular SM **3314** and data from L2 cache **3404** is fetched and stored in each of L1 caches for processing in functional units of SMs **3314**. In at least one embodiment, L2 cache **3404** is coupled to memory interface **3406** and XBar **3220**.

[0278] ROP unit **3402** performs graphics raster operations related to pixel color, such as color compression, pixel blending, and more, in at least one embodiment. ROP unit **3402**, in at least one embodiment, implements depth testing in conjunction with raster engine **3308**, receiving a depth for a sample location associated with a pixel fragment from culling engine of raster engine **3308**. In at least one embodiment, depth is tested against a corresponding depth in a depth buffer for a sample location associated with fragment. In at least one embodiment, if fragment passes depth test for sample location, then ROP unit **3402** updates depth buffer and transmits a result of depth test to raster engine **3308**. It will be appreciated that number of partition units **3400** may be different than number of GPCs and, therefore, each ROP unit **3402** can, in at least one embodiment, be coupled to each of GPCs. In at least one embodiment, ROP unit **3402** tracks packets received from different GPCs and determines which that a result generated by ROP unit **3402** is routed to through XBar **3220**.

[0279] FIG. **35** illustrates a streaming multi-processor ("SM") **3500**, in at least one embodiment. In at least one embodiment, SM **3500** is SM of FIG. **33**. In at least one embodiment, SM **3500** includes, without limitation, an instruction cache **3502**; one or more scheduler units **3504**; a register file **3508**; one or more processing cores ("cores")

3510; one or more special function units (“SFUs”) **3512**; one or more load/store units (“LSUs”) **3514**; an interconnect network **3516**; a shared memory/level one (“L1”) cache **3518**; and any suitable combination thereof. 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 task is associated with a shader program, task is allocated to one of SMs **3500**. In at least one embodiment, scheduler unit **3504** receives tasks from work distribution unit and manages instruction scheduling for one or more thread blocks assigned to SM **3500**. In at least one embodiment, scheduler unit **3504** schedules thread blocks for execution as warps of parallel threads, wherein each thread block is allocated at least one warp. In at least one embodiment, each warp executes threads. In at least one embodiment, scheduler unit **3504** manages a plurality of different thread blocks, allocating warps to different thread blocks and then dispatching instructions from plurality of different cooperative groups to various functional units (e.g., processing cores **3510**, SFUs **3512**, and LSUs **3514**) during each clock cycle.

[0280] In at least one embodiment, Cooperative Groups may refer to a programming model for organizing groups of communicating threads that allows developers to express granularity at which threads are communicating, enabling expression of richer, more efficient parallel decompositions. In at least one embodiment, cooperative launch APIs support synchronization amongst thread blocks for execution of parallel algorithms. In at least one embodiment, applications of programming models provide a single, simple construct for synchronizing cooperating threads: a barrier across all threads of a thread block (e.g., a `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 and multi-block granularities, and to perform collective operations such as synchronization on threads in a cooperative group. A 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.

[0281] In at least one embodiment, a dispatch unit **3506** is configured to transmit instructions to one or more of functional units and scheduler unit **3504** includes, without limitation, two dispatch units **3506** that enable two different instructions from same warp to be dispatched during each clock cycle. In at least one embodiment, each scheduler unit **3504** includes a single dispatch unit **3506** or additional dispatch units **3506**.

[0282] In at least one embodiment, each SM **3500**, in at least one embodiment, includes, without limitation, register file **3508** that provides a set of registers for functional units of SM **3500**. In at least one embodiment, register file **3508**

is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file **3508**. In at least one embodiment, register file **3508** is divided between different warps being executed by SM **3500** and register file **3508** provides temporary storage for operands connected to data paths of functional units. In at least one embodiment, each SM **3500** comprises, without limitation, a plurality of L processing cores **3510**. In at least one embodiment, SM **3500** includes, without limitation, a large number (e.g., 128 or more) of distinct processing cores **3510**. In at least one embodiment, each processing core **3510**, in at least one embodiment, includes, without limitation, a fully-pipelined, single-precision, double-precision, and/or mixed precision processing unit that includes, without limitation, a floating point arithmetic logic unit and an integer arithmetic logic unit. In at least one embodiment, floating point arithmetic logic units implement IEEE 754-2008 standard for floating point arithmetic. In at least one embodiment, processing cores **3510** include, without limitation, 64 single-precision (32-bit) floating point cores, 64 integer cores, 32 double-precision (64-bit) floating point cores, and 8 tensor cores.

[0283] Tensor cores are configured to perform matrix operations in accordance with at least one embodiment. In at least one embodiment, one or more tensor cores are included in processing cores **3510**. In at least one embodiment, tensor cores are configured to perform deep learning matrix arithmetic, such as convolution operations for neural network training and inferencing. In at least one embodiment, each tensor core operates on a 4x4 matrix and performs a matrix multiply and accumulate operation D=AxB+C, where A, B, C, and D are 4x4 matrices.

[0284] 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 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 CUDA level, warp-level interface assumes 16x16 size matrices spanning all 32 threads of warp.

[0285] In at least one embodiment, each SM **3500** comprises, without limitation, M SFUs **3512** that perform special functions (e.g., attribute evaluation, reciprocal square root, and like). In at least one embodiment, SFUs **3512** include, without limitation, a tree traversal unit configured to traverse a hierarchical tree data structure. In at least one embodiment, SFUs **3512** include, without limitation, a texture unit configured to perform texture map filtering operations. In at least one embodiment, texture units are configured to load texture maps (e.g., a 2D array of texels) from memory and sample texture maps to produce sampled texture values for use in shader programs executed by SM **3500**. In at least one embodiment, texture maps are stored in shared memory/L1

cache **3518**. In at least one embodiment, texture units implement texture operations such as filtering operations using mip-maps (e.g., texture maps of varying levels of detail), in accordance with at least one embodiment. In at least one embodiment, each SM **3500** includes, without limitation, two texture units.

[0286] Each SM **3500** comprises, without limitation, N LSUs **3514** that implement load and store operations between shared memory/L1 cache **3518** and register file **3508**, in at least one embodiment. Each SM **3500** includes, without limitation, interconnect network **3516** that connects each of functional units to register file **3508** and LSU **3514** to register file **3508** and shared memory/L1 cache **3518** in at least one embodiment. In at least one embodiment, interconnect network **3516** is a crossbar that can be configured to connect any of functional units to any of registers in register file **3508** and connect LSUs **3514** to register file **3508** and memory locations in shared memory/L1 cache **3518**.

[0287] In at least one embodiment, shared memory/L1 cache **3518** is an array of on-chip memory that allows for data storage and communication between SM **3500** and primitive engine and between threads in SM **3500**, in at least one embodiment. In at least one embodiment, shared memory/L1 cache **3518** comprises, without limitation, 128 KB of storage capacity and is in path from SM **3500** to partition unit. In at least one embodiment, shared memory/L1 cache **3518**, in at least one embodiment, is used to cache reads and writes. In at least one embodiment, one or more of shared memory/L1 cache **3518**, L2 cache, and memory are backing stores.

[0288] 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 capacity, texture and load/store operations can use remaining capacity. Integration within shared memory/L1 cache **3518** enables shared memory/L1 cache **3518** to function as a high-throughput conduit for streaming data while simultaneously providing high-bandwidth and low-latency access to frequently reused data, in accordance with at least one embodiment. In at least one embodiment, when configured for general purpose parallel computation, a simpler configuration can be used compared with graphics processing. In at least one embodiment, fixed function graphics processing units are bypassed, creating a much simpler programming model. In general purpose parallel computation configuration, 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 same program, using a unique thread ID in calculation to ensure each thread generates unique results, using SM **3500** to execute program and perform calculations, shared memory/L1 cache **3518** to communicate between threads, and LSU **3514** to read and write global memory through shared memory/L1 cache **3518** and memory partition unit. In at least one embodiment, when configured for general purpose parallel computation, SM **3500** writes commands that scheduler unit **3504** can use to launch new work on DPCs.

[0289] In at least one embodiment, 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, PPU is embodied on a single semiconductor substrate. In at least one embodiment, 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.

[0290] In at least one embodiment, PPU may be included on a graphics card that includes one or more memory devices. A graphics card may be configured to interface with a PCIe slot on a motherboard of a desktop computer. In at least one embodiment, PPU may be an integrated graphics processing unit ("iGPU") included in chipset of motherboard.

[0291] Inference and/or training logic **1015** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **1015** are provided below in conjunction with FIGS. **10** and/or **11**. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to SM **3500**. In at least one embodiment, SM **3500** is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by SM **3500**. In at least one embodiment, SM **3500** may be used to perform one or more neural network use cases described herein.

[0292] 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 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.

[0293] 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.

[0294] 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. Terms "comprising," "having," "including," and "containing" are to be construed as open-ended terms (meaning "including, but not limited to,") unless otherwise noted. term "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 indi-

ividually 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. 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.

[0295] 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 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). A 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.”

[0296] 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 by one or more processors of a computer system, cause computer system to perform operations described herein. A set of non-transitory computer-readable storage media, in at least one embodiment, 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.

[0297] 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.

[0298] 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.

[0299] 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.

[0300] 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.

[0301] 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.

[0302] 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. Terms "system" and "method" are used herein interchangeably insofar as system may embody one or more methods and methods may be considered a system.

[0303] 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. A 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 some implementations, process of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a serial or parallel interface. In another implementation, process 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. References may also be made to providing, outputting, transmitting, sending, or presenting analog or digital data. In various examples, process 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 inter-process communication mechanism.

[0304] Although discussion above sets 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 are defined above for purposes of discussion, various functions and responsibilities might be distributed and divided in different ways, depending on circumstances.

[0305] 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 claims.

What is claimed is:

1. A processor, comprising:

one or more arithmetic logic units (ALUs) to help detect one or more objects in an image based, at least in part, on one or more pseudo-labels corresponding to the one or more objects.

2. The processor of claim 1, wherein the one or more ALUs are further to be configured to:

detect the one or more objects using a neural network that is trained in part by generating a region of interest (ROI) score for each of a set of object proposals, the one or more pseudo-labels determined using the ROI scores.

3. The processor of claim 2, wherein the one or more ALUs are further to be configured to:

detect the one or more objects using a neural network that is trained in part by providing the ROI score to at least one refinement branch capable of producing a refined ROI score for the one or more pseudo-labels.

4. The processor of claim 3, wherein the at least one refinement branch performs at least one of ROI quantization and selection, or removal of pseudo-labels for redundant proposals.

5. The processor of claim 1, wherein the one or more ALUs are further to be configured to:

detect the one or more objects using a neural network that is trained in part by selecting the one or more pseudo-labels corresponding to the one or more objects having refined ROI scores that exceed an object threshold.

6. A system comprising:

one or more processors to be configured to detect, using one or more neural networks, one or more objects in an image based, at least in part, on one or more pseudo-labels corresponding to the one or more objects; and one or more memories to store the one or more neural networks.

7. The system of claim 6, wherein the one or more processors are further to be configured to:

detect the one or more objects using a neural network that is trained in part by generating a region of interest (ROI) score for each of a set of object proposals, the one or more pseudo-labels determined using the ROI scores.

8. The system of claim 7, wherein the one or more processors are further to be configured to:

detect the one or more objects using a neural network that is trained in part by providing the ROI score to at least one refinement branch capable of producing a refined ROI score for the one or more pseudo-labels.

9. The system of claim 8, wherein the at least one refinement branch performs at least one of ROI quantization and selection, or removal of pseudo-labels for redundant proposals.

10. The system of claim 6, wherein the one or more processors are further to be configured to:

detect the one or more objects using a neural network that is trained in part by selecting the one or more pseudo-labels corresponding to the one or more objects having refined ROI scores that exceed an object threshold.

11. A processor comprising:

one or more arithmetic logic units (ALUs) to help train one or more neural networks to be used to detect one or more objects in an image based, at least in part, on one or more pseudo-labels corresponding to the one or more objects.

12. The processor of claim 11, wherein the one or more ALUs are further to be configured to:

generate a region of interest (ROI) score for each of a set of object proposals, the one or more pseudo-labels determined using the ROI scores.

13. The processor of claim 12, wherein the one or more ALUs are further to be configured to:

provide the ROI score to at least one refinement branch capable of producing a refined ROI score for the one or more pseudo-labels.

14. The processor of claim 13, wherein the at least one refinement branch performs at least one of ROI quantization and selection, or removal of pseudo-labels for redundant proposals.

15. The processor of claim 11, wherein the one or more ALUs are further to be configured to:

select the one or more pseudo-labels corresponding to the one or more objects having refined ROI scores that exceed an object threshold.

16. A system comprising:

one or more processors to help train one or more neural networks to be used to detect one or more objects in an

- image based, at least in part, on one or more pseudo-labels corresponding to the one or more objects.
- 17.** The system of claim **16**, wherein the one or more processors are further to be configured to:
- generate a region of interest (ROI) score for each of a set of object proposals, the one or more pseudo-labels determined using the ROI scores.
- 18.** The system of claim **17**, wherein the one or more processors are further to be configured to:
- provide the ROI score to at least one refinement branch capable of producing a refined ROI score for the one or more pseudo-labels.
- 19.** The system of claim **18**, wherein the at least one refinement branch performs at least one of ROI quantization and selection, or removal of pseudo-labels for redundant proposals.
- 20.** The system of claim **17**, wherein the one or more processors are further to be configured to:
- select the one or more pseudo-labels corresponding to the one or more objects having refined ROI scores that exceed an object threshold.
- 21.** A method comprising:
- training one or more neural networks to detect one or more objects in an image based, at least in part, on one or more pseudo-labels corresponding to the one or more objects.
- 22.** The method of claim **21**, further comprising:
- generating a region of interest (ROI) score for each of a set of object proposals, the one or more pseudo-labels determined using the ROI scores.
- 23.** The method of claim **22**, further comprising:
- providing the ROI score to at least one refinement branch capable of producing a refined ROI score for the one or more pseudo-labels.
- 24.** The method of claim **23**, wherein the at least one refinement branch performs at least one of ROI quantization and selection, or removal of pseudo-labels for redundant proposals.
- 25.** The method of claim **21**, further comprising:
- selecting the one or more pseudo-labels corresponding to the one or more objects having refined ROI scores that exceed an object threshold.
- 26.** A system, comprising:
- a camera configured to capture an image;
 - one or more processors to be configured to detect, using one or more neural networks, one or more objects in the image based, at least in part, on one or more pseudo-labels corresponding to the one or more objects; and
 - a storage device to store information regarding the detected objects.
- 27.** The system of claim **26**, wherein the one or more processors are further to be configured to:
- detect the one or more objects using a neural network that is trained in part by generating a region of interest (ROI) score for each of a set of object proposals, the one or more pseudo-labels determined using the ROI scores.
- 28.** The system of claim **27**, wherein the one or more processors are further to be configured to:
- detect the one or more objects using a neural network that is trained in part by providing the ROI score to at least one refinement branch capable of producing a refined ROI score for the one or more pseudo-labels.
- 29.** The system of claim **28**, wherein the at least one refinement branch performs at least one of ROI quantization and selection, or removal of pseudo-labels for redundant proposals.
- 30.** The system of claim **26**, wherein the one or more processors are further to be configured to:
- detect the one or more objects using a neural network that is trained in part by selecting the one or more pseudo-labels corresponding to the one or more objects having refined ROI scores that exceed an object threshold.

* * * * *