



(51) International Patent Classification:

G06N 3/06 (2006.01)

G06T 1/60 (2006.01)

G06T 1/20 (2006.01)

(21) International Application Number:

PCT/US2021/045622

(22) International Filing Date:

11 August 2021 (11.08.2021)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

16/994,381

14 August 2020 (14.08.2020)

US

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

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

(72) Inventor: SKALJAK, Bojan; 2788 San Tomas Express-

way, Santa Clara, California 95051 (US).

(74) Agent: ROSENTHAL, Gregory, B. et al.; c/o IP Docket-

ing Dept., Davis Wright Tremaine LLP, 920 Fifth Avenue, Suite 3300, Seattle, Washington 98104-1610 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM,

AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

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

GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,

(54) Title: TECHNIQUES FOR TRAINING AND INFERENCE USING MULTIPLE PROCESSOR RESOURCES

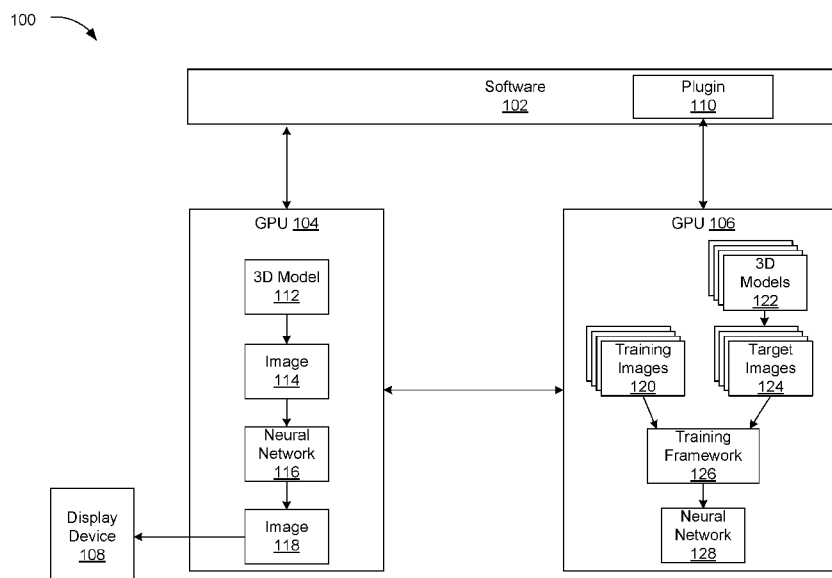


FIG. 1

(57) Abstract: Apparatuses, systems, and techniques for neural network training and inference using multiple processor resources. In at least one embodiment, one or more neural networks are used to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images.

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

**Declarations under Rule 4.17:**

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

**Published:**

- *with international search report (Art. 21(3))*
- *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

## 5      **TECHNIQUES FOR TRAINING AND INFERENCE USING MULTIPLE PROCESSOR RESOURCES**

### **CROSS REFERENCE TO RELATED APPLICATION**

[0001] This application claims priority to U.S. Patent Application No. 16/994,381, filed August 14, 2020, entitled “TECHNIQUES FOR TRAINING AND INFERENCE USING  
10      **MULTIPLE PROCESSOR RESOURCES,**” the entire contents of which is incorporated herein by reference in its entirety and for all purposes.

### **TECHNICAL FIELD**

[0002] At least one embodiment pertains to techniques to train one or more neural networks embedded in software and using multi-processor configuration. For example, at least one  
15      embodiment pertains to real-time deep neural network training and inferencing using one or more neural networks embedded in 3D software and using a multi-GPU configuration.

### **BACKGROUND**

[0003] Techniques to train a neural networks may utilize large amounts of data and may be very time consuming. For example, a process for training a neural network may involve  
20      capturing, storing, transferring, and preprocessing data. Amounts of memory, time, or computing resources used to perform computation operations can be improved.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

[0004] FIG. 1 illustrates a computing environment, in accordance with at least one embodiment;

25      [0005] FIG. 2 illustrates a computing environment in which a multi-processor configuration is implemented to facilitate multi-processor training and inference. In at least one embodiment;

[0006] FIG. 3 illustrates a computing environment in which at least one embodiment can be implemented;

30      [0007] FIG. 4 illustrates an example of a vehicle, according to at least one embodiment;

- 5     **[0008]** FIG. 5 illustrates real-time neural network training and inference using multi-GPU configuration, according to at least one embodiment;
- [0009]** FIG. 6 shows an illustrative example of a process for real-time neural network training and inference using multi-processor configuration, in accordance with at least one embodiment;
- 10    **[0010]** FIG. 7A illustrates an example of an autonomous vehicle, according to at least one embodiment;
- [0011]** FIG. 7B illustrates an example of camera locations and fields of view for the autonomous vehicle of FIG. 7A, according to at least one embodiment;
- [0012]** FIG. 7C is a block diagram illustrating an example system architecture for the  
15    autonomous vehicle of FIG. 7A, according to at least one embodiment;
- [0013]** FIG. 7D is a diagram illustrating a system for communication between cloud-based server(s) and the autonomous vehicle of FIG. 7A, according to at least one embodiment;
- [0014]** FIG. 8A illustrates inference and/or training logic, according to at least one embodiment;
- 20    **[0015]** FIG. 8B illustrates inference and/or training logic, according to at least one embodiment;
- [0016]** FIG. 9 illustrates an exemplary data center, in accordance with at least one embodiment;
- [0017]** FIG. 10 illustrates a processing system, in accordance with at least one embodiment;
- 25    **[0018]** FIG. 11 illustrates a computer system, in accordance with at least one embodiment;
- [0019]** FIG. 12 illustrates a system, in accordance with at least one embodiment;
- [0020]** FIG. 13 illustrates an exemplary integrated circuit, in accordance with at least one embodiment;
- [0021]** FIG. 14 illustrates a computing system, according to at least one embodiment;
- 30    **[0022]** FIG. 15 illustrates an APU, in accordance with at least one embodiment;
- [0023]** FIG. 16 illustrates a CPU, in accordance with at least one embodiment;
- [0024]** FIG. 17 illustrates an exemplary accelerator integration slice, in accordance with at least one embodiment;

- 5     **[0025]** FIGS. 18A and 18B illustrate exemplary graphics processors, in accordance with at least one embodiment;
- [0026]** FIG. 19A illustrates a graphics core, in accordance with at least one embodiment;
- [0027]** FIG. 19B illustrates a GPGPU, in accordance with at least one embodiment;
- [0028]** FIG. 20A illustrates a parallel processor, in accordance with at least one embodiment;
- 10    **[0029]** FIG. 20B illustrates a processing cluster, in accordance with at least one embodiment;
- [0030]** FIG. 20C illustrates a graphics multiprocessor, in accordance with at least one embodiment;
- [0031]** FIG. 21 illustrates a graphics processor, in accordance with at least one embodiment;
- [0032]** FIG. 22 illustrates a processor, in accordance with at least one embodiment;
- 15    **[0033]** FIG. 23 illustrates a processor, in accordance with at least one embodiment;
- [0034]** FIG. 24 illustrates a graphics processor core, in accordance with at least one embodiment;
- [0035]** FIG. 25 illustrates a PPU, in accordance with at least one embodiment;
- [0036]** FIG. 26 illustrates a GPC, in accordance with at least one embodiment;
- 20    **[0037]** FIG. 27 illustrates a streaming multiprocessor, in accordance with at least one embodiment;
- [0038]** FIG. 28 illustrates a software stack of a programming platform, in accordance with at least one embodiment;
- [0039]** FIG. 29 illustrates a CUDA implementation of a software stack of FIG. 28, in
- 25    accordance with at least one embodiment;
- [0040]** FIG. 30 illustrates a ROCm implementation of a software stack of FIG. 28, in accordance with at least one embodiment;
- [0041]** FIG. 31 illustrates an OpenCL implementation of a software stack of FIG. 28, in accordance with at least one embodiment;
- 30    **[0042]** FIG. 32 illustrates software that is supported by a programming platform, in accordance with at least one embodiment;

- 5     **[0043]** FIG. 33 illustrates compiling code to execute on programming platforms of FIGS. 28 - 31, in accordance with at least one embodiment;
- [0044]** FIG. 34 illustrates in greater detail compiling code to execute on programming platforms of FIGS. 28 - 31, in accordance with at least one embodiment;
- [0045]** FIG. 35 illustrates translating source code prior to compiling source code, in  
10    accordance with at least one embodiment;
- [0046]** FIG. 36A illustrates a system configured to compile and execute CUDA source code using different types of processing units, in accordance with at least one embodiment;
- [0047]** FIG. 36B illustrates a system configured to compile and execute CUDA source code of FIG. 36A using a CPU and a CUDA-enabled GPU, in accordance with at least one  
15    embodiment;
- [0048]** FIG. 36C illustrates a system configured to compile and execute CUDA source code of FIG. 36A using a CPU and a non-CUDA-enabled GPU, in accordance with at least one embodiment;
- [0049]** FIG. 37 illustrates an exemplary kernel translated by CUDA-to-HIP translation tool of  
20    FIG. 36C, in accordance with at least one embodiment;
- [0050]** FIG. 38 illustrates non-CUDA-enabled GPU of FIG. 36C in greater detail, in accordance with at least one embodiment;
- [0051]** FIG. 39 illustrates how threads of an exemplary CUDA grid are mapped to different compute units of FIG. 38, in accordance with at least one embodiment; and
- 25    **[0052]** FIG. 40 illustrates how to migrate existing CUDA code to Data Parallel C++ code, in accordance with at least one embodiment.

#### **DETAILED DESCRIPTION**

- [0053]** FIG. 1 illustrates a computing environment 100, in accordance with at least one embodiment. In at least one embodiment, FIG. 1 illustrates a computing environment for  
30    implementing real-time deep neural network training and inference embedded in 3D software and using multi-processor configuration. In at least one embodiment, computing environment 100 is implemented in context of a computer system with at least two discrete hardware graphics processing units. In at least one embodiment, a first processor resource (e.g., first GPU hardware device) is used to produce images and a second processor resource (e.g.,

5 second GPU hardware device) is used to perform preprocessing, training, inference, etc. as part of a deep learning framework. In at least one embodiment, feedback from training is generated and provided in real-time and is presented directly in running software which can speed up research and development. In at least one embodiment, FIG. 1 illustrates a computing environment 100 comprising software 102, graphics processing unit 104, graphics  
10 processing unit 106, and display device 108. In at least one embodiment, software 102 includes a plugin 110 that coordinates real-time multi-GPU training and inference. In at least one embodiment, techniques described in FIG. 1 are applicable in context of at least some of FIGS. 2-40.

[0054] In at least one embodiment, software 102 refers to a software application that runs on  
15 a computer system. In at least one embodiment, software 102 is a software application such as a game engine or 3D computer graphics program for making 3D animations, models, games, etc. In at least some embodiments, software 102 is an executable program that is launched on a computer system comprising at least one central processing unit and two graphics processing units. In at least one embodiment, software 102 is an executable that is  
20 launched in context of an operating system that runs on at least one CPU. In at least one embodiment, an operating system utilizes one or more CPUs to facilitate execution of an executable application such as software 102. In at least one embodiment, software 102 utilizes GPU 104 to render images.

[0055] In at least one embodiment, GPU 104 refers to a discrete physical hardware device  
25 that is communicatively coupled to a motherboard of a computer system that includes at least one CPU that executes software 102. In at least one embodiment, GPU 104 uses a 3D model 112 to render an image 114 of a scene. In at least one embodiment, images are rendered for a 3D computer game or animation which is shown to a user via a display device 108. In at least one embodiment, software 102 provides instructions to GPU 104 to render images (e.g.,  
30 frame) for a game or animation, which may exceed a rate of 20 frames per second. In at least one embodiment, GPU 104 is provided instructions to render image (e.g., frames) at a rate of 60 frames per second (FPS) or even higher, in some cases. In at least some embodiments, a frame (e.g., image) is rendered based at least in part on a 3D model that represents a scene. In at least one embodiment, a 3D model includes one or more light sources that affects how  
35 objects appear – for instance, based on placement of a light source, an object may cast a shadow. In at least one embodiment, GPU 104 utilizes a non-deterministic technique to render a 2D image from a 3D representation (e.g., model). In at least one embodiment, a ray

5 tracing or path tracing technique is utilized to render a 2D image of a 3D scene in a non-deterministic manner. In at least one embodiment, Monte Carlo path tracing is a non-deterministic technique that is utilized to render a 2D image of a 3D representation.

[0056] In at least one embodiment, 3D model 112 is used to generate one or more outputs such as image 114. In at least one embodiment, 3D model 112 comprises a set of objects  
10 arranged to form a scene. In at least one embodiment, a 2D image of a 3D model is rendered from a particular perspective, which may simulate a camera view of said scene from a particular location. In at least one embodiment, an object in a 3D model has various properties, including but not limited to any suitable combination of: base color (e.g., in neutral lighting); transparency; reflectiveness; brightness (e.g., whether an object has a light  
15 source); and more. In at least one embodiment, 3D model 112 generates an image that comprises a two-dimensional grid of final color values (e.g., separate RGB values for each pixel) representing how a scene is viewed from a certain perspective. In at least one embodiment, GPU 104 generates a set of properties for some or all pixels of an image that is rendered, including but not limited to any suitable combination of: final color; depth; normal;  
20 albedo; roughness; motion vector; and more. In at least one embodiment, 3D model 112 is used to generate images, diffuse maps, normal maps, albedo maps, roughness maps, motion vector maps, and various combinations thereof. A normal map (e.g., bump map) may refer to a map of RGB color values that correspond to X, Y, and Z coordinates of surface normals of an object. An albedo map may refer to a map that represents base colors at each position  
25 (e.g., pixel) without shadow effects, lighting effects, etc. that are included as part of a diffuse map. A roughness (gloss) map may refer to how rough a surface is. A motion vector map may refer to position, velocity, and acceleration of objects in a scene. In at least one embodiment, any suitable texture map is generated using 3D model 112 including but not limited to color maps; transparency maps; bump maps; specular maps; environment maps;  
30 light maps; and more.

[0057] In at least one embodiment, image 114 is generated from 3D model 112 using one or more non-deterministic techniques such as Monte Carlo path tracing which results in noise in image 114. In at least one embodiment, noise in image 114 is zero-mean with non-zero variance that affects final output color. In at least one embodiment, noise refers to Gaussian  
35 noise.

[0058] In at least one embodiment neural network 116 refers to one or more artificial neural networks 116 trained to accept image 114 as an input and generate image 118 as an output. In

5 at least one embodiment, one or more neural networks generate one or more second versions of one or more images based, at least in part, on a first version of said one or more images and a three-dimensional representation of said one or more first versions of said one or more images. In at least one embodiment, neural network 116 is a deep learning based denoiser. In at least one embodiment, parameters that control neural network 116 (e.g., weight values) are  
10 provided by plugin 110 during development and tuning phase to refine said parameters. In at least one embodiment, a first image generated from 3D model 112 is provided as an input to neural network 116 (e.g., denoiser) which generates a second image as image 118 (or a component thereof) which is provided to display device 108 and presented to a user such as a researcher or developer that is tuning neural network 116 in real-time.

15 **[0059]** In at least one embodiment, display device 108 is or comprises a liquid crystal display (LCD) monitor or any other suitable device for displaying images. In at least one embodiment, display device 108 is any suitable display device including but not limited to: computer monitor; television set; laptop screen; tablet screen; smart device screen; display integrated in an embedded augmented/virtual reality device; and more. In at least one  
20 embodiment, images provided to display device 108 are presented at a fixed or variable frequency (e.g., fixed 60 frames per second) as part of a video game, animation, video, etc.

**[0060]** In at least one embodiment, GPU 106 refers to a discrete physical hardware device that is communicatively coupled to a motherboard of a computer system that includes at least one CPU that executes software 102. In at least one embodiment, GPU 104 and GPU 106  
25 have different computational capabilities to perform an inference operation using one or more neural networks. In at least one embodiment, GPU 106, as compared to GPU 104, has greater computational capability to perform inference and/or training with neural networks. In at least one embodiment, GPU 104 is a data producer and GPU 106 is a consumer of such produced data. In at least one embodiment, GPU 104 produces frames (e.g., images) and  
30 pushes them to a consumer/producer queue in memory shared between GPU 104 and GPU 106. In at least one embodiment, GPU 104 In at least one embodiment, GPU 104 writes data to a ring buffer (also referred to as a cyclic buffer, circular buffer, circular queue, etc.) and GPU 106 reads data from said ring buffer. In at least one embodiment, images generated by GPU 104 (e.g., image 118 shown in FIG. 1) are shared with GPU 106. In at least one  
35 embodiment, GPU 104 uses 3D models to render images and those images are stored in shared memory accessible to GPU 106.

5     **[0061]** In at least one embodiment, 3D models 122 refer to 3D models that were used to generate corresponding images from training images 120 that were rendered by GPU 104. In at least one embodiment, due to limitations in computational capabilities, GPU 104 may utilize stochastic techniques such as Monte Carlo path tracing to simulate effects in a scene to generate an image of said scene. In at least one embodiment, GPU 106 has greater  
10    computational capabilities than GPU 104 and uses 3D models 122 of scenes to render target images 124. In at least one embodiment, target images 124 serve as ground truth data for training a neural network and are compared against training images 120 (e.g., generated by GPU 104). In at least one embodiment, training framework 126

15    **[0062]** FIG. 1 illustrates training and deployment of a deep neural network, according to at least one embodiment. In at least one embodiment, an untrained neural network is trained using a training dataset such as training images 120. In at least one embodiment, training framework 126 is a PyTorch framework, whereas in other embodiments, training framework 126 is a TensorFlow, Boost, Caffe, Microsoft Cognitive Toolkit/CNTK, MXNet, Chainer, Keras, Deeplearning4j, or other training framework. In at least one embodiment, training  
20    framework 126 trains an untrained neural network and enables it to be trained using processing resources described herein to generate a trained neural network 128. In at least one embodiment, neural network 128 is a denoiser with weights that may be different from neural network 116. 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  
25    in either a supervised, partially supervised, or unsupervised manner.

30    **[0063]** In at least one embodiment, an untrained neural network is trained using supervised learning, wherein a training dataset includes an input paired with a desired output for an input, or where training dataset includes input having a known output and an output of neural network is manually graded. In at least one embodiment, untrained neural network is trained  
35    in a supervised manner and processes inputs from training dataset and compares resulting outputs against a set of expected or desired outputs. In at least one embodiment, target images 124 are expected or desired outputs that training images 120 are compared against. In at least one embodiment, errors are propagated back through untrained neural network. In at least one embodiment, training framework 126 adjusts weights that control untrained neural network.  
In at least one embodiment, training framework 126 includes tools to monitor how well untrained neural network is converging towards a model, such as trained neural network 128, suitable to generating correct answers. In at least one embodiment, training framework 126

5 trains untrained neural network repeatedly while adjust weights to refine an output of untrained neural network using a loss function and adjustment algorithm, such as stochastic gradient descent. In at least one embodiment, training framework 126 trains untrained neural network until untrained neural network achieves a desired accuracy. In at least some embodiments, training framework 126 continuously trains a neural network on incoming data  
10 and completes training in response to a command sent from plugin 110. In at least one embodiment, trained neural network 128 can then be deployed to implement any number of machine learning operations. In at least one embodiment, neural network 128 is deployed on GPU 104 and replaces neural network 116. In at least one embodiment, neural network 128 is a denoiser.

15 **[0064]** In at least one embodiment, plugin 110 refers to a component of software 102 such as a static or dynamically linked library. In at least one embodiment, plugin 110 may be a component as part of a development framework, such as a software development kit. In at least one embodiment, during a development phase (e.g., during development of a game, creation of a 3D animated video) plugin 110 is utilized for real-time neural network training  
20 and inference. In at least one embodiment, plugin 110 is embedded in software 102. In at least one embodiment, plugin 110 has corresponding plugin components running in GPU 104 and GPU 106 to facilitate coordination of real-time multi-GPU training and inference. In at least one embodiment, plugin 110 is compiled into software 102 during development to allow for a researcher or engineer to refine a neural network such as a denoiser that will be included  
25 in a production version of software 102. In at least one embodiment, plugin 110 is a library compiled into software 102 during development or research but is later removed. In at least one embodiment, a final product (e.g., that is or includes a trained neural network) is exported as an NGX module using fast inference components (e.g., those used in DL4RT tool). In at least one embodiment, computing environment 100 is used to implement  
30 processes such as those described in connection with FIG. 6.

**[0065]** FIG. 2 illustrates a computing environment 200 in which a multi-processor configuration is implemented to facilitate multi-processor training and inference. In at least one embodiment, FIG. 2 illustrates real-time deep neural network training and inference embedded in 3D software and using multi-GPU configuration. In at least one embodiment,  
35 techniques described in FIG. 2 are applicable in context of at least some of FIGS. 1 and 3-40.

**[0066]** In at least one embodiment, ring buffer 202 refers to a region of shared memory that is accessible to both GPU 204 and GPU 206(1)...(N). In at least one embodiment, ring buffer

5 202 (also referred to as a cyclic buffer, circular buffer, circular queue, etc.) is implemented as a consumer/producer queue in memory (e.g., RAM). In at least one embodiment, data such as frames generated by GPU 204 to be shown on a display device are pushed to consumer/producer queue by GPU 204 and popped off of said queue by one of GPU 206(1)...(N). In at least one embodiment, ring buffer 202 is any suitable size and is reserved  
10 enough memory to store at least one frame of data. In an embodiment, ring buffer 202 is a first-in-first-out (FIFO) data structure in which data is stored (pushed) to end of the cyclic buffer and removed (popped) from front. In at least one embodiment, ring buffer 202 is used to implement a consumer/producer queue shared by GPU 204 and GPU 206(1)...(N). In at least one embodiment, ring buffer 202 (or a portion thereof) is populated by GPU 204, which  
15 continuously stores frame data in slots  $i$ ,  $i+1$ ...,  $i+w$  wherein ring buffer 202 has a capacity of  $w+1$ . In at least one embodiment, one of GPU 206(1)...(N) pops frame data of  $i$  from front of ring buffer 202 and provides such data to respective training framework 214(1)...(N) to train a neural network. In at least one embodiment, GPU 204 writes to next slots in ring buffer 202 and overwrites any data previously located at such slot – for instance, GPU 204 may write  
20 frame data for frame  $i$  at slot  $i$  of ring buffer 202 and continue writing frame data to ring buffer 202 such that when it writes frame data for frame  $i+w+1$ , ring buffer 202 wraps around and overwrites slot  $i$  with frame data for frame  $i+w+1$ .

[0067] In at least one embodiment, GPU 204 pushes frame data to ring buffer 202 at a fixed rate or target fixed rate (e.g., target 60 frames per second, with possible slowdowns or  
25 speedups) based on how quickly GPU 204 is to render frames to a display device, and GPU 206(1)...(N) pops frame data from ring buffer 202 and processes such frame data at least as quickly as GPU 204 pushes frame data, on average. In at least one embodiment, GPU 206(1)...(N) pops frame data from ring buffer 202 at a same rate as GPU 206 pushes frame data to ring buffer 202 and holds such data in a separate memory space (e.g., a cache shared  
30 by GPU 206(1)...(N)) to prevent stalls that would result from GPU 204 waiting for GPU 206(1)...(N) to vacate a slot in ring buffer 202; GPU 206(1)...(N) may use this separate memory space to temporarily store frame data while it catches up on processing frame data, and then continues to read from ring buffer 202 directly when said separate memory space is emptied (e.g., as a result of GPU 206(1)...(N) catching up). In at least one embodiment, GPU  
35 206(1)...(N) pops frame data at a 4 frames delay to prevent stalling GPU and allow software to run at full speed.

5 [0068] In at least one embodiment, GPU 204 is implemented in accordance with at least a GPU described in connection with FIG. 1 that is utilized to render images. In at least one embodiment, GPU 206(1)...206(N) refers to an array of N GPUs (e.g., GPU 206(1), GPU 206(2), and so on) that collectively work together to train a neural network. While an array of N GPUs is illustrated in FIG. 2, any suitable number of GPUs may be utilized, which may  
10 include N=1. In at least one embodiment, GPU 204 renders frames based on 3D models where objects, perspective, etc. may be dynamically moving.

[0069] In at least one embodiment, GPU 206(1)...(N) obtain images generated by GPU 204 and uses them as training images. In at least one embodiment, GPU 206(1)...(N) obtain frame data 210 used by GPU to render images via ring buffer 202 and generate target images from  
15 such frame data. In at least one embodiment, training images comprise of a set of images.

[0070] In at least one embodiment, frame data 210 refers to data that is used to generate an image or frame which may be displayed. In at least one embodiment, various frame data 210 including but not limited to various combinations of final color, albedo, depth, normals, and roughness are stored in a slot of ring buffer 202. In at least one embodiment, 3D model used  
20 to generate a frame is pushed to a slot of ring buffer 202 in connection with frame data 210. In at least one embodiment, an object in a 3D model has various properties, including but not limited to any suitable combination of: base color (e.g., in neutral lighting); transparency; reflectiveness; brightness (e.g., whether an object has a light source); and more. In at least one embodiment, 3D model generates an image that comprises a two-dimensional grid of  
25 final color values (e.g., separate RGB values for each pixel) representing how a scene is viewed from a certain perspective. In at least one embodiment, GPU 204 generates frame data for a rendered frame an image that is rendered, including but not limited to any suitable combination of: final color; depth; normal; albedo; roughness; motion vector; and more. In at least one embodiment, final color to be displayed is modified by a neural network (e.g.,  
30 denoiser). In at least one embodiment, frame data 210 comprises final color map, diffuse maps, normal maps, albedo maps, roughness maps, motion vector maps, and various combinations thereof. A normal map (e.g., bump map) may refer to a map of RGB color values that correspond to X, Y, and Z coordinates of surface normals of an object. An albedo map may refer to a map that represents base colors at each position (e.g., pixel) without  
35 shadow effects, lighting effects, etc. that are included as part of a diffuse map. A roughness (gloss) map may refer to how rough a surface is. A motion vector map may refer to position, velocity, and acceleration of objects in a scene.

5 [0071] In at least one embodiment, plugin 212(1)...(N) are running asynchronously on respective GPU 206(1)...(N) and collectively train a neural network (e.g., denoiser). In at least one embodiment, plugin 212(1)...(N) performs pre-processing (break frame in tiles, perform augmentation, etc.) and feeds training framework 214(1)...(N) with data to train a neural network. In at least one embodiment, training is performed continuous on GPU  
10 206(1)...(N) using training framework 214(1)...(N) and when new frames arrive, information such as loss, latest inference results, etc. are captures and passed back to running software so that it can be shown in real-time. Software UI may be modified to present loss function from training, preview of inference frame (current state of training) or any other data which may be useful. In at least one embodiment, trained weights are serialized so training can be restarted  
15 from a specific point, from scratch, etc.

[0072] In at least one embodiment, GPU 204 and GPU 206(1)...(N) are a non-limiting instances of a specific type of processor resource, namely, a GPU. In at least one embodiment, different types of processor resources may be utilized in place of one or more GPUs. In at least one embodiment, a processor resource that can be used in place of a GPU  
20 includes one or more streaming multiprocessors, one or more compute units, one or more warps, one or more threads, and more. In at least one embodiment, one or more GPUs described in connection with FIGS. 1-40 may be replaced by one or more processor resources. In at least one embodiment, computing environment 200 is used to implement processes such as those described in connection with FIG. 6.

25 [0073] FIG. 3 illustrates a computing environment 300 in which at least one embodiment can be implemented. In at least one embodiment, a computer system comprises or otherwise utilizes software 302, GPU 304, GPU 306, and display device 308. In at least one embodiment, a user 310 is a researcher or developer that views display device 308 and interacts with software 302 using a human interface device (e.g., keyboard and mouse) via a  
30 graphical user interface.

[0074] In at least one embodiment, software 302 refers to a software application that runs on a computer system. In at least one embodiment, a software 302 is an executable application that is launched in context of an operating system. In at least one embodiment, software 302 is a computer game or 3D computer graphics program for making 3D animations, models,  
35 games, etc. In at least one embodiment, software 302 is optionally compiled with plugin 312 (e.g., as a DLL or static library) and communicates with corresponding plugin 314 and plugin 316 code on GPU 304 and GPU 306, respectively to facilitate real-time multi-GPU training

5 and inference. In at least one embodiment, software 302 or a component thereof (e.g., plugin 312) utilizes GPU 304 to render visual data to be shown to user 310 via display device 308 and receives training information from GPU 306 such as loss, latest inference results, and so on.

[0075] In at least some embodiments, GPU 304 renders frames based on a 3D model of a scene that may include lighting effects which are simulated using a non-deterministic algorithm such as Monte Carlo path tracing. In at least one embodiment, an image generated using Monte Carlo path tracing is referred to as a noisy image that includes Gaussian noise, wherein amounts of noise can be reduced by increasing sample size used for Monte Carlo path tracing – however, increasing sample size may increase computational work required to render a frame or image. In at least one embodiment, a noisy frame or image generated by GPU 304 is fed as an input to neural network 318 to generate a different image. In at least one embodiment, neural network 318 is a denoiser, input is a noisy image, and output is a denoised image that is provided to display device 308 to show to user 310.

[0076] In at least one embodiment, frame data generated by GPU 304 is collected by plugin 314 and shared with GPU 306. In at least one embodiment, plugin 314 shares a final image (e.g., same frame or image provided to display device 308) and frame data used to generate said final image, which may include a 3D model used to generate said final image, different types of maps (e.g., depth; normal; albedo; roughness; motion vector maps) and various combinations thereof. In at least one embodiment, plugin 314 pushes frame data to a ring buffer allocated in memory shared with GPU 306. In at least one embodiment, plugin 314 is an optional component that runs on GPU 304 in response to instructions from plugin 312 running in software 302.

[0077] In at least one embodiment, GPU 306 runs plugin 316 code or software that access frame data generated by GPU 304 as part of a training process. In at least one embodiment, GPU 306 pops frame data from a ring buffer. In at least one embodiment, GPU 306 obtains images generated by GPU 304 using neural network 318, which serve as training images for a neural network training framework. In at least one embodiment, each training image that GPU 306 obtains has a corresponding 3D model or other information that GPU 306 uses to generate a ground truth data that is used as a reference to compare against images rendered by GPU 304. In at least one embodiment, GPU 306 obtains a 3D model and an image of said 3D model that was rendered and denoised by GPU 304 and, renders a reference image from said 3D model, and uses such reference model to train a denoiser neural network. In at least one

5 embodiment, GPU 304 and GPU 306 both use Monte Carlo path tracing to render an image of a scene, but GPU 306 does so using a greater number of samples to render a more accurate representation of said scene which serves as a ground truth image. In at least one embodiment, an array of GPUs can be used to train a neural network (e.g., denoiser) rather than a single GPU (e.g., GPU 306 shown in FIG. 3). In at least one embodiment, plugin 316  
10 of GPU 306 trains a neural network as frame and image data is being collected in real-time and provides training information (e.g., loss, latest inference results, etc.) to plugin 312 of software 302. In at least one embodiment, plugin 312 provides such training information to GPU 304 to render on display device 308 – for instance, as an overlay. In at least one embodiment, user 310 views training information on display device 308 in real-time (e.g.,  
15 with a 4-frame delay or other negligible delay) and controls a training process in real-time via plugin 312. In at least one embodiment, user 310 submits commands to software 302 to control how training proceeds – for instance, user 310 may use plugin 312 of software 312 to import training parameters from a neural network being trained on GPU 306 to neural network 318 of GPU 304 to see how adjustments to neural network 318 affect results on  
20 display device 308. In at least one embodiment, amounts of time and resource spent to determine suitable parameters for a denoiser or other neural networks for improving graphical outputs can be improved by using techniques described herein for real-time inference and training using a multi-GPU configuration. In at least one embodiment, computing environment 300 is used to implement processes such as those described in connection with  
25 FIG. 6.

[0078] FIG. 4 illustrates an example of a vehicle 400, according to at least one embodiment. In at least one embodiment, vehicle 400 is an autonomous vehicle in accordance with those described in connection with FIG. 7A. In at least one embodiment, FIG. 4 describes an environment to implement real-time unsupervised training. In at least one embodiment,  
30 vehicle 400 comprises software 402 that interacts with camera 404, system on chip (“SoC”) 406, and GPU 408. In at least one embodiment, includes one or more electronic systems execute software 402. In at least one embodiment, a plugin 410 is compiled into software 402 during development to train a neural network and final version of said software with trained neural network does not have plugin 410.

35 [0079] In at least one embodiment, vehicle 400 comprises camera 404 or other image capturing device that is used to capture frame data at a fixed or variable frequency. In at least one embodiment, camera 404 captures an image 412 and image 412 is provided to SoC 406.

5 In at least one embodiment, SoC 406 uses image 412 as an input to neural network 414. In at least one embodiment, neural network 414 enhances image 412 by removing fog or rain to produce a clearer image which can be displayed to a driver of vehicle 400, provided to a classification neural network, and so on. In at least one embodiment, neural network 414 is a classification neural network that performs object detection and/or classification on image  
10 412 to detect presence of other vehicles, pedestrians, road signs, etc. In at least one embodiment result 416 is a modified version of image 410. In at least one embodiment, result 416 is an image with weather effects removed or attenuated to allow for better driving decisions to be made by an operator of vehicle, which may be a human or autonomous system). In at least one embodiment, result 416 is or includes a classification, such as a  
15 bounding box around an object detected in image 412 and a classification identifying said object.

[0080] In at least one embodiment, data is shared between SoC 406 and GPU 408. In at least one embodiment, SoC 406 and GPU 408 interact in a producer/consumer relationships wherein SoC 406 provides data to a shared memory space with GPU 408. In at least one  
20 embodiment, image 412 and/or result 416 is shared with GPU 408. In at least one embodiment, GPU 408 uses training data 418 to train a neural network in an unsupervised manner. In at least one embodiment, training data 418 comprises a batch of data that is collected from SoC 406 and/or camera 404 over a period of time. In at least one embodiment, a contiguous sequence of images captured by camera 404 forms a batch of training data. In at  
25 least one embodiment, untrained training framework 420 is used to train a neural network using unsupervised learning, wherein untrained neural network attempts to train itself using unlabeled data. In at least one embodiment, unsupervised learning training data 418 will include input data without any associated output data or “ground truth” data. In at least one embodiment, untrained neural network can learn groupings within training data 418 and can  
30 determine how individual inputs are related to untrained data. In at least one embodiment, unsupervised training can be used to generate a self-organizing map in trained neural network 422 capable of performing operations useful in reducing dimensionality of new dataset. In at least one embodiment, unsupervised training can also be used to perform anomaly detection, which allows identification of data points in new dataset that deviate from normal patterns of  
35 new dataset. In at least one embodiment, neural network 422 is provided by GPU 408 to plugin 410 and results generated by unsupervised training framework may be presented via a graphical user interface and software 402 may update neural network 414 located in SoC 406

5 based on training information provided to software 402 by GPU 408. In at least one embodiment, vehicle 400 is used to implement processes such as those described in connection with FIG. 6.

[0081] FIG. 5 illustrates real-time neural network training and inference using multi-GPU configuration, according to at least one embodiment. In at least one embodiment, CPU 500  
10 refers to a processor that controls execution of a software application such as a 3D computer game or 3D animation or video rendering software. In at least one embodiment, GPU 502 is utilized to render one or more frames which are presented to a user and using inferencing. In at least one embodiment, GPU 504 is utilized to train a neural network and provide training information to CPU 500.

15 [0082] In at least one embodiment, a computer system running a software application utilizes CPU 500, GPU 502, and GPU 504 for time neural network raining and inference in a multi-GPU configuration. In at least one embodiment, CPU 500 provides parameters for a neural network such as a denoiser. In at least one embodiment, GPU 502 initializes a neural network (e.g., denoiser) according to parameters supplied by CPU 500. In at least one embodiment  
20 CPU 500 provides a 3D model to GPU 502 with commands to render frame based on said 3D model. In at least one embodiment, GPU 502 renders a noisy 2D image from a 3D model using a non-deterministic rendering technique such as Monte Carlo path tracing, where images can be rendered using any number of samples wherein more samples correlates with better image quality but greater computational demand. In at least one embodiment, GPU 502  
25 uses a neural network (e.g., denoiser) to generate a denoised image. In at least one embodiment, a denoised image is presented on a display device. In at least one embodiment, denoised image and 3D model are pushed to a ring buffer in a region of memory that is shared with GPU 504. In at least one embodiment, GPU 504 obtains denoised images and 3D models from GPU 502, generate reference images from said 3D models, provides a training  
30 framework denoised images as training images, and images rendered by GPU 504 as target images to generate a denoiser. In at least one embodiment, parameters for a resulting denoiser are provided by GPU 504 to CPU 500. In at least one embodiment, a user is able to review denoised image provided by GPU 502 and/or training information. In at least one  
embodiment, a user reviews training information supplied by GPU 504 and if a user deems  
35 that parameters of trained neural network (denoiser) of GPU 504 appear suitable, then parameters of neural network GPU 502 may be modified to use these updated parameters.

5     **[0083]** FIG. 6 shows an illustrative example of a process 600 for real-time neural network training and inference using multi-processor configuration, in accordance with at least one embodiment. In at least one embodiment, some or all of the process 600 (or any other processes described herein, or variations and/or combinations thereof) is performed under the control of one or more computer systems configured with computer-executable instructions and may be implemented as code (e.g., computer-executable instructions, one or more computer programs, or one or more applications) executing collectively on one or more processors, by hardware, software, or combinations thereof. The code, in at least one embodiment, is stored on a computer-readable storage medium in the form of a computer program comprising a plurality of computer-readable instructions executable by one or more processors. The computer-readable storage medium, in at least one embodiment, is a non-transitory computer-readable medium. In at least one embodiment, at least some of the computer-readable instructions usable to perform the process 600 are not stored solely using transitory signals (e.g., a propagating transient electric or electromagnetic transmission). A non-transitory computer-readable medium does not necessarily include non-transitory data storage circuitry (e.g., buffers, caches, and queues) within transceivers of transitory signals. In at least one embodiment, process 6 is implemented in accordance with at least some of FIGS. 1-5 and 7-40.

25     **[0084]** In at least one embodiment software performing process 600 is used to obtain 602 an image that has Gaussian noise. IN at least one embodiment, software performing process 600 produces noisy images. In at least one embodiment, software performing process 600 may be a game engine or 3D computer graphics program. In at least one embodiment, a noisy image is produced using a non-deterministic algorithm such as Monte Carlo path tracing algorithm

30     **[0085]** In at least one embodiment, a real-time deep learning training plugin is integrated with software and initialized 604 using network structure provided in a script that encodes parameters for a neural network and/or neural network training framework including but not limited to weights of a neural network, loss functions, training hyperparameters, and more.

35     **[0086]** In at least one embodiment, a first GPU or processor resource produces 606 images or frames on behalf of software (e.g., game engine). In at least one embodiment, images or frames are generated using a 3D model to render a 2D image of a scene using a non-deterministic algorithm such as Monte Carlo path tracing. In at least one embodiment, a noisy image generated by a path tracing or ray tracing algorithm is provided as an input to a denoiser neural network to generate a denoised image.

5 [0087] In at least one embodiment, a first GPU shares 608 frame data to a ring buffer that is shared with a second GPU. In at least one embodiment, frame data is pushed to a ring buffer by a first GPU and popped off of said ring buffer by a second GPU in a consumer/producer queue that is shared between said first and second GPUs. In at least one embodiment, frame data includes but is not limited to any suitable combination of: final color; depth; normal;  
10 albedo; roughness; motion vector; and more. In at least one embodiment, a 3D model used by a first GPU to render an image is shared with a second GPU.

[0088] In at least one embodiment, a plugin running asynchronously on a second GPU dequeues or otherwise obtains 610 frame data from a ring buffer that was previously enqueued by a first GPU. In at least one embodiment, second GPU obtains a noisy image  
15 generated by a first GPU and a 3D model used to generate said noisy image.

[0089] In at least one embodiment, a plugin running on a second GPU uses frame data to train 612 a neural network. In at least one embodiment, a second GPU trains a denoiser neural network and provides training information such as loss information, latest inference results, weights, etc. to a plugin running on a software application. In at least one  
20 embodiment, weights or other results from training performed on a second GPU are used to update 614 a neural network that is running on a first GPU.

[0090] In at least one embodiment, a processor comprises one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of said one or more images and a three-dimensional  
25 representation of said one or more first versions of said one or more images. In at least one embodiment, one or more circuits are to use a first processor resource to generate said first one or more images based at least in part on said three-dimensional representation; provide said first one or more images and said three-dimensional representation to a second processor resource; and use said second processor resource to train said one or more neural network to  
30 generate said one or more second versions of said one or more images using said first version of said one or more images and said three-dimensional representation. In at least one embodiment, said one or more circuits are to: provide said one or more images and said three-dimensional representation to said second processor resource via a ring buffer, wherein:  
said first processor resource is to write said first version of said one or more images and said  
35 three-dimensional representation to said ring buffer; and said second processor resource is to read said first version of said one or more images and said three-dimensional representation from said ring buffer; and wherein said ring buffer is allocated in memory shared between

5 said first processor resource and said second processor resource. In at least one embodiment, said second processor resource is to read said first version of said one or more images and three-dimensional representation from said ring buffer with a four-frame delay after said first processor resource writes said first version of said one or more images to said ring buffer. In at least one embodiment, said one or more circuits are to provide additional image data  
10 associated with said first one or more images to said ring buffer. In at least one embodiment, additional image data includes depth data, normal data, albedo data, roughness data, or motion vector data. In at least one embodiment, said first processor resource comprises a first graphics processing unit (GPU) and said second processor resource comprises a second GPU.

[0091] In at least one embodiment, a system comprises: one or more processors to use one or  
15 more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of said one or more images and a three-dimensional representation of said one or more first versions of said one or more images. In at least one embodiment, said one or more processors comprises a first processor resource to execute a software application comprising a plugin that provides said three-dimensional representation  
20 to a second processor resource that generates said first version of said one or more images and controls training of said one or more neural networks on a third processor resource. In at least one embodiment, said second processor resource is connected to a display device for presenting a version of said one or more images. In at least one embodiment, said plugin controls whether said first version or said second version of said one or more images is to be  
25 presented on said display device. In at least one embodiment, said plugin is to: determine a set of parameters from training said one or more neural networks; and update a different one or more neural networks used by said first processor resource to generate said first version of said one or more images to use said set of parameters. In at least one embodiment, said plugin is to receive training information from said third processor resource; and said first processors  
30 resource provides said training information to said second processors resource to be presented using said display device.

[0092] In at least one embodiment, a machine-readable medium having stored thereon a set of instructions, which if performed by one or more processor resources, causes said one or more processor resources to at least: use one or more neural networks to generate a second  
35 version of one or more images based, at least in part, on a first version of said one or more images and a three-dimensional representation of said one or more first versions of said one or more images. In at least one embodiment, said set of instructions include instructions to:

5 use a first processor resource of said one or more processor resources to render said first version of said one or more images based at least in part on said three-dimensional representation and a second one or more neural networks; provide said first version of said one or more images and said three-dimensional representation to a second processor resource of said one or more processor resources; and use said second processor resource to train said  
10 one or more neural network to generate said second version of said one or more images using said first version of said one or more images and said three-dimensional representation. In at least one embodiment, said first processor resource to render a noisy version of said one or more images using said three-dimensional representation; and use said second one or more neural networks to generate a denoised version of said one or more images from said noisy  
15 version of said one or more images, wherein said denoised version is said first version. In at least one embodiment, said noisy version of said one or more images are to be rendered using a non-deterministic algorithm. In at least one embodiment, said non-deterministic algorithm is a Monte Carlo path tracing algorithm. In at least one embodiment, first processor resource is to render said first version of said one or more images using a first number of samples and  
20 said second processor resource is to render said second version of said one or more images using a second number of samples that is greater than said first number. In at least one embodiment, said second version of said one or more images are used as ground truth data to train said one or more neural networks. In at least one embodiment, said first processor resource is a graphics processing unit (GPU) and said second processor resource is an array  
25 of GPUs to collectively train said one or more neural networks.

[0093] In at least one embodiment, a processor comprises: two or more processor resources having different computational capabilities to perform an inference operation using a first version of one or more neural networks and a first processor resource of said two or more processor resources and to train a second version of said one or more neural networks using a  
30 second processor resource of said two or more processor resources. In at least one embodiment, said first processor resource is to: render a first image based, at least in part, on a three-dimensional model; generate a second image from said first image using said first version of said one or more neural networks; and provide said second image and said three-dimensional model to said second processor resource; and said second processor resource is  
35 to use said second image and said three-dimensional model to train said second version of said one or more neural networks. In at least one embodiment, wherein said first processor resource is to push at least said second image to end of a queue and said second processor

5 resource is to pop at least said second image from front of said queue. In at least one embodiment, said first processor resource is to push images to said queue at a rate of 60 images per second or more. In at least one embodiment, said first processor resource pushes additional data associated with said second image to said queue that includes depth data, normal data, albedo data, roughness data, or motion vector data. In at least one embodiment, 10 said three-dimensional model is to be used by said second processor resource to generate a ground truth image that is to be compared against said second image as part of training said second version of said one or more neural networks. In at least one embodiment, said one or more neural networks is a denoiser neural network, further wherein said first version and said second version of said one or more neural networks have differing weights. In at least one 15 embodiment, said first processor resource comprises a first graphics processing unit (GPU) and said second processor resource comprises a second GPU.

[0094] In at least one embodiment, a system comprises: two or more processor resources having different computational capabilities to perform an inference operation using a first version of one or more neural networks and a first processor resource of said two or more 20 processor resources and to train a second version of said one or more neural networks using a second processor resource of said two or more processor resources. In at least one embodiment, said system further comprises a third processor resource of said two or more processor resources to execute a software application comprising a plugin that provides a three-dimensional representation to said first processor resource to generate one or more 25 images using said first version of said one or more neural networks and controls training of said second version of said one or more neural networks on said second processor resource. In at least one embodiment, said first processor resource is connected to a display device for presenting images generated by said first processor resource. In at least one embodiment, said first processor resource is to generate a first version of an image using said three-dimensional representation and generate a second version of said image using said first version of said one 30 or more neural networks. In at least one embodiment, said plugin controls whether said first version or said second version of said image is to be presented on said display device. In at least one embodiment, said plugin is to: obtain a set of parameters from training said second version of said one or more neural networks; and update said first version of said one or more 35 neural networks to use said set of parameters. In at least one embodiment, said plugin is to provide said set of parameters to said first processor resource to render on a display device. In at least one embodiment, said plugin is an optional component of said software application.

5 [0095] In at least one embodiment, a machine-readable medium having stored thereon a set of instructions, which if performed by two or more processor resources having different computational capabilities, causes said two or more processor resources to at least: perform an inference operation using a first version of one or more neural networks and a first processor resource of said two or more processor resources and to train a second version of  
10 said one or more neural networks using a second processor resource of said two or more processor resources. In at least one embodiment, said set of instructions include instructions to: use said inference operation to generate a first version of one or more images based at least in part on a three-dimensional representation; provide said first version of said one or more images and said three-dimensional representation to said second processor resource;  
15 and train said second version of said one or more neural network to generate a second version of said one or more images using said first version of said one or more images and said three-dimensional representation. In at least one embodiment, said first processor resource is to: render a noisy version of said one or more images using said three-dimensional representation; and use said first version of said one or more neural networks to generate a  
20 denoised version of said one or more images from said noisy version of said one or more images, wherein said denoised version of said one or more images is said first version of said one or more images. In at least one embodiment, said noisy version of said one or more images are to be rendered using a non-deterministic algorithm. In at least one embodiment, said non-deterministic algorithm is a Monte Carlo path tracing algorithm.

25 [0096] FIG. 8A illustrates inference and/or training logic 815 used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 815 are provided below in conjunction with FIGS. 8A and/or 8B.

[0097] In at least one embodiment, inference and/or training logic 815 may include, without  
30 limitation, code and/or data storage 801 to store forward and/or output weight and/or input/output data, and/or other parameters to configure neurons or layers of a neural network trained and/or used for inferencing in aspects of one or more embodiments. In at least one embodiment, training logic 815 may include, or be coupled to code and/or data storage 801 to store graph code or other software to control timing and/or order, in which weight and/or  
35 other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs). In at least one embodiment, code, such as graph code, loads weight or other parameter information into processor ALUs

5 based on an architecture of a neural network to which such code corresponds. In at least one embodiment, code and/or data storage 801 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  
10 embodiment, any portion of code and/or data storage 801 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

[0098] In at least one embodiment, any portion of code and/or data storage 801 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or code and/or data storage 801 may be cache memory,  
15 dynamic randomly addressable memory ("DRAM"), static randomly addressable memory ("SRAM"), non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, a choice of whether code and/or code and/or data storage 801 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash or some other storage type may depend on available storage on-chip versus off-chip, latency requirements  
20 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, inference and/or training logic 815 may include, without limitation, a code and/or data storage 805 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  
25 for inferencing in aspects of one or more embodiments. In at least one embodiment, code and/or data storage 805 stores weight parameters and/or input/output data of each layer of a neural network trained or used in conjunction with one or more embodiments during backward propagation of input/output data and/or weight parameters during training and/or inferencing using aspects of one or more embodiments. In at least one embodiment, training  
30 logic 815 may include, or be coupled to code and/or data storage 805 to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs)).

[0100] In at least one embodiment, code, such as graph code, causes the loading of weight or  
35 other parameter information into processor ALUs based on an architecture of a neural network to which such code corresponds. In at least one embodiment, any portion of code and/or data storage 805 may be included with other on-chip or off-chip data storage,

5 including a processor's L1, L2, or L3 cache or system memory. In at least one embodiment, any portion of code and/or data storage 805 may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or data storage 805 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, a choice of whether code  
10 and/or data storage 805 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash memory or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

15 **[0101]** In at least one embodiment, code and/or data storage 801 and code and/or data storage 805 may be separate storage structures. In at least one embodiment, code and/or data storage 801 and code and/or data storage 805 may be a combined storage structure. In at least one embodiment, code and/or data storage 801 and code and/or data storage 805 may be partially combined and partially separate. In at least one embodiment, any portion of code and/or  
20 data storage 801 and code and/or data storage 805 may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

**[0102]** In at least one embodiment, inference and/or training logic 815 may include, without limitation, one or more arithmetic logic unit(s) ("ALU(s)") 810, including integer and/or floating point units, to perform logical and/or mathematical operations based, at least in part  
25 on, or indicated by, training and/or inference code (e.g., graph code), a result of which may produce activations (e.g., output values from layers or neurons within a neural network) stored in an activation storage 820 that are functions of input/output and/or weight parameter data stored in code and/or data storage 801 and/or code and/or data storage 805. In at least one embodiment, activations stored in activation storage 820 are generated according to  
30 linear algebraic and or matrix-based mathematics performed by ALU(s) 810 in response to performing instructions or other code, wherein weight values stored in code and/or data storage 805 and/or data storage 801 are used as operands along with other values, such as bias values, gradient information, momentum values, or other parameters or hyperparameters, any or all of which may be stored in code and/or data storage 805 or code and/or data storage  
35 801 or another storage on or off-chip.

**[0103]** In at least one embodiment, ALU(s) 810 are included within one or more processors or other hardware logic devices or circuits, whereas in another embodiment, ALU(s) 810 may

5 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 810 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  
10 at least one embodiment, code and/or data storage 801, code and/or data storage 805, and activation storage 820 may share a processor or other hardware logic device or circuit, whereas in another embodiment, they may be in different processors or other hardware logic devices or circuits, or some combination of same and different processors or other hardware logic devices or circuits. In at least one embodiment, any portion of activation storage 820  
15 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.

20 **[0104]** In at least one embodiment, activation storage 820 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, activation storage 820 may be completely or partially within or external to one or more processors or other logical circuits. In at least one embodiment, a choice of whether activation storage 820 is internal or external to a processor, for example, or comprising  
25 DRAM, SRAM, flash memory or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

**[0105]** In at least one embodiment, inference and/or training logic 815 illustrated in FIG. 8A  
30 may be used in conjunction with an application-specific integrated circuit ("ASIC"), such as a TensorFlow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., "Lake Crest") processor from Intel Corp. In at least one embodiment, inference and/or training logic 815 illustrated in FIG. 8A may be used in conjunction with central processing unit ("CPU") hardware, graphics processing unit  
35 ("GPU") hardware or other hardware, such as field programmable gate arrays ("FPGAs").

**[0106]** FIG. 8B illustrates inference and/or training logic 815, according to at least one embodiment. In at least one embodiment, inference and/or training logic 815 may include,

5 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 815 illustrated in FIG. 8B may be used in conjunction with an application-specific integrated circuit (ASIC), such as TensorFlow®  
10 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 815 illustrated in FIG. 8B 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,  
15 inference and/or training logic 815 includes, without limitation, code and/or data storage 801 and code and/or data storage 805, which may be used to store code (e.g., graph code), weight values and/or other information, including bias values, gradient information, momentum values, and/or other parameter or hyperparameter information. In at least one embodiment illustrated in FIG. 8B, each of code and/or data storage 801 and code and/or data storage 805  
20 is associated with a dedicated computational resource, such as computational hardware 802 and computational hardware 806, respectively. In at least one embodiment, each of computational hardware 802 and computational hardware 806 comprises one or more ALUs that perform mathematical functions, such as linear algebraic functions, only on information stored in code and/or data storage 801 and code and/or data storage 805, respectively, result  
25 of which is stored in activation storage 820.

[0107] In at least one embodiment, each of code and/or data storage 801 and 805 and corresponding computational hardware 802 and 806, respectively, correspond to different layers of a neural network, such that resulting activation from one storage/computational pair 801/802 of code and/or data storage 801 and computational hardware 802 is provided as an  
30 input to a next storage/computational pair 805/806 of code and/or data storage 805 and computational hardware 806, in order to mirror a conceptual organization of a neural network. In at least one embodiment, each of storage/computational pairs 801/802 and 805/806 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  
35 storage/computation pairs 801/802 and 805/806 may be included in inference and/or training logic 815.

5 [0108] FIG. 7A illustrates an example of an autonomous vehicle 700, according to at least one embodiment. In at least one embodiment, autonomous vehicle 700 (alternatively referred to herein as “vehicle 700”) may be, without limitation, a passenger vehicle, such as a car, a truck, a bus, and/or another type of vehicle that accommodates one or more passengers. In at least one embodiment, vehicle 700 may be a semi-tractor-trailer truck used for hauling  
10 cargo. In at least one embodiment, vehicle 700 may be an airplane, robotic vehicle, or other kind of vehicle.

[0109] Autonomous vehicles may be described in terms of automation levels, defined by National Highway Traffic Safety Administration (“NHTSA”), a division of US Department of Transportation, and Society of Automotive Engineers (“SAE”) “Taxonomy and  
15 Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles” (e.g., Standard No. J3016-201806, published on June 15, 2018, Standard No. J3016-201609, published on September 30, 2016, and previous and future versions of this standard). In at least one embodiment, vehicle 700 may be capable of functionality in accordance with one or more of Level 1 through Level 5 of autonomous driving levels. For example, in at least one  
20 embodiment, vehicle 700 may be capable of conditional automation (Level 3), high automation (Level 4), and/or full automation (Level 5), depending on embodiment.

[0110] In at least one embodiment, vehicle 700 may include, without limitation, components such as a chassis, a vehicle body, wheels (e.g., 2, 4, 6, 8, 18, etc.), tires, axles, and other components of a vehicle. In at least one embodiment, vehicle 700 may include, without  
25 limitation, a propulsion system 750, such as an internal combustion engine, hybrid electric power plant, an all-electric engine, and/or another propulsion system type. In at least one embodiment, propulsion system 750 may be connected to a drive train of vehicle 700, which may include, without limitation, a transmission, to enable propulsion of vehicle 700. In at least one embodiment, propulsion system 750 may be controlled in response to receiving  
30 signals from a throttle/accelerator(s) 752.

[0111] In at least one embodiment, a steering system 754, which may include, without limitation, a steering wheel, is used to steer vehicle 700 (e.g., along a desired path or route) when propulsion system 750 is operating (e.g., when vehicle 700 is in motion). In at least one embodiment, steering system 754 may receive signals from steering actuator(s) 756. In  
35 at least one embodiment, a steering wheel may be optional for full automation (Level 5) functionality. In at least one embodiment, a brake sensor system 746 may be used to operate

5 vehicle brakes in response to receiving signals from brake actuator(s) 748 and/or brake sensors.

[0112] In at least one embodiment, controller(s) 736, which may include, without limitation, one or more system on chips (“SoCs”) (not shown in FIG. 7A) and/or graphics processing unit(s) (“GPU(s)”), provide signals (e.g., representative of commands) to one or more  
10 components and/or systems of vehicle 700. For instance, in at least one embodiment, controller(s) 736 may send signals to operate vehicle brakes via brake actuator(s) 748, to operate steering system 754 via steering actuator(s) 756, to operate propulsion system 750 via throttle/accelerator(s) 752. In at least one embodiment, controller(s) 736 may include one or more onboard (e.g., integrated) computing devices that process sensor signals, and output  
15 operation commands (e.g., signals representing commands) to enable autonomous driving and/or to assist a human driver in driving vehicle 700. In at least one embodiment, controller(s) 736 may include a first controller for autonomous driving functions, a second controller for functional safety functions, a third controller for artificial intelligence functionality (e.g., computer vision), a fourth controller for infotainment functionality, a fifth  
20 controller for redundancy in emergency conditions, and/or other controllers. In at least one embodiment, a single controller may handle two or more of above functionalities, two or more controllers may handle a single functionality, and/or any combination thereof.

[0113] In at least one embodiment, controller(s) 736 provide signals for controlling one or more components and/or systems of vehicle 700 in response to sensor data received from one  
25 or more sensors (e.g., sensor inputs). In at least one embodiment, sensor data may be received from, for example and without limitation, global navigation satellite systems (“GNSS”) sensor(s) 758 (e.g., Global Positioning System sensor(s)), RADAR sensor(s) 760, ultrasonic sensor(s) 762, LIDAR sensor(s) 764, inertial measurement unit (“IMU”) sensor(s) 766 (e.g., accelerometer(s), gyroscope(s), a magnetic compass or magnetic compasses,  
30 magnetometer(s), etc.), microphone(s) 796, stereo camera(s) 768, wide-view camera(s) 770 (e.g., fisheye cameras), infrared camera(s) 772, surround camera(s) 774 (e.g., 360 degree cameras), long-range cameras (not shown in FIG. 7A), mid-range camera(s) (not shown in FIG. 7A), speed sensor(s) 744 (e.g., for measuring speed of vehicle 700), vibration sensor(s) 742, steering sensor(s) 740, brake sensor(s) (e.g., as part of brake sensor system 746), and/or  
35 other sensor types.

[0114] In at least one embodiment, one or more of controller(s) 736 may receive inputs (e.g., represented by input data) from an instrument cluster 732 of vehicle 700 and provide outputs

5 (e.g., represented by output data, display data, etc.) via a human-machine interface (“HMI”) display 734, an audible annunciator, a loudspeaker, and/or via other components of vehicle 700. In at least one embodiment, outputs may include information such as vehicle velocity, speed, time, map data (e.g., a High Definition map (not shown in FIG. 7A), location data (e.g., vehicle’s 700 location, such as on a map), direction, location of other vehicles (e.g., an  
10 occupancy grid), information about objects and status of objects as perceived by controller(s) 736, etc. For example, in at least one embodiment, HMI display 734 may display information about presence of one or more objects (e.g., a street sign, caution sign, traffic light changing, etc.), and/or information about driving maneuvers vehicle has made, is making, or will make (e.g., changing lanes now, taking exit 34B in two miles, etc.).

15 [0115] In at least one embodiment, vehicle 700 further includes a network interface 724 which may use wireless antenna(s) 726 and/or modem(s) to communicate over one or more networks. For example, in at least one embodiment, network interface 724 may be capable of communication over Long-Term Evolution (“LTE”), Wideband Code Division Multiple Access (“WCDMA”), Universal Mobile Telecommunications System (“UMTS”), Global  
20 System for Mobile communication (“GSM”), IMT-CDMA Multi-Carrier (“CDMA2000”) networks, etc. In at least one embodiment, wireless antenna(s) 726 may also enable communication between objects in environment (e.g., vehicles, mobile devices, etc.), using local area network(s), such as Bluetooth, Bluetooth Low Energy (“LE”), Z-Wave, ZigBee, etc., and/or low power wide-area network(s) (“LPWANs”), such as LoRaWAN, SigFox, etc.  
25 protocols.

[0116] Inference and/or training logic 815 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 815 are provided herein in conjunction with FIGS. 8A and/or 8B. In at least one embodiment, inference and/or training logic 815 may be used in system FIG. 7A for  
30 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.

[0117] In at least one embodiment, techniques described in connection with FIGS. 1-6 are implemented in context of vehicle 700. In at least one embodiment, vehicle 700 is  
35 implemented in context of a processor comprising: one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at

5 least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images

[0118] FIG. 7B illustrates an example of camera locations and fields of view for autonomous vehicle 700 of FIG. 7A, according to at least one embodiment. In at least one embodiment, cameras and respective fields of view are one example embodiment and are not intended to  
10 be limiting. For instance, in at least one embodiment, additional and/or alternative cameras may be included and/or cameras may be located at different locations on vehicle 700.

[0119] In at least one embodiment, camera types for cameras may include, but are not limited to, digital cameras that may be adapted for use with components and/or systems of vehicle 700. In at least one embodiment, camera(s) may operate at automotive safety integrity level  
15 (“ASIL”) B and/or at another ASIL. In at least one embodiment, camera types may be capable of any image capture rate, such as 60 frames per second (fps), 1220 fps, 240 fps, etc., depending on embodiment. In at least one embodiment, cameras may be capable of using rolling shutters, global shutters, another type of shutter, or a combination thereof. In at least one embodiment, color filter array may include a red clear clear clear (“RCCC”) color filter  
20 array, a red clear clear blue (“RCCB”) color filter array, a red blue green clear (“RBGC”) color filter array, a Foveon X3 color filter array, a Bayer sensors (“RGGB”) color filter array, a monochrome sensor color filter array, and/or another type of color filter array. In at least one embodiment, clear pixel cameras, such as cameras with an RCCC, an RCCB, and/or an RBGC color filter array, may be used in an effort to increase light sensitivity.

25 [0120] In at least one embodiment, one or more of camera(s) may be used to perform advanced driver assistance systems (“ADAS”) functions (e.g., as part of a redundant or fail-safe design). For example, in at least one embodiment, a Multi-Function Mono Camera may be installed to provide functions including lane departure warning, traffic sign assist and intelligent headlamp control. In at least one embodiment, one or more of camera(s) (e.g., all  
30 cameras) may record and provide image data (e.g., video) simultaneously.

[0121] In at least one embodiment, one or more camera may be mounted in a mounting assembly, such as a custom designed (three-dimensional (“3D”) printed) assembly, in order to cut out stray light and reflections from within vehicle 700 (e.g., reflections from dashboard reflected in windshield mirrors) which may interfere with camera image data capture  
35 abilities. With reference to wing-mirror mounting assemblies, in at least one embodiment, wing-mirror assemblies may be custom 3D printed so that a camera mounting plate matches a

5 shape of a wing-mirror. In at least one embodiment, camera(s) may be integrated into wing-mirrors. In at least one embodiment, for side-view cameras, camera(s) may also be integrated within four pillars at each corner of a cabin.

[0122] In at least one embodiment, cameras with a field of view that include portions of an environment in front of vehicle 700 (e.g., front-facing cameras) may be used for surround  
10 view, to help identify forward facing paths and obstacles, as well as aid in, with help of one or more of controller(s) 736 and/or control SoCs, providing information critical to generating an occupancy grid and/or determining preferred vehicle paths. In at least one embodiment, front-facing cameras may be used to perform many similar ADAS functions as LIDAR, including, without limitation, emergency braking, pedestrian detection, and collision  
15 avoidance. In at least one embodiment, front-facing cameras may also be used for ADAS functions and systems including, without limitation, Lane Departure Warnings (“LDW”), Autonomous Cruise Control (“ACC”), and/or other functions such as traffic sign recognition.

[0123] In at least one embodiment, a variety of cameras may be used in a front-facing configuration, including, for example, a monocular camera platform that includes a CMOS  
20 (“complementary metal oxide semiconductor”) color imager. In at least one embodiment, a wide-view camera 770 may be used to perceive objects coming into view from a periphery (e.g., pedestrians, crossing traffic or bicycles). Although only one wide-view camera 770 is illustrated in FIG. 7B, in other embodiments, there may be any number (including zero) wide-view cameras on vehicle 700. In at least one embodiment, any number of long-range  
25 camera(s) 798 (e.g., a long-view stereo camera pair) may be used for depth-based object detection, especially for objects for which a neural network has not yet been trained. In at least one embodiment, long-range camera(s) 798 may also be used for object detection and classification, as well as basic object tracking.

[0124] In at least one embodiment, any number of stereo camera(s) 768 may also be included  
30 in a front-facing configuration. In at least one embodiment, one or more of stereo camera(s) 768 may include an integrated control unit comprising a scalable processing unit, which may provide a programmable logic (“FPGA”) and a multi-core micro-processor with an integrated Controller Area Network (“CAN”) or Ethernet interface on a single chip. In at least one embodiment, such a unit may be used to generate a 3D map of an environment of vehicle  
35 700, including a distance estimate for all points in an image. In at least one embodiment, one or more of stereo camera(s) 768 may include, without limitation, compact stereo vision sensor(s) that may include, without limitation, two camera lenses (one each on left and right)

5 and an image processing chip that may measure distance from vehicle 700 to target object and use generated information (e.g., metadata) to activate autonomous emergency braking and lane departure warning functions. In at least one embodiment, other types of stereo camera(s) 768 may be used in addition to, or alternatively from, those described herein.

[0125] In at least one embodiment, cameras with a field of view that include portions of  
10 environment to sides of vehicle 700 (e.g., side-view cameras) may be used for surround view, providing information used to create and update an occupancy grid, as well as to generate side impact collision warnings. For example, in at least one embodiment, surround camera(s) 774 (e.g., four surround cameras as illustrated in FIG. 7B) could be positioned on vehicle 700. In at least one embodiment, surround camera(s) 774 may include, without  
15 limitation, any number and combination of wide-view cameras, fisheye camera(s), 360 degree camera(s), and/or similar cameras. For instance, in at least one embodiment, four fisheye cameras may be positioned on a front, a rear, and sides of vehicle 700. In at least one embodiment, vehicle 700 may use three surround camera(s) 774 (e.g., left, right, and rear), and may leverage one or more other camera(s) (e.g., a forward-facing camera) as a  
20 fourth surround-view camera.

[0126] In at least one embodiment, cameras with a field of view that include portions of an environment behind vehicle 700 (e.g., rear-view cameras) may be used for parking assistance, surround view, rear collision warnings, and creating and updating an occupancy grid. In at least one embodiment, a wide variety of cameras may be used including, but not limited to,  
25 cameras that are also suitable as a front-facing camera(s) (e.g., long-range cameras 798 and/or mid-range camera(s) 776, stereo camera(s) 768, infrared camera(s) 772, etc.), as described herein.

[0127] Inference and/or training logic 815 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or  
30 training logic 815 are provided herein in conjunction with FIGS. 8A and/or 8B. In at least one embodiment, inference and/or training logic 815 may be used in system FIG. 7B 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.

35 [0128] In at least one embodiment, techniques described in connection with FIGS. 1-6 are implemented in context of autonomous vehicle 700 in accordance with FIG. 7B. In at least

5 one embodiment, vehicle 700 is implemented in context of a processor comprising: one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images

[0129] FIG. 7C is a block diagram illustrating an example system architecture for  
10 autonomous vehicle 700 of FIG. 7A, according to at least one embodiment. In at least one embodiment, each of components, features, and systems of vehicle 700 in FIG. 7C is illustrated as being connected via a bus 702. In at least one embodiment, bus 702 may include, without limitation, a CAN data interface (alternatively referred to herein as a “CAN bus”). In at least one embodiment, a CAN may be a network inside vehicle 700 used to aid  
15 in control of various features and functionality of vehicle 700, such as actuation of brakes, acceleration, braking, steering, windshield wipers, etc. In at least one embodiment, bus 702 may be configured to have dozens or even hundreds of nodes, each with its own unique identifier (e.g., a CAN ID). In at least one embodiment, bus 702 may be read to find steering wheel angle, ground speed, engine revolutions per minute (“RPMs”), button positions, and/or  
20 other vehicle status indicators. In at least one embodiment, bus 702 may be a CAN bus that is ASIL B compliant.

[0130] In at least one embodiment, in addition to, or alternatively from CAN, FlexRay and/or Ethernet protocols may be used. In at least one embodiment, there may be any number of busses forming bus 702, which may include, without limitation, zero or more CAN busses,  
25 zero or more FlexRay busses, zero or more Ethernet busses, and/or zero or more other types of busses using different protocols. In at least one embodiment, two or more busses may be used to perform different functions, and/or may be used for redundancy. For example, a first bus may be used for collision avoidance functionality and a second bus may be used for actuation control. In at least one embodiment, each bus of bus 702 may communicate with  
30 any of components of vehicle 700, and two or more busses of bus 702 may communicate with corresponding components. In at least one embodiment, each of any number of system(s) on chip(s) (“SoC(s)”) 704 (such as SoC 704(A) and SoC 704(B), each of controller(s) 736, and/or each computer within vehicle may have access to same input data (e.g., inputs from sensors of vehicle 700), and may be connected to a common bus, such  
35 CAN bus.

[0131] In at least one embodiment, vehicle 700 may include one or more controller(s) 736, such as those described herein with respect to FIG. 7A. In at least one embodiment,

5 controller(s) 736 may be used for a variety of functions. In at least one embodiment, controller(s) 736 may be coupled to any of various other components and systems of vehicle 700, and may be used for control of vehicle 700, artificial intelligence of vehicle 700, infotainment for vehicle 700, and/or other functions.

[0132] In at least one embodiment, vehicle 700 may include any number of SoCs 704. In at  
10 least one embodiment, each of SoCs 704 may include, without limitation, central processing units (“CPU(s)”) 706, graphics processing units (“GPU(s)”) 708, processor(s) 710, cache(s) 712, accelerator(s) 714, data store(s) 716, and/or other components and features not illustrated. In at least one embodiment, SoC(s) 704 may be used to control vehicle 700 in a variety of platforms and systems. For example, in at least one embodiment, SoC(s) 704 may  
15 be combined in a system (e.g., system of vehicle 700) with a High Definition (“HD”) map 722 which may obtain map refreshes and/or updates via network interface 724 from one or more servers (not shown in FIG. 7C).

[0133] In at least one embodiment, CPU(s) 706 may include a CPU cluster or CPU complex (alternatively referred to herein as a “CCPLEX”). In at least one embodiment, CPU(s) 706  
20 may include multiple cores and/or level two (“L2”) caches. For instance, in at least one embodiment, CPU(s) 706 may include eight cores in a coherent multi-processor configuration. In at least one embodiment, CPU(s) 706 may include four dual-core clusters where each cluster has a dedicated L2 cache (e.g., a 2 megabyte (MB) L2 cache). In at least one embodiment, CPU(s) 706 (e.g., CCPLEX) may be configured to support simultaneous  
25 cluster operations enabling any combination of clusters of CPU(s) 706 to be active at any given time.

[0134] In at least one embodiment, one or more of CPU(s) 706 may implement power management capabilities that include, without limitation, one or more of following features: individual hardware blocks may be clock-gated automatically when idle to save dynamic  
30 power; each core clock may be gated when such core is not actively executing instructions due to execution of Wait for Interrupt (“WFI”)/Wait for Event (“WFE”) instructions; each core may be independently power-gated; each core cluster may be independently clock-gated when all cores are clock-gated or power-gated; and/or each core cluster may be independently power-gated when all cores are power-gated. In at least one embodiment,  
35 CPU(s) 706 may further implement an enhanced algorithm for managing power states, where allowed power states and expected wakeup times are specified, and hardware/microcode determines which best power state to enter for core, cluster, and CCPLEX. In at least one

5 embodiment, processing cores may support simplified power state entry sequences in software with work offloaded to microcode.

[0135] In at least one embodiment, GPU(s) 708 may include an integrated GPU (alternatively referred to herein as an “iGPU”). In at least one embodiment, GPU(s) 708 may be programmable and may be efficient for parallel workloads. In at least one embodiment, GPU(s) 708 may use an enhanced tensor instruction set. In at least one embodiment, GPU(s) 708 may include one or more streaming microprocessors, where each streaming microprocessor may include a level one (“L1”) cache (e.g., an L1 cache with at least 96 KB storage capacity), and two or more streaming microprocessors may share an L2 cache (e.g., an L2 cache with a 512 KB storage capacity). In at least one embodiment, GPU(s) 708 may include at least eight streaming microprocessors. In at least one embodiment, GPU(s) 708 may use compute application programming interface(s) (API(s)). In at least one embodiment, GPU(s) 708 may use one or more parallel computing platforms and/or programming models (e.g., NVIDIA’s CUDA model).

[0136] In at least one embodiment, one or more of GPU(s) 708 may be power-optimized for best performance in automotive and embedded use cases. For example, in at least one embodiment, GPU(s) 708 could be fabricated on Fin field-effect transistor (“FinFET”) circuitry. In at least one embodiment, each streaming microprocessor may incorporate a number of mixed-precision processing cores partitioned into multiple blocks. For example, and without limitation, 64 PF32 cores and 32 PF64 cores could be partitioned into four processing blocks. In at least one embodiment, each processing block could be allocated 16 FP32 cores, 8 FP64 cores, 16 INT32 cores, two mixed-precision NVIDIA Tensor cores for deep learning matrix arithmetic, a level zero (“L0”) instruction cache, a warp scheduler, a dispatch unit, and/or a 64 KB register file. In at least one embodiment, streaming microprocessors may include independent parallel integer and floating-point data paths to provide for efficient execution of workloads with a mix of computation and addressing calculations. In at least one embodiment, streaming microprocessors may include independent thread scheduling capability to enable finer-grain synchronization and cooperation between parallel threads. In at least one embodiment, streaming microprocessors may include a combined L1 data cache and shared memory unit in order to improve performance while simplifying programming.

[0137] In at least one embodiment, one or more of GPU(s) 708 may include a high bandwidth memory (“HBM) and/or a 16 GB HBM2 memory subsystem to provide, in some

5 examples, about 900 GB/second peak memory bandwidth. In at least one embodiment, in addition to, or alternatively from, HBM memory, a synchronous graphics random-access memory (“SGRAM”) may be used, such as a graphics double data rate type five synchronous random-access memory (“GDDR5”).

[0138] In at least one embodiment, GPU(s) 708 may include unified memory technology. In  
10 at least one embodiment, address translation services (“ATS”) support may be used to allow GPU(s) 708 to access CPU(s) 706 page tables directly. In at least one embodiment, embodiment, when a GPU of GPU(s) 708 memory management unit (“MMU”) experiences a miss, an address translation request may be transmitted to CPU(s) 706. In response, 2 CPU of CPU(s) 706 may look in its page tables for a virtual-to-physical mapping for an address  
15 and transmit translation back to GPU(s) 708, in at least one embodiment. In at least one embodiment, unified memory technology may allow a single unified virtual address space for memory of both CPU(s) 706 and GPU(s) 708, thereby simplifying GPU(s) 708 programming and porting of applications to GPU(s) 708.

[0139] In at least one embodiment, GPU(s) 708 may include any number of access counters  
20 that may keep track of frequency of access of GPU(s) 708 to memory of other processors. In at least one embodiment, access counter(s) may help ensure that memory pages are moved to physical memory of a processor that is accessing pages most frequently, thereby improving efficiency for memory ranges shared between processors.

[0140] In at least one embodiment, one or more of SoC(s) 704 may include any number of  
25 cache(s) 712, including those described herein. For example, in at least one embodiment, cache(s) 712 could include a level three (“L3”) cache that is available to both CPU(s) 706 and GPU(s) 708 (e.g., that is connected to CPU(s) 706 and GPU(s) 708). In at least one embodiment, cache(s) 712 may include a write-back cache that may keep track of states of lines, such as by using a cache coherence protocol (e.g., MEI, MESI, MSI, etc.). In at least  
30 one embodiment, a L3 cache may include 4 MB of memory or more, depending on embodiment, although smaller cache sizes may be used.

[0141] In at least one embodiment, one or more of SoC(s) 704 may include one or more  
accelerator(s) 714 (e.g., hardware accelerators, software accelerators, or a combination thereof). In at least one embodiment, SoC(s) 704 may include a hardware acceleration  
35 cluster that may include optimized hardware accelerators and/or large on-chip memory. In at least one embodiment, large on-chip memory (e.g., 4 MB of SRAM), may enable a

5 hardware acceleration cluster to accelerate neural networks and other calculations. In at least one embodiment, a hardware acceleration cluster may be used to complement GPU(s) 708 and to off-load some of tasks of GPU(s) 708 (e.g., to free up more cycles of GPU(s) 708 for performing other tasks). In at least one embodiment, accelerator(s) 714 could be used for targeted workloads (e.g., perception, convolutional neural networks (“CNNs”), recurrent  
10 neural networks (“RNNs”), etc.) that are stable enough to be amenable to acceleration. In at least one embodiment, a CNN may include a region-based or regional convolutional neural networks (“RCNNs”) and Fast RCNNs (e.g., as used for object detection) or other type of CNN.

[0142] In at least one embodiment, accelerator(s) 714 (e.g., hardware acceleration cluster)  
15 may include one or more deep learning accelerator (“DLA”). In at least one embodiment, DLA(s) may include, without limitation, one or more Tensor processing units (“TPUs”) that may be configured to provide an additional ten trillion operations per second for deep learning applications and inferencing. In at least one embodiment, TPUs may be accelerators configured to, and optimized for, performing image processing functions (e.g.,  
20 for CNNs, RCNNs, etc.). In at least one embodiment, DLA(s) may further be optimized for a specific set of neural network types and floating point operations, as well as inferencing. In at least one embodiment, design of DLA(s) may provide more performance per millimeter than a typical general-purpose GPU, and typically vastly exceeds performance of a CPU. In at least one embodiment, TPU(s) may perform several functions, including a single-instance  
25 convolution function, supporting, for example, INT8, INT16, and FP16 data types for both features and weights, as well as post-processor functions. In at least one embodiment, DLA(s) may quickly and efficiently execute neural networks, especially CNNs, on processed or unprocessed data for any of a variety of functions, including, for example and without limitation: a CNN for object identification and detection using data from camera sensors; a  
30 CNN for distance estimation using data from camera sensors; a CNN for emergency vehicle detection and identification and detection using data from microphones; a CNN for facial recognition and vehicle owner identification using data from camera sensors; and/or a CNN for security and/or safety related events.

[0143] In at least one embodiment, DLA(s) may perform any function of GPU(s) 708, and by  
35 using an inference accelerator, for example, a designer may target either DLA(s) or GPU(s) 708 for any function. For example, in at least one embodiment, a designer may focus

5 processing of CNNs and floating point operations on DLA(s) and leave other functions to GPU(s) 708 and/or accelerator(s) 714.

[0144] In at least one embodiment, accelerator(s) 714 may include programmable vision accelerator (“PVA”), which may alternatively be referred to herein as a computer vision accelerator. In at least one embodiment, PVA may be designed and configured to accelerate  
10 computer vision algorithms for advanced driver assistance system (“ADAS”) 738, autonomous driving, augmented reality (“AR”) applications, and/or virtual reality (“VR”) applications. In at least one embodiment, PVA may provide a balance between performance and flexibility. For example, in at least one embodiment, each PVA may include, for example and without limitation, any number of reduced instruction set computer (“RISC”)  
15 cores, direct memory access (“DMA”), and/or any number of vector processors.

[0145] In at least one embodiment, RISC cores may interact with image sensors (e.g., image sensors of any cameras described herein), image signal processor(s), etc. In at least one embodiment, each RISC core may include any amount of memory. In at least one embodiment, RISC cores may use any of a number of protocols, depending on embodiment.  
20 In at least one embodiment, RISC cores may execute a real-time operating system (“RTOS”). In at least one embodiment, RISC cores may be implemented using one or more integrated circuit devices, application specific integrated circuits (“ASICs”), and/or memory devices. For example, in at least one embodiment, RISC cores could include an instruction cache and/or a tightly coupled RAM.

[0146] In at least one embodiment, DMA may enable components of PVA to access system memory independently of CPU(s) 706. In at least one embodiment, DMA may support any number of features used to provide optimization to a PVA including, but not limited to, supporting multi-dimensional addressing and/or circular addressing. In at least one embodiment, DMA may support up to six or more dimensions of addressing, which may  
30 include, without limitation, block width, block height, block depth, horizontal block stepping, vertical block stepping, and/or depth stepping.

[0147] In at least one embodiment, vector processors may be programmable processors that may be designed to efficiently and flexibly execute programming for computer vision algorithms and provide signal processing capabilities. In at least one embodiment, a PVA  
35 may include a PVA core and two vector processing subsystem partitions. In at least one embodiment, a PVA core may include a processor subsystem, DMA engine(s) (e.g., two

5 DMA engines), and/or other peripherals. In at least one embodiment, a vector processing subsystem may operate as a primary processing engine of a PVA, and may include a vector processing unit (“VPU”), an instruction cache, and/or vector memory (e.g., “VMEM”). In at least one embodiment, VPU core may include a digital signal processor such as, for example, a single instruction, multiple data (“SIMD”), very long instruction word (“VLIW”) digital  
10 signal processor. In at least one embodiment, a combination of SIMD and VLIW may enhance throughput and speed.

[0148] In at least one embodiment, each of vector processors may include an instruction cache and may be coupled to dedicated memory. As a result, in at least one embodiment, each of vector processors may be configured to execute independently of other vector  
15 processors. In at least one embodiment, vector processors that are included in a particular PVA may be configured to employ data parallelism. For instance, in at least one embodiment, plurality of vector processors included in a single PVA may execute a common computer vision algorithm, but on different regions of an image. In at least one embodiment, vector processors included in a particular PVA may simultaneously execute  
20 different computer vision algorithms, on one image, or even execute different algorithms on sequential images or portions of an image. In at least one embodiment, among other things, any number of PVAs may be included in hardware acceleration cluster and any number of vector processors may be included in each PVA. In at least one embodiment, PVA may include additional error correcting code (“ECC”) memory, to enhance overall system safety.

25 [0149] In at least one embodiment, accelerator(s) 714 may include a computer vision network on-chip and static random-access memory (“SRAM”), for providing a high-bandwidth, low latency SRAM for accelerator(s) 714. In at least one embodiment, on-chip memory may include at least 4 MB SRAM, comprising, for example and without limitation, eight field-configurable memory blocks, that may be accessible by both a PVA and a DLA.  
30 In at least one embodiment, each pair of memory blocks may include an advanced peripheral bus (“APB”) interface, configuration circuitry, a controller, and a multiplexer. In at least one embodiment, any type of memory may be used. In at least one embodiment, a PVA and a DLA may access memory via a backbone that provides a PVA and a DLA with high-speed access to memory. In at least one embodiment, a backbone may include a computer vision  
35 network on-chip that interconnects a PVA and a DLA to memory (e.g., using APB).

[0150] In at least one embodiment, a computer vision network on-chip may include an interface that determines, before transmission of any control signal/address/data, that both a

5 PVA and a DLA provide ready and valid signals. In at least one embodiment, an interface may provide for separate phases and separate channels for transmitting control signals/addresses/data, as well as burst-type communications for continuous data transfer. In at least one embodiment, an interface may comply with International Organization for Standardization (“ISO”) 26262 or International Electrotechnical Commission (“IEC”) 61508  
10 standards, although other standards and protocols may be used.

[0151] In at least one embodiment, one or more of SoC(s) 704 may include a real-time ray-tracing hardware accelerator. In at least one embodiment, real-time ray-tracing hardware accelerator may be used to quickly and efficiently determine positions and extents of objects (e.g., within a world model), to generate real-time visualization simulations, for RADAR  
15 signal interpretation, for sound propagation synthesis and/or analysis, for simulation of SONAR systems, for general wave propagation simulation, for comparison to LIDAR data for purposes of localization and/or other functions, and/or for other uses.

[0152] In at least one embodiment, accelerator(s) 714 can have a wide array of uses for autonomous driving. In at least one embodiment, a PVA may be used for key processing  
20 stages in ADAS and autonomous vehicles. In at least one embodiment, a PVA’s capabilities are a good match for algorithmic domains needing predictable processing, at low power and low latency. In other words, a PVA performs well on semi-dense or dense regular computation, even on small data sets, which might require predictable run-times with low latency and low power. In at least one embodiment, such as in vehicle 700, PVAs might be  
25 designed to run classic computer vision algorithms, as they can be efficient at object detection and operating on integer math.

[0153] For example, according to at least one embodiment of technology, a PVA is used to perform computer stereo vision. In at least one embodiment, a semi-global matching-based algorithm may be used in some examples, although this is not intended to be limiting. In at  
30 least one embodiment, applications for Level 3-5 autonomous driving use motion estimation/stereo matching on-the-fly (e.g., structure from motion, pedestrian recognition, lane detection, etc.). In at least one embodiment, a PVA may perform computer stereo vision functions on inputs from two monocular cameras.

[0154] In at least one embodiment, a PVA may be used to perform dense optical flow. For  
35 example, in at least one embodiment, a PVA could process raw RADAR data (e.g., using a 4D Fast Fourier Transform) to provide processed RADAR data. In at least one embodiment,

5 a PVA is used for time of flight depth processing, by processing raw time of flight data to provide processed time of flight data, for example.

[0155] In at least one embodiment, a DLA may be used to run any type of network to enhance control and driving safety, including for example and without limitation, a neural network that outputs a measure of confidence for each object detection. In at least one  
10 embodiment, confidence may be represented or interpreted as a probability, or as providing a relative “weight” of each detection compared to other detections. In at least one embodiment, a confidence measure enables a system to make further decisions regarding which detections should be considered as true positive detections rather than false positive detections. In at least one embodiment, a system may set a threshold value for confidence  
15 and consider only detections exceeding threshold value as true positive detections. In an embodiment in which an automatic emergency braking (“AEB”) system is used, false positive detections would cause vehicle to automatically perform emergency braking, which is obviously undesirable. In at least one embodiment, highly confident detections may be considered as triggers for AEB. In at least one embodiment, a DLA may run a neural  
20 network for regressing confidence value. In at least one embodiment, neural network may take as its input at least some subset of parameters, such as bounding box dimensions, ground plane estimate obtained (e.g., from another subsystem), output from IMU sensor(s) 766 that correlates with vehicle 700 orientation, distance, 3D location estimates of object obtained from neural network and/or other sensors (e.g., LIDAR sensor(s) 764 or RADAR sensor(s)  
25 760), among others.

[0156] In at least one embodiment, one or more of SoC(s) 704 may include data store(s) 716 (e.g., memory). In at least one embodiment, data store(s) 716 may be on-chip memory of SoC(s) 704, which may store neural networks to be executed on GPU(s) 708 and/or a DLA. In at least one embodiment, data store(s) 716 may be large enough in capacity to store  
30 multiple instances of neural networks for redundancy and safety. In at least one embodiment, data store(s) 716 may comprise L2 or L3 cache(s).

[0157] In at least one embodiment, one or more of SoC(s) 704 may include any number of processor(s) 710 (e.g., embedded processors). In at least one embodiment, processor(s) 710 may include a boot and power management processor that may be a dedicated processor and  
35 subsystem to handle boot power and management functions and related security enforcement. In at least one embodiment, a boot and power management processor may be a part of a boot sequence of SoC(s) 704 and may provide runtime power management services. In at least

5 one embodiment, a boot power and management processor may provide clock and voltage programming, assistance in system low power state transitions, management of SoC(s) 704 thermals and temperature sensors, and/or management of SoC(s) 704 power states. In at least one embodiment, each temperature sensor may be implemented as a ring-oscillator whose output frequency is proportional to temperature, and SoC(s) 704 may use ring-  
10 oscillators to detect temperatures of CPU(s) 706, GPU(s) 708, and/or accelerator(s) 714. In at least one embodiment, if temperatures are determined to exceed a threshold, then a boot and power management processor may enter a temperature fault routine and put SoC(s) 704 into a lower power state and/or put vehicle 700 into a chauffeur to safe stop mode (e.g., bring vehicle 700 to a safe stop).

15 **[0158]** In at least one embodiment, processor(s) 710 may further include a set of embedded processors that may serve as an audio processing engine which may be an audio subsystem that enables full hardware support for multi-channel audio over multiple interfaces, and a broad and flexible range of audio I/O interfaces. In at least one embodiment, an audio processing engine is a dedicated processor core with a digital signal processor with dedicated  
20 RAM.

**[0159]** In at least one embodiment, processor(s) 710 may further include an always-on processor engine that may provide necessary hardware features to support low power sensor management and wake use cases. In at least one embodiment, an always-on processor engine may include, without limitation, a processor core, a tightly coupled RAM, supporting  
25 peripherals (e.g., timers and interrupt controllers), various I/O controller peripherals, and routing logic.

**[0160]** In at least one embodiment, processor(s) 710 may further include a safety cluster engine that includes, without limitation, a dedicated processor subsystem to handle safety management for automotive applications. In at least one embodiment, a safety cluster  
30 engine may include, without limitation, two or more processor cores, a tightly coupled RAM, support peripherals (e.g., timers, an interrupt controller, etc.), and/or routing logic. In a safety mode, two or more cores may operate, in at least one embodiment, in a lockstep mode and function as a single core with comparison logic to detect any differences between their operations. In at least one embodiment, processor(s) 710 may further include a real-time  
35 camera engine that may include, without limitation, a dedicated processor subsystem for handling real-time camera management. In at least one embodiment, processor(s) 710 may further include a high-dynamic range signal processor that may include, without limitation,

5 an image signal processor that is a hardware engine that is part of a camera processing pipeline.

[0161] In at least one embodiment, processor(s) 710 may include a video image compositor that may be a processing block (e.g., implemented on a microprocessor) that implements video post-processing functions needed by a video playback application to produce a final  
10 image for a player window. In at least one embodiment, a video image compositor may perform lens distortion correction on wide-view camera(s) 770, surround camera(s) 774, and/or on in-cabin monitoring camera sensor(s). In at least one embodiment, in-cabin monitoring camera sensor(s) are preferably monitored by a neural network running on another instance of SoC 704, configured to identify in cabin events and respond accordingly.  
15 In at least one embodiment, an in-cabin system may perform, without limitation, lip reading to activate cellular service and place a phone call, dictate emails, change a vehicle's destination, activate or change a vehicle's infotainment system and settings, or provide voice-activated web surfing. In at least one embodiment, certain functions are available to a driver when a vehicle is operating in an autonomous mode and are disabled otherwise.  
20 [0162] In at least one embodiment, a video image compositor may include enhanced temporal noise reduction for both spatial and temporal noise reduction. For example, in at least one embodiment, where motion occurs in a video, noise reduction weights spatial information appropriately, decreasing weights of information provided by adjacent frames. In at least one embodiment, where an image or portion of an image does not include motion,  
25 temporal noise reduction performed by video image compositor may use information from a previous image to reduce noise in a current image.

[0163] In at least one embodiment, a video image compositor may also be configured to perform stereo rectification on input stereo lens frames. In at least one embodiment, a video image compositor may further be used for user interface composition when an operating  
30 system desktop is in use, and GPU(s) 708 are not required to continuously render new surfaces. In at least one embodiment, when GPU(s) 708 are powered on and active doing 3D rendering, a video image compositor may be used to offload GPU(s) 708 to improve performance and responsiveness.

[0164] In at least one embodiment, one or more SoC of SoC(s) 704 may further include a  
35 mobile industry processor interface ("MIPI") camera serial interface for receiving video and input from cameras, a high-speed interface, and/or a video input block that may be used for a

5 camera and related pixel input functions. In at least one embodiment, one or more of SoC(s) 704 may further include an input/output controller(s) that may be controlled by software and may be used for receiving I/O signals that are uncommitted to a specific role.

[0165] In at least one embodiment, one or more Soc of SoC(s) 704 may further include a broad range of peripheral interfaces to enable communication with peripherals, audio  
10 encoders/decoders (“codecs”), power management, and/or other devices. In at least one embodiment, SoC(s) 704 may be used to process data from cameras (e.g., connected over Gigabit Multimedia Serial Link and Ethernet channels), sensors (e.g., LIDAR sensor(s) 764, RADAR sensor(s) 760, etc. that may be connected over Ethernet channels), data from bus 702 (e.g., speed of vehicle 700, steering wheel position, etc.), data from GNSS sensor(s) 758  
15 (e.g., connected over a Ethernet bus or a CAN bus), etc. In at least one embodiment, one or more SoC of SoC(s) 704 may further include dedicated high-performance mass storage controllers that may include their own DMA engines, and that may be used to free CPU(s) 706 from routine data management tasks.

[0166] In at least one embodiment, SoC(s) 704 may be an end-to-end platform with a flexible  
20 architecture that spans automation Levels 3-5, thereby providing a comprehensive functional safety architecture that leverages and makes efficient use of computer vision and ADAS techniques for diversity and redundancy, and provides a platform for a flexible, reliable driving software stack, along with deep learning tools. In at least one embodiment, SoC(s) 704 may be faster, more reliable, and even more energy-efficient and space-efficient than  
25 conventional systems. For example, in at least one embodiment, accelerator(s) 714, when combined with CPU(s) 706, GPU(s) 708, and data store(s) 716, may provide for a fast, efficient platform for Level 3-5 autonomous vehicles.

[0167] In at least one embodiment, computer vision algorithms may be executed on CPUs, which may be configured using a high-level programming language, such as C, to execute a  
30 wide variety of processing algorithms across a wide variety of visual data. However, in at least one embodiment, CPUs are oftentimes unable to meet performance requirements of many computer vision applications, such as those related to execution time and power consumption, for example. In at least one embodiment, many CPUs are unable to execute complex object detection algorithms in real-time, which is used in in-vehicle ADAS  
35 applications and in practical Level 3-5 autonomous vehicles.

5 [0168] Embodiments described herein allow for multiple neural networks to be performed simultaneously and/or sequentially, and for results to be combined together to enable Level 3-5 autonomous driving functionality. For example, in at least one embodiment, a CNN executing on a DLA or a discrete GPU (e.g., GPU(s) 720) may include text and word recognition, allowing reading and understanding of traffic signs, including signs for which a  
10 neural network has not been specifically trained. In at least one embodiment, a DLA may further include a neural network that is able to identify, interpret, and provide semantic understanding of a sign, and to pass that semantic understanding to path planning modules running on a CPU Complex.

[0169] In at least one embodiment, multiple neural networks may be run simultaneously, as  
15 for Level 3, 4, or 5 driving. For example, in at least one embodiment, a warning sign stating “Caution: flashing lights indicate icy conditions,” along with an electric light, may be independently or collectively interpreted by several neural networks. In at least one embodiment, such warning sign itself may be identified as a traffic sign by a first deployed neural network (e.g., a neural network that has been trained), text “flashing lights indicate icy  
20 conditions” may be interpreted by a second deployed neural network, which informs a vehicle’s path planning software (preferably executing on a CPU Complex) that when flashing lights are detected, icy conditions exist. In at least one embodiment, a flashing light may be identified by operating a third deployed neural network over multiple frames, informing a vehicle’s path-planning software of a presence (or an absence) of flashing lights.  
25 In at least one embodiment, all three neural networks may run simultaneously, such as within a DLA and/or on GPU(s) 708.

[0170] In at least one embodiment, a CNN for facial recognition and vehicle owner identification may use data from camera sensors to identify presence of an authorized driver and/or owner of vehicle 700. In at least one embodiment, an always-on sensor processing  
30 engine may be used to unlock a vehicle when an owner approaches a driver door and turns on lights, and, in a security mode, to disable such vehicle when an owner leaves such vehicle. In this way, SoC(s) 704 provide for security against theft and/or carjacking.

[0171] In at least one embodiment, a CNN for emergency vehicle detection and identification may use data from microphones 796 to detect and identify emergency vehicle sirens. In at  
35 least one embodiment, SoC(s) 704 use a CNN for classifying environmental and urban sounds, as well as classifying visual data. In at least one embodiment, a CNN running on a DLA is trained to identify a relative closing speed of an emergency vehicle (e.g., by using a

5 Doppler effect). In at least one embodiment, a CNN may also be trained to identify emergency vehicles specific to a local area in which a vehicle is operating, as identified by GNSS sensor(s) 758. In at least one embodiment, when operating in Europe, a CNN will seek to detect European sirens, and when in North America, a CNN will seek to identify only North American sirens. In at least one embodiment, once an emergency vehicle is detected,  
10 a control program may be used to execute an emergency vehicle safety routine, slowing a vehicle, pulling over to a side of a road, parking a vehicle, and/or idling a vehicle, with assistance of ultrasonic sensor(s) 762, until emergency vehicles pass.

[0172] In at least one embodiment, vehicle 700 may include CPU(s) 718 (e.g., discrete CPU(s), or dCPU(s)), that may be coupled to SoC(s) 704 via a high-speed interconnect (e.g.,  
15 PCIe). In at least one embodiment, CPU(s) 718 may include an X86 processor, for example. CPU(s) 718 may be used to perform any of a variety of functions, including arbitrating potentially inconsistent results between ADAS sensors and SoC(s) 704, and/or monitoring status and health of controller(s) 736 and/or an infotainment system on a chip (“infotainment SoC”) 730, for example.

20 [0173] In at least one embodiment, vehicle 700 may include GPU(s) 720 (e.g., discrete GPU(s), or dGPU(s)), that may be coupled to SoC(s) 704 via a high-speed interconnect (e.g., NVIDIA’s NVLINK channel). In at least one embodiment, GPU(s) 720 may provide additional artificial intelligence functionality, such as by executing redundant and/or different neural networks, and may be used to train and/or update neural networks based at least in part  
25 on input (e.g., sensor data) from sensors of a vehicle 700.

[0174] In at least one embodiment, vehicle 700 may further include network interface 724 which may include, without limitation, wireless antenna(s) 726 (e.g., one or more wireless antennas for different communication protocols, such as a cellular antenna, a Bluetooth antenna, etc.). In at least one embodiment, network interface 724 may be used to enable  
30 wireless connectivity to Internet cloud services (e.g., with server(s) and/or other network devices), with other vehicles, and/or with computing devices (e.g., client devices of passengers). In at least one embodiment, to communicate with other vehicles, a direct link may be established between vehicle 70 and another vehicle and/or an indirect link may be established (e.g., across networks and over the Internet). In at least one embodiment, direct  
35 links may be provided using a vehicle-to-vehicle communication link. In at least one embodiment, a vehicle-to-vehicle communication link may provide vehicle 700 information about vehicles in proximity to vehicle 700 (e.g., vehicles in front of, on a side of, and/or

5 behind vehicle 700). In at least one embodiment, such aforementioned functionality may be part of a cooperative adaptive cruise control functionality of vehicle 700.

[0175] In at least one embodiment, network interface 724 may include an SoC that provides modulation and demodulation functionality and enables controller(s) 736 to communicate over wireless networks. In at least one embodiment, network interface 724 may include a  
10 radio frequency front-end for up-conversion from baseband to radio frequency, and down conversion from radio frequency to baseband. In at least one embodiment, frequency conversions may be performed in any technically feasible fashion. For example, frequency conversions could be performed through well-known processes, and/or using super-heterodyne processes. In at least one embodiment, radio frequency front end functionality  
15 may be provided by a separate chip. In at least one embodiment, network interfaces may include wireless functionality for communicating over LTE, WCDMA, UMTS, GSM, CDMA2000, Bluetooth, Bluetooth LE, Wi-Fi, Z-Wave, ZigBee, LoRaWAN, and/or other wireless protocols.

[0176] In at least one embodiment, vehicle 700 may further include data store(s) 728 which  
20 may include, without limitation, off-chip (e.g., off SoC(s) 704) storage. In at least one embodiment, data store(s) 728 may include, without limitation, one or more storage elements including RAM, SRAM, dynamic random-access memory (“DRAM”), video random-access memory (“VRAM”), flash memory, hard disks, and/or other components and/or devices that may store at least one bit of data.

25 [0177] In at least one embodiment, vehicle 700 may further include GNSS sensor(s) 758 (e.g., GPS and/or assisted GPS sensors), to assist in mapping, perception, occupancy grid generation, and/or path planning functions. In at least one embodiment, any number of GNSS sensor(s) 758 may be used, including, for example and without limitation, a GPS using a USB connector with an Ethernet-to-Serial (e.g., RS-232) bridge.

30 [0178] In at least one embodiment, vehicle 700 may further include RADAR sensor(s) 760. In at least one embodiment, RADAR sensor(s) 760 may be used by vehicle 700 for long-range vehicle detection, even in darkness and/or severe weather conditions. In at least one embodiment, RADAR functional safety levels may be ASIL B. In at least one embodiment, RADAR sensor(s) 760 may use a CAN bus and/or bus 702 (e.g., to transmit data generated  
35 by RADAR sensor(s) 760) for control and to access object tracking data, with access to Ethernet channels to access raw data in some examples. In at least one embodiment, a wide

5 variety of RADAR sensor types may be used. For example, and without limitation, RADAR sensor(s) 760 may be suitable for front, rear, and side RADAR use. In at least one embodiment, one or more sensor of RADAR sensors(s) 760 is a Pulse Doppler RADAR sensor.

[0179] In at least one embodiment, RADAR sensor(s) 760 may include different  
10 configurations, such as long-range with narrow field of view, short-range with wide field of view, short-range side coverage, etc. In at least one embodiment, long-range RADAR may be used for adaptive cruise control functionality. In at least one embodiment, long-range RADAR systems may provide a broad field of view realized by two or more independent scans, such as within a 250 m (meter) range. In at least one embodiment, RADAR sensor(s)  
15 760 may help in distinguishing between static and moving objects, and may be used by ADAS system 738 for emergency brake assist and forward collision warning. In at least one embodiment, sensors 760(s) included in a long-range RADAR system may include, without limitation, monostatic multimodal RADAR with multiple (e.g., six or more) fixed RADAR antennae and a high-speed CAN and FlexRay interface. In at least one embodiment, with six  
20 antennae, a central four antennae may create a focused beam pattern, designed to record vehicle's 700 surroundings at higher speeds with minimal interference from traffic in adjacent lanes. In at least one embodiment, another two antennae may expand field of view, making it possible to quickly detect vehicles entering or leaving a lane of vehicle 700.

[0180] In at least one embodiment, mid-range RADAR systems may include, as an example,  
25 a range of up to 160 m (front) or 80 m (rear), and a field of view of up to 42 degrees (front) or 150 degrees (rear). In at least one embodiment, short-range RADAR systems may include, without limitation, any number of RADAR sensor(s) 760 designed to be installed at both ends of a rear bumper. When installed at both ends of a rear bumper, in at least one embodiment, a RADAR sensor system may create two beams that constantly monitor blind  
30 spots in a rear direction and next to a vehicle. In at least one embodiment, short-range RADAR systems may be used in ADAS system 738 for blind spot detection and/or lane change assist.

[0181] In at least one embodiment, vehicle 700 may further include ultrasonic sensor(s) 762. In at least one embodiment, ultrasonic sensor(s) 762, which may be positioned at a front, a  
35 back, and/or side location of vehicle 700, may be used for parking assist and/or to create and update an occupancy grid. In at least one embodiment, a wide variety of ultrasonic sensor(s) 762 may be used, and different ultrasonic sensor(s) 762 may be used for different ranges of

5 detection (e.g., 2.5 m, 4 m). In at least one embodiment, ultrasonic sensor(s) 762 may operate at functional safety levels of ASIL B.

[0182] In at least one embodiment, vehicle 700 may include LIDAR sensor(s) 764. In at least one embodiment, LIDAR sensor(s) 764 may be used for object and pedestrian detection, emergency braking, collision avoidance, and/or other functions. In at least one embodiment,  
10 LIDAR sensor(s) 764 may operate at functional safety level ASIL B. In at least one embodiment, vehicle 700 may include multiple LIDAR sensors 764 (e.g., two, four, six, etc.) that may use an Ethernet channel (e.g., to provide data to a Gigabit Ethernet switch).

[0183] In at least one embodiment, LIDAR sensor(s) 764 may be capable of providing a list of objects and their distances for a 360-degree field of view. In at least one embodiment,  
15 commercially available LIDAR sensor(s) 764 may have an advertised range of approximately 100 m, with an accuracy of 2 cm to 3 cm, and with support for a 100 Mbps Ethernet connection, for example. In at least one embodiment, one or more non-protruding LIDAR sensors may be used. In such an embodiment, LIDAR sensor(s) 764 may include a small device that may be embedded into a front, a rear, a side, and/or a corner location of vehicle  
20 700. In at least one embodiment, LIDAR sensor(s) 764, in such an embodiment, may provide up to a 120-degree horizontal and 35-degree vertical field-of-view, with a 200 m range even for low-reflectivity objects. In at least one embodiment, front-mounted LIDAR sensor(s) 764 may be configured for a horizontal field of view between 45 degrees and 135 degrees.

25 [0184] In at least one embodiment, LIDAR technologies, such as 3D flash LIDAR, may also be used. In at least one embodiment, 3D flash LIDAR uses a flash of a laser as a transmission source, to illuminate surroundings of vehicle 700 up to approximately 200 m. In at least one embodiment, a flash LIDAR unit includes, without limitation, a receptor, which records laser pulse transit time and reflected light on each pixel, which in turn  
30 corresponds to a range from vehicle 700 to objects. In at least one embodiment, flash LIDAR may allow for highly accurate and distortion-free images of surroundings to be generated with every laser flash. In at least one embodiment, four flash LIDAR sensors may be deployed, one at each side of vehicle 700. In at least one embodiment, 3D flash LIDAR systems include, without limitation, a solid-state 3D staring array LIDAR camera with no  
35 moving parts other than a fan (e.g., a non-scanning LIDAR device). In at least one embodiment, flash LIDAR device may use a 5 nanosecond class I (eye-safe) laser pulse per

5 frame and may capture reflected laser light as a 3D range point cloud and co-registered intensity data.

[0185] In at least one embodiment, vehicle 700 may further include IMU sensor(s) 766. In at least one embodiment, IMU sensor(s) 766 may be located at a center of a rear axle of vehicle 700. In at least one embodiment, IMU sensor(s) 766 may include, for example and  
10 without limitation, accelerometer(s), magnetometer(s), gyroscope(s), a magnetic compass, magnetic compasses, and/or other sensor types. In at least one embodiment, such as in six-axis applications, IMU sensor(s) 766 may include, without limitation, accelerometers and gyroscopes. In at least one embodiment, such as in nine-axis applications, IMU sensor(s) 766 may include, without limitation, accelerometers, gyroscopes, and magnetometers.

15 [0186] In at least one embodiment, IMU sensor(s) 766 may be implemented as a miniature, high performance GPS-Aided Inertial Navigation System (“GPS/INS”) that combines micro-electro-mechanical systems (“MEMS”) inertial sensors, a high-sensitivity GPS receiver, and advanced Kalman filtering algorithms to provide estimates of position, velocity, and attitude. In at least one embodiment, IMU sensor(s) 766 may enable vehicle 700 to estimate its  
20 heading without requiring input from a magnetic sensor by directly observing and correlating changes in velocity from a GPS to IMU sensor(s) 766. In at least one embodiment, IMU sensor(s) 766 and GNSS sensor(s) 758 may be combined in a single integrated unit.

[0187] In at least one embodiment, vehicle 700 may include microphone(s) 796 placed in and/or around vehicle 700. In at least one embodiment, microphone(s) 796 may be used for  
25 emergency vehicle detection and identification, among other things.

[0188] In at least one embodiment, vehicle 700 may further include any number of camera types, including stereo camera(s) 768, wide-view camera(s) 770, infrared camera(s) 772, surround camera(s) 774, long-range camera(s) 798, mid-range camera(s) 776, and/or other camera types. In at least one embodiment, cameras may be used to capture image data  
30 around an entire periphery of vehicle 700. In at least one embodiment, which types of cameras used depends on vehicle 700. In at least one embodiment, any combination of camera types may be used to provide necessary coverage around vehicle 700. In at least one embodiment, a number of cameras deployed may differ depending on embodiment. For example, in at least one embodiment, vehicle 700 could include six cameras, seven cameras,  
35 ten cameras, twelve cameras, or another number of cameras. In at least one embodiment, cameras may support, as an example and without limitation, Gigabit Multimedia Serial Link

5 (“GMSL”) and/or Gigabit Ethernet communications. In at least one embodiment, each camera might be as described with more detail previously herein with respect to FIG. 7A and FIG. 7B.

[0189] In at least one embodiment, vehicle 700 may further include vibration sensor(s) 742. In at least one embodiment, vibration sensor(s) 742 may measure vibrations of components of  
10 vehicle 700, such as axle(s). For example, in at least one embodiment, changes in vibrations may indicate a change in road surfaces. In at least one embodiment, when two or more vibration sensors 742 are used, differences between vibrations may be used to determine friction or slippage of road surface (e.g., when a difference in vibration is between a power-driven axle and a freely rotating axle).

15 [0190] In at least one embodiment, vehicle 700 may include ADAS system 738. In at least one embodiment, ADAS system 738 may include, without limitation, an SoC, in some examples. In at least one embodiment, ADAS system 738 may include, without limitation, any number and combination of an autonomous/adaptive/automatic cruise control (“ACC”) system, a cooperative adaptive cruise control (“CACC”) system, a forward crash warning  
20 (“FCW”) system, an automatic emergency braking (“AEB”) system, a lane departure warning (“LDW”) system, a lane keep assist (“LKA”) system, a blind spot warning (“BSW”) system, a rear cross-traffic warning (“RCTW”) system, a collision warning (“CW”) system, a lane centering (“LC”) system, and/or other systems, features, and/or functionality.

[0191] In at least one embodiment, ACC system may use RADAR sensor(s) 760, LIDAR  
25 sensor(s) 764, and/or any number of camera(s). In at least one embodiment, ACC system may include a longitudinal ACC system and/or a lateral ACC system. In at least one embodiment, a longitudinal ACC system monitors and controls distance to another vehicle immediately ahead of vehicle 700 and automatically adjusts speed of vehicle 700 to maintain a safe distance from vehicles ahead. In at least one embodiment, a lateral ACC system  
30 performs distance keeping, and advises vehicle 700 to change lanes when necessary. In at least one embodiment, a lateral ACC is related to other ADAS applications, such as LC and CW.

[0192] In at least one embodiment, a CACC system uses information from other vehicles that may be received via network interface 724 and/or wireless antenna(s) 726 from other vehicles  
35 via a wireless link, or indirectly, over a network connection (e.g., over the Internet). In at least one embodiment, direct links may be provided by a vehicle-to-vehicle (“V2V”)

5 communication link, while indirect links may be provided by an infrastructure-to-vehicle (“I2V”) communication link. In general, V2V communication provides information about immediately preceding vehicles (e.g., vehicles immediately ahead of and in same lane as vehicle 700), while I2V communication provides information about traffic further ahead. In at least one embodiment, a CACC system may include either or both I2V and V2V  
10 information sources. In at least one embodiment, given information of vehicles ahead of vehicle 700, a CACC system may be more reliable and it has potential to improve traffic flow smoothness and reduce congestion on road.

[0193] In at least one embodiment, an FCW system is designed to alert a driver to a hazard, so that such driver may take corrective action. In at least one embodiment, an FCW system  
15 uses a front-facing camera and/or RADAR sensor(s) 760, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, an FCW system may provide a warning, such as in form of a sound, visual warning, vibration and/or a quick brake pulse.

20 [0194] In at least one embodiment, an AEB system detects an impending forward collision with another vehicle or other object, and may automatically apply brakes if a driver does not take corrective action within a specified time or distance parameter. In at least one embodiment, AEB system may use front-facing camera(s) and/or RADAR sensor(s) 760, coupled to a dedicated processor, DSP, FPGA, and/or ASIC. In at least one embodiment,  
25 when an AEB system detects a hazard, it will typically first alert a driver to take corrective action to avoid collision and, if that driver does not take corrective action, that AEB system may automatically apply brakes in an effort to prevent, or at least mitigate, an impact of a predicted collision. In at least one embodiment, an AEB system may include techniques such as dynamic brake support and/or crash imminent braking.

30 [0195] In at least one embodiment, an LDW system provides visual, audible, and/or tactile warnings, such as steering wheel or seat vibrations, to alert driver when vehicle 700 crosses lane markings. In at least one embodiment, an LDW system does not activate when a driver indicates an intentional lane departure, such as by activating a turn signal. In at least one embodiment, an LDW system may use front-side facing cameras, coupled to a dedicated  
35 processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, an LKA

5 system is a variation of an LDW system. In at least one embodiment, an LKA system provides steering input or braking to correct vehicle 700 if vehicle 700 starts to exit its lane.

[0196] In at least one embodiment, a BSW system detects and warns a driver of vehicles in an automobile's blind spot. In at least one embodiment, a BSW system may provide a visual, audible, and/or tactile alert to indicate that merging or changing lanes is unsafe. In at  
10 least one embodiment, a BSW system may provide an additional warning when a driver uses a turn signal. In at least one embodiment, a BSW system may use rear-side facing camera(s) and/or RADAR sensor(s) 760, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component.

15 [0197] In at least one embodiment, an RCTW system may provide visual, audible, and/or tactile notification when an object is detected outside a rear-camera range when vehicle 700 is backing up. In at least one embodiment, an RCTW system includes an AEB system to ensure that vehicle brakes are applied to avoid a crash. In at least one embodiment, an RCTW system may use one or more rear-facing RADAR sensor(s) 760, coupled to a  
20 dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to provide driver feedback, such as a display, speaker, and/or vibrating component.

[0198] In at least one embodiment, conventional ADAS systems may be prone to false positive results which may be annoying and distracting to a driver, but typically are not catastrophic, because conventional ADAS systems alert a driver and allow that driver to  
25 decide whether a safety condition truly exists and act accordingly. In at least one embodiment, vehicle 700 itself decides, in case of conflicting results, whether to heed result from a primary computer or a secondary computer (e.g., a first controller or a second controller of controllers 736). For example, in at least one embodiment, ADAS system 738 may be a backup and/or secondary computer for providing perception information to a  
30 backup computer rationality module. In at least one embodiment, a backup computer rationality monitor may run redundant diverse software on hardware components to detect faults in perception and dynamic driving tasks. In at least one embodiment, outputs from ADAS system 738 may be provided to a supervisory MCU. In at least one embodiment, if outputs from a primary computer and outputs from a secondary computer conflict, a  
35 supervisory MCU determines how to reconcile conflict to ensure safe operation.

5 [0199] In at least one embodiment, a primary computer may be configured to provide a supervisory MCU with a confidence score, indicating that primary computer's confidence in a chosen result. In at least one embodiment, if that confidence score exceeds a threshold, that supervisory MCU may follow that primary computer's direction, regardless of whether that secondary computer provides a conflicting or inconsistent result. In at least one  
10 embodiment, where a confidence score does not meet a threshold, and where primary and secondary computers indicate different results (e.g., a conflict), a supervisory MCU may arbitrate between computers to determine an appropriate outcome.

[0200] In at least one embodiment, a supervisory MCU may be configured to run a neural network(s) that is trained and configured to determine, based at least in part on outputs from a  
15 primary computer and outputs from a secondary computer, conditions under which that secondary computer provides false alarms. In at least one embodiment, neural network(s) in a supervisory MCU may learn when a secondary computer's output may be trusted, and when it cannot. For example, in at least one embodiment, when that secondary computer is a RADAR-based FCW system, a neural network(s) in that supervisory MCU may learn when  
20 an FCW system is identifying metallic objects that are not, in fact, hazards, such as a drainage grate or manhole cover that triggers an alarm. In at least one embodiment, when a secondary computer is a camera-based LDW system, a neural network in a supervisory MCU may learn to override LDW when bicyclists or pedestrians are present and a lane departure is, in fact, a safest maneuver. In at least one embodiment, a supervisory MCU may include at  
25 least one of a DLA or a GPU suitable for running neural network(s) with associated memory. In at least one embodiment, a supervisory MCU may comprise and/or be included as a component of SoC(s) 704.

[0201] In at least one embodiment, ADAS system 738 may include a secondary computer that performs ADAS functionality using traditional rules of computer vision. In at least one  
30 embodiment, that secondary computer may use classic computer vision rules (if-then), and presence of a neural network(s) in a supervisory MCU may improve reliability, safety and performance. For example, in at least one embodiment, diverse implementation and intentional non-identity makes an overall system more fault-tolerant, especially to faults caused by software (or software-hardware interface) functionality. For example, in at least  
35 one embodiment, if there is a software bug or error in software running on a primary computer, and non-identical software code running on a secondary computer provides a consistent overall result, then a supervisory MCU may have greater confidence that an overall

5 result is correct, and a bug in software or hardware on that primary computer is not causing a material error.

[0202] In at least one embodiment, an output of ADAS system 738 may be fed into a primary computer's perception block and/or a primary computer's dynamic driving task block. For example, in at least one embodiment, if ADAS system 738 indicates a forward crash warning  
10 due to an object immediately ahead, a perception block may use this information when identifying objects. In at least one embodiment, a secondary computer may have its own neural network that is trained and thus reduces a risk of false positives, as described herein.

[0203] In at least one embodiment, vehicle 700 may further include infotainment SoC 730 (e.g., an in-vehicle infotainment system (IVI)). Although illustrated and described as an  
15 SoC, infotainment system SoC 730, in at least one embodiment, may not be an SoC, and may include, without limitation, two or more discrete components. In at least one embodiment, infotainment SoC 730 may include, without limitation, a combination of hardware and software that may be used to provide audio (e.g., music, a personal digital assistant, navigational instructions, news, radio, etc.), video (e.g., TV, movies, streaming, etc.), phone  
20 (e.g., hands-free calling), network connectivity (e.g., LTE, WiFi, etc.), and/or information services (e.g., navigation systems, rear-parking assistance, a radio data system, vehicle related information such as fuel level, total distance covered, brake fuel level, oil level, door open/close, air filter information, etc.) to vehicle 700. For example, infotainment SoC 730 could include radios, disk players, navigation systems, video players, USB and Bluetooth  
25 connectivity, carputers, in-car entertainment, WiFi, steering wheel audio controls, hands free voice control, a heads-up display ("HUD"), HMI display 734, a telematics device, a control panel (e.g., for controlling and/or interacting with various components, features, and/or systems), and/or other components. In at least one embodiment, infotainment SoC 730 may further be used to provide information (e.g., visual and/or audible) to user(s) of vehicle 700,  
30 such as information from ADAS system 738, autonomous driving information such as planned vehicle maneuvers, trajectories, surrounding environment information (e.g., intersection information, vehicle information, road information, etc.), and/or other information.

[0204] In at least one embodiment, infotainment SoC 730 may include any amount and type  
35 of GPU functionality. In at least one embodiment, infotainment SoC 730 may communicate over bus 702 with other devices, systems, and/or components of vehicle 700. In at least one embodiment, infotainment SoC 730 may be coupled to a supervisory MCU such that a GPU

5 of an infotainment system may perform some self-driving functions in event that primary controller(s) 736 (e.g., primary and/or backup computers of vehicle 700) fail. In at least one embodiment, infotainment SoC 730 may put vehicle 700 into a chauffeur to safe stop mode, as described herein.

[0205] In at least one embodiment, vehicle 700 may further include instrument cluster 732  
10 (e.g., a digital dash, an electronic instrument cluster, a digital instrument panel, etc.). In at least one embodiment, instrument cluster 732 may include, without limitation, a controller and/or supercomputer (e.g., a discrete controller or supercomputer). In at least one embodiment, instrument cluster 732 may include, without limitation, any number and combination of a set of instrumentation such as a speedometer, fuel level, oil pressure,  
15 tachometer, odometer, turn indicators, gearshift position indicator, seat belt warning light(s), parking-brake warning light(s), engine-malfunction light(s), supplemental restraint system (e.g., airbag) information, lighting controls, safety system controls, navigation information, etc. In some examples, information may be displayed and/or shared among infotainment SoC 730 and instrument cluster 732. In at least one embodiment, instrument cluster 732  
20 may be included as part of infotainment SoC 730, or vice versa.

[0206] Inference and/or training logic 815 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 815 are provided herein in conjunction with FIGS. 8A and/or 8B. In at least one embodiment, inference and/or training logic 815 may be used in system FIG. 7C for  
25 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.

[0207] In at least one embodiment, techniques described in connection with FIGS. 1-6 are implemented in context of a system architecture in accordance with FIG. 7C. In at least one  
30 embodiment, vehicle 700 is implemented in context of a processor comprising: one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images

[0208] FIG. 7D is a diagram of a system 776 for communication between cloud-based  
35 server(s) and autonomous vehicle 700 of FIG. 7A, according to at least one embodiment. In at least one embodiment, system 776 may include, without limitation, server(s) 778,

5 network(s) 790, and any number and type of vehicles, including vehicle 700. In at least one embodiment, server(s) 778 may include, without limitation, a plurality of GPUs 784(A)-784(H) (collectively referred to herein as GPUs 784), PCIe switches 782(A)-782(D) (collectively referred to herein as PCIe switches 782), and/or CPUs 780(A)-780(B) (collectively referred to herein as CPUs 780). In at least one embodiment, GPUs 784, CPUs 10 780, and PCIe switches 782 may be interconnected with high-speed interconnects such as, for example and without limitation, NVLink interfaces 788 developed by NVIDIA and/or PCIe connections 786. In at least one embodiment, GPUs 784 are connected via an NVLink and/or NVSwitch SoC and GPUs 784 and PCIe switches 782 are connected via PCIe interconnects. Although eight GPUs 784, two CPUs 780, and four PCIe switches 782 are 15 illustrated, this is not intended to be limiting. In at least one embodiment, each of server(s) 778 may include, without limitation, any number of GPUs 784, CPUs 780, and/or PCIe switches 782, in any combination. For example, in at least one embodiment, server(s) 778 could each include eight, sixteen, thirty-two, and/or more GPUs 784.

[0209] In at least one embodiment, server(s) 778 may receive, over network(s) 790 and from 20 vehicles, image data representative of images showing unexpected or changed road conditions, such as recently commenced road-work. In at least one embodiment, server(s) 778 may transmit, over network(s) 790 and to vehicles, neural networks 792, updated or otherwise, and/or map information 794, including, without limitation, information regarding traffic and road conditions. In at least one embodiment, updates to map information 794 25 may include, without limitation, updates for HD map 722, such as information regarding construction sites, potholes, detours, flooding, and/or other obstructions. In at least one embodiment, neural networks 792, and/or map information 794 may have resulted from new training and/or experiences represented in data received from any number of vehicles in an environment, and/or based at least in part on training performed at a data center (e.g., using 30 server(s) 778 and/or other servers).

[0210] In at least one embodiment, server(s) 778 may be used to train machine learning models (e.g., neural networks) based at least in part on training data. In at least one embodiment, training data may be generated by vehicles, and/or may be generated in a simulation (e.g., using a game engine). In at least one embodiment, any amount of training 35 data is tagged (e.g., where associated neural network benefits from supervised learning) and/or undergoes other pre-processing. In at least one embodiment, any amount of training data is not tagged and/or pre-processed (e.g., where associated neural network does not

5 require supervised learning). In at least one embodiment, once machine learning models are trained, machine learning models may be used by vehicles (e.g., transmitted to vehicles over network(s) 790), and/or machine learning models may be used by server(s) 778 to remotely monitor vehicles.

[0211] In at least one embodiment, server(s) 778 may receive data from vehicles and apply  
10 data to up-to-date real-time neural networks for real-time intelligent inferencing. In at least one embodiment, server(s) 778 may include deep-learning supercomputers and/or dedicated AI computers powered by GPU(s) 784, such as a DGX and DGX Station machines developed by NVIDIA. However, in at least one embodiment, server(s) 778 may include deep learning infrastructure that uses CPU-powered data centers.

15 [0212] In at least one embodiment, deep-learning infrastructure of server(s) 778 may be capable of fast, real-time inferencing, and may use that capability to evaluate and verify health of processors, software, and/or associated hardware in vehicle 700. For example, in at least one embodiment, deep-learning infrastructure may receive periodic updates from vehicle 700, such as a sequence of images and/or objects that vehicle 700 has located in that  
20 sequence of images (e.g., via computer vision and/or other machine learning object classification techniques). In at least one embodiment, deep-learning infrastructure may run its own neural network to identify objects and compare them with objects identified by vehicle 700 and, if results do not match and deep-learning infrastructure concludes that AI in vehicle 700 is malfunctioning, then server(s) 778 may transmit a signal to vehicle 700  
25 instructing a fail-safe computer of vehicle 700 to assume control, notify passengers, and complete a safe parking maneuver.

[0213] In at least one embodiment, server(s) 778 may include GPU(s) 784 and one or more programmable inference accelerators (e.g., NVIDIA's TensorRT 3 devices). In at least one embodiment, a combination of GPU-powered servers and inference acceleration may make  
30 real-time responsiveness possible. In at least one embodiment, such as where performance is less critical, servers powered by CPUs, FPGAs, and other processors may be used for inferencing. In at least one embodiment, hardware structure(s) 815 are used to perform one or more embodiments. Details regarding hardware structure(s) 815 are provided herein in conjunction with FIGS. 8A and/or 8B.

35 [0214] In at least one embodiment, techniques described in connection with FIGS. 1-6 are implemented in context of a system system for communication between cloud-based server(s)

- 5 and an autonomous vehicle of FIG. 7A, in accordance with FIG. 7C. In at least one embodiment, vehicle 700 is implemented in context of a processor comprising: one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images
- 10 [0215] In the following description, numerous specific details are set forth to provide a more thorough understanding of at least one embodiment. However, it will be apparent to one skilled in the art that the inventive concepts may be practiced without one or more of these specific details.

### Data Center

- 15 [0216] FIG. 9 illustrates an exemplary data center 900, in accordance with at least one embodiment. In at least one embodiment, data center 900 includes, without limitation, a data center infrastructure layer 910, a framework layer 920, a software layer 930 and an application layer 940.
- [0217] In at least one embodiment, as shown in FIG. 9, data center infrastructure layer 910 may include a resource orchestrator 912, grouped computing resources 914, and node computing resources ("node C.R.s") 916(1)-916(N), where "N" represents any whole, positive integer. In at least one embodiment, node C.R.s 916(1)-916(N) may include, but are not limited to, any number of central processing units ("CPUs") or other processors (including accelerators, field programmable gate arrays ("FPGAs"), graphics processors, etc.), memory 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 916(1)-916(N) may be a server having one or more of above-mentioned computing resources.
- 20
- 25
- 30 [0218] In at least one embodiment, grouped computing resources 914 may include separate groupings of node C.R.s housed within one or more racks (not shown), or many racks housed in data centers at various geographical locations (also not shown). Separate groupings of node C.R.s within grouped computing resources 914 may include grouped compute, network, memory or storage resources that may be configured or allocated to support one or more
- 35 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

5 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.

[0219] In at least one embodiment, resource orchestrator 912 may configure or otherwise control one or more node C.R.s 916(1)-916(N) and/or grouped computing resources 914. In at least one embodiment, resource orchestrator 912 may include a software design  
10 infrastructure (“SDI”) management entity for data center 900. In at least one embodiment, resource orchestrator 912 may include hardware, software or some combination thereof.

[0220] In at least one embodiment, as shown in FIG. 9, framework layer 920 includes, without limitation, a job scheduler 932, a configuration manager 934, a resource manager 936 and a distributed file system 938. In at least one embodiment, framework layer 920 may  
15 include a framework to support software 952 of software layer 930 and/or one or more application(s) 942 of application layer 940. In at least one embodiment, software 952 or application(s) 942 may respectively include web-based service software or applications, such as those provided by Amazon Web Services, Google Cloud and Microsoft Azure. In at least one embodiment, framework layer 920 may be, but is not limited to, a type of free and open-  
20 source software web application framework such as Apache Spark™ (hereinafter “Spark”) that may utilize distributed file system 938 for large-scale data processing (e.g., “big data”). In at least one embodiment, job scheduler 932 may include a Spark driver to facilitate scheduling of workloads supported by various layers of data center 900. In at least one embodiment, configuration manager 934 may be capable of configuring different layers such  
25 as software layer 930 and framework layer 920, including Spark and distributed file system 938 for supporting large-scale data processing. In at least one embodiment, resource manager 936 may be capable of managing clustered or grouped computing resources mapped to or allocated for support of distributed file system 938 and job scheduler 932. In at least one embodiment, clustered or grouped computing resources may include grouped computing  
30 resource 914 at data center infrastructure layer 910. In at least one embodiment, resource manager 936 may coordinate with resource orchestrator 912 to manage these mapped or allocated computing resources.

[0221] In at least one embodiment, software 952 included in software layer 930 may include software used by at least portions of node C.R.s 916(1)-916(N), grouped computing resources  
35 914, and/or distributed file system 938 of framework layer 920. 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.

- 5     **[0222]** In at least one embodiment, application(s) 942 included in application layer 940 may include one or more types of applications used by at least portions of node C.R.s 916(1)-916(N), grouped computing resources 914, and/or distributed file system 938 of framework layer 920. In at least one or more types of applications may include, without limitation, CUDA applications.
- 10    **[0223]** In at least one embodiment, any of configuration manager 934, resource manager 936, and resource orchestrator 912 may implement any number and type of self-modifying actions based on any amount and type of data acquired in any technically feasible fashion. In at least one embodiment, self-modifying actions may relieve a data center operator of data center 900 from making possibly bad configuration decisions and possibly avoiding underutilized and/or
- 15    poor performing portions of a data center.

### **Computer-Based Systems**

**[0224]** The following FIGS. set forth, without limitation, exemplary computer-based systems that can be used to implement at least one embodiment.

- [0225]** FIG. 10 illustrates a processing system 1000, in accordance with at least one
- 20    embodiment. In at least one embodiment, processing system 1000 includes one or more processors 1002 and one or more graphics processors 1008, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 1002 or processor cores 1007. In at least one embodiment, processing system 1000 is a processing platform incorporated within a system-on-a-chip (“SoC”)
- 25    integrated circuit for use in mobile, handheld, or embedded devices.

- [0226]** In at least one embodiment, processing system 1000 can include, or be incorporated within a server-based gaming platform, a game console, a media console, a mobile gaming console, a handheld game console, or an online game console. In at least one embodiment, processing system 1000 is a mobile phone, smart phone, tablet computing device or mobile
- 30    Internet device. In at least one embodiment, processing system 1000 can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, smart eyewear device, augmented reality device, or virtual reality device. In at least one embodiment, processing system 1000 is a television or set top box device having one or more processors 1002 and a graphical interface generated by one or more graphics processors
- 35    1008.

- 5     **[0227]** In at least one embodiment, one or more processors 1002 each include one or more processor cores 1007 to process instructions which, when executed, perform operations for system and user software. In at least one embodiment, each of one or more processor cores 1007 is configured to process a specific instruction set 1009. In at least one embodiment, instruction set 1009 may facilitate Complex Instruction Set Computing (“CISC”), Reduced  
10    Instruction Set Computing (“RISC”), or computing via a Very Long Instruction Word (“VLIW”). In at least one embodiment, processor cores 1007 may each process a different instruction set 1009, which may include instructions to facilitate emulation of other instruction sets. In at least one embodiment, processor core 1007 may also include other processing devices, such as a digital signal processor (“DSP”).
- 15    **[0228]** In at least one embodiment, processor 1002 includes cache memory (“cache”) 1004. In at least one embodiment, processor 1002 can have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory is shared among various components of processor 1002. In at least one embodiment, processor 1002 also uses an external cache (e.g., a Level 3 (“L3”) cache or Last Level Cache (“LLC”)) (not shown),  
20    which may be shared among processor cores 1007 using known cache coherency techniques. In at least one embodiment, register file 1006 is additionally included in processor 1002 which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). In at least one embodiment, register file 1006 may include general-purpose registers or other  
25    registers.
- 30    **[0229]** In at least one embodiment, one or more processor(s) 1002 are coupled with one or more interface bus(es) 1010 to transmit communication signals such as address, data, or control signals between processor 1002 and other components in processing system 1000. In at least one embodiment interface bus 1010, in one embodiment, can be a processor bus, such as a version of a Direct Media Interface (“DMI”) bus. In at least one embodiment, interface  
35    bus 1010 is not limited to a DMI bus, and may include one or more Peripheral Component Interconnect buses (e.g., “PCI,” PCI Express (“PCIe”)), memory buses, or other types of interface buses. In at least one embodiment processor(s) 1002 include an integrated memory controller 1016 and a platform controller hub 1030. In at least one embodiment, memory controller 1016 facilitates communication between a memory device and other components of processing system 1000, while platform controller hub (“PCH”) 1030 provides connections to Input/Output (“I/O”) devices via a local I/O bus.

5     **[0230]** In at least one embodiment, memory device 1020 can be a dynamic random access memory (“DRAM”) device, a static random access memory (“SRAM”) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as processor memory. In at least one embodiment memory device 1020 can operate as system memory for processing system 1000, to store data 1022 and

10    instructions 1021 for use when one or more processors 1002 executes an application or process. In at least one embodiment, memory controller 1016 also couples with an optional external graphics processor 1012, which may communicate with one or more graphics processors 1008 in processors 1002 to perform graphics and media operations. In at least one embodiment, a display device 1011 can connect to processor(s) 1002. In at least one

15    embodiment display device 1011 can include one or more of an internal display device, as in a mobile electronic device or a laptop device or an external display device attached via a display interface (e.g., DisplayPort, etc.). In at least one embodiment, display device 1011 can include a head mounted display (“HMD”) such as a stereoscopic display device for use in virtual reality (“VR”) applications or augmented reality (“AR”) applications.

20    **[0231]** In at least one embodiment, platform controller hub 1030 enables peripherals to connect to memory device 1020 and processor 1002 via a high-speed I/O bus. In at least one embodiment, I/O peripherals include, but are not limited to, an audio controller 1046, a network controller 1034, a firmware interface 1028, a wireless transceiver 1026, touch sensors 1025, a data storage device 1024 (e.g., hard disk drive, flash memory, etc.). In at least

25    one embodiment, data storage device 1024 can connect via a storage interface (e.g., SATA) or via a peripheral bus, such as PCI, or PCIe. In at least one embodiment, touch sensors 1025 can include touch screen sensors, pressure sensors, or fingerprint sensors. In at least one embodiment, wireless transceiver 1026 can be a Wi-Fi transceiver, a Bluetooth transceiver, or a mobile network transceiver such as a 3G, 4G, or Long Term Evolution (“LTE”) transceiver.

30    In at least one embodiment, firmware interface 1028 enables communication with system firmware, and can be, for example, a unified extensible firmware interface (“UEFI”). In at least one embodiment, network controller 1034 can enable a network connection to a wired network. In at least one embodiment, a high-performance network controller (not shown) couples with interface bus 1010. In at least one embodiment, audio controller 1046 is a multi-

35    channel high definition audio controller. In at least one embodiment, processing system 1000 includes an optional legacy I/O controller 1040 for coupling legacy (e.g., Personal System 2 (“PS/2”)) devices to processing system 1000. In at least one embodiment, platform controller

5 hub 1030 can also connect to one or more Universal Serial Bus (“USB”) controllers 1042 connect input devices, such as keyboard and mouse 1043 combinations, a camera 1044, or other USB input devices.

[0232] In at least one embodiment, an instance of memory controller 1016 and platform controller hub 1030 may be integrated into a discreet external graphics processor, such as  
10 external graphics processor 1012. In at least one embodiment, platform controller hub 1030 and/or memory controller 1016 may be external to one or more processor(s) 1002. For example, in at least one embodiment, processing system 1000 can include an external memory controller 1016 and platform controller hub 1030, which may be configured as a memory controller hub and peripheral controller hub within a system chipset that is in  
15 communication with processor(s) 1002.

[0233] FIG. 11 illustrates a computer system 1100, in accordance with at least one embodiment. In at least one embodiment, computer system 1100 may be a system with interconnected devices and components, an SOC, or some combination. In at least one embodiment, computer system 1100 is formed with a processor 1102 that may include  
20 execution units to execute an instruction. In at least one embodiment, computer system 1100 may include, without limitation, a component, such as processor 1102 to employ execution units including logic to perform algorithms for processing data. In at least one embodiment, computer system 1100 may include processors, such as PENTIUM® Processor family, Xeon™, Itanium®, XScale™ and/or StrongARM™, Intel® Core™, or Intel® Nervana™  
25 microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and like) may also be used. In at least one embodiment, computer system 1100 may execute a version of WINDOWS® operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems (UNIX and Linux for example),  
30 embedded software, and/or graphical user interfaces, may also be used.

[0234] In at least one embodiment, computer system 1100 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  
35 include a microcontroller, a digital signal processor (DSP), an SoC, network computers (“NetPCs”), set-top boxes, network hubs, wide area network (“WAN”) switches, or any other system that may perform one or more instructions.

5 [0235] In at least one embodiment, computer system 1100 may include, without limitation, processor 1102 that may include, without limitation, one or more execution units 1108 that may be configured to execute a Compute Unified Device Architecture (“CUDA”) (CUDA® is developed by NVIDIA Corporation of Santa Clara, CA) program. In at least one embodiment, a CUDA program is at least a portion of a software application written in a  
10 CUDA programming language. In at least one embodiment, computer system 1100 is a single processor desktop or server system. In at least one embodiment, computer system 1100 may be a multiprocessor system. In at least one embodiment, processor 1102 may include, without limitation, a CISC microprocessor, a RISC microprocessor, a VLIW microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such  
15 as a digital signal processor, for example. In at least one embodiment, processor 1102 may be coupled to a processor bus 1110 that may transmit data signals between processor 1102 and other components in computer system 1100.

[0236] In at least one embodiment, processor 1102 may include, without limitation, a Level 1 (“L1”) internal cache memory (“cache”) 1104. In at least one embodiment, processor 1102  
20 may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor 1102. In at least one embodiment, processor 1102 may also include a combination of both internal and external caches. In at least one embodiment, a register file 1106 may store different types of data in various registers including, without limitation, integer registers, floating point registers, status  
25 registers, and instruction pointer register.

[0237] In at least one embodiment, execution unit 1108, including, without limitation, logic to perform integer and floating point operations, also resides in processor 1102. Processor 1102 may also include a microcode (“ucode”) read only memory (“ROM”) that stores microcode for certain macro instructions. In at least one embodiment, execution unit 1108  
30 may include logic to handle a packed instruction set 1109. In at least one embodiment, by including packed instruction set 1109 in an instruction set of a general-purpose processor 1102, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using packed data in a general-purpose processor 1102. In at least one embodiment, many multimedia applications may be accelerated and  
35 executed more efficiently by using full width of a processor's data bus for performing operations on packed data, which may eliminate a need to transfer smaller units of data across a processor's data bus to perform one or more operations one data element at a time.

5 [0238] In at least one embodiment, execution unit 1108 may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. In at least one embodiment, computer system 1100 may include, without limitation, a memory 1120. In at least one embodiment, memory 1120 may be implemented as a DRAM device, an SRAM device, flash memory device, or other memory device. Memory 1120 may store instruction(s)  
10 1119 and/or data 1121 represented by data signals that may be executed by processor 1102.

[0239] In at least one embodiment, a system logic chip may be coupled to processor bus 1110 and memory 1120. In at least one embodiment, the system logic chip may include, without limitation, a memory controller hub (“MCH”) 1116, and processor 1102 may communicate with MCH 1116 via processor bus 1110. In at least one embodiment, MCH 1116 may  
15 provide a high bandwidth memory path 1118 to memory 1120 for instruction and data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH 1116 may direct data signals between processor 1102, memory 1120, and other components in computer system 1100 and to bridge data signals between processor bus 1110, memory 1120, and a system I/O 1122. In at least one embodiment, system logic chip may provide a  
20 graphics port for coupling to a graphics controller. In at least one embodiment, MCH 1116 may be coupled to memory 1120 through high bandwidth memory path 1118 and graphics/video card 1112 may be coupled to MCH 1116 through an Accelerated Graphics Port (“AGP”) interconnect 1114.

[0240] In at least one embodiment, computer system 1100 may use system I/O 1122 that is a  
25 proprietary hub interface bus to couple MCH 1116 to I/O controller hub (“ICH”) 1130. In at least one embodiment, ICH 1130 may provide direct connections to some I/O devices via a local I/O bus. In at least one embodiment, local I/O bus may include, without limitation, a high-speed I/O bus for connecting peripherals to memory 1120, a chipset, and processor 1102. Examples may include, without limitation, an audio controller 1129, a firmware hub (“flash BIOS”) 1128, a wireless transceiver 1126, a data storage 1124, a legacy I/O controller  
30 1123 containing a user input interface 1125 and a keyboard interface, a serial expansion port 1127, such as a USB, and a network controller 1134. Data storage 1124 may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

35 [0241] In at least one embodiment, FIG. 11 illustrates a system, which includes interconnected hardware devices or “chips.” In at least one embodiment, FIG. 11 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 11 may

5 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 1100 are interconnected using compute express link (“CXL”) interconnects.

[0242] FIG. 12 illustrates a system 1200, in accordance with at least one embodiment. In at least one embodiment, system 1200 is an electronic device that utilizes a processor 1210. In  
10 at least one embodiment, system 1200 may be, for example and without limitation, a notebook, a tower server, a rack server, a blade server, a laptop, a desktop, a tablet, a mobile device, a phone, an embedded computer, or any other suitable electronic device.

[0243] In at least one embodiment, system 1200 may include, without limitation, processor 1210 communicatively coupled to any suitable number or kind of components, peripherals,  
15 modules, or devices. In at least one embodiment, processor 1210 is coupled using a bus or interface, such as an I<sup>2</sup>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 USB (versions 1, 2, 3), or a Universal Asynchronous Receiver/Transmitter (“UART”) bus. In at least one embodiment,  
20 FIG. 12 illustrates a system which includes interconnected hardware devices or “chips.” In at least one embodiment, FIG. 12 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 12 may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of FIG. 12 are interconnected using CXL  
25 interconnects.

[0244] In at least one embodiment, FIG 12 may include a display 1224, a touch screen 1225, a touch pad 1230, a Near Field Communications unit (“NFC”) 1245, a sensor hub 1240, a thermal sensor 1246, an Express Chipset (“EC”) 1235, a Trusted Platform Module (“TPM”) 1238, BIOS/firmware/flash memory (“BIOS, FW Flash”) 1222, a DSP 1260, a Solid State  
30 Disk (“SSD”) or Hard Disk Drive (“HDD”) 1220, a wireless local area network unit (“WLAN”) 1250, a Bluetooth unit 1252, a Wireless Wide Area Network unit (“WWAN”) 1256, a Global Positioning System (“GPS”) 1255, a camera (“USB 3.0 camera”) 1254 such as a USB 3.0 camera, or a Low Power Double Data Rate (“LPDDR”) memory unit (“LPDDR3”) 1215 implemented in, for example, LPDDR3 standard. These components may  
35 each be implemented in any suitable manner.

5 [0245] In at least one embodiment, other components may be communicatively coupled to processor 1210 through components discussed above. In at least one embodiment, an accelerometer 1241, an Ambient Light Sensor (“ALS”) 1242, a compass 1243, and a gyroscope 1244 may be communicatively coupled to sensor hub 1240. In at least one embodiment, a thermal sensor 1239, a fan 1237, a keyboard 1236, and a touch pad 1230 may  
 10 be communicatively coupled to EC 1235. In at least one embodiment, a speaker 1263, a headphones 1264, and a microphone (“mic”) 1265 may be communicatively coupled to an audio unit (“audio codec and class d amp”) 1262, which may in turn be communicatively coupled to DSP 1260. In at least one embodiment, audio unit 1262 may include, for example and without limitation, an audio coder/decoder (“codec”) and a class D amplifier. In at least  
 15 one embodiment, a SIM card (“SIM”) 1257 may be communicatively coupled to WWAN unit 1256. In at least one embodiment, components such as WLAN unit 1250 and Bluetooth unit 1252, as well as WWAN unit 1256 may be implemented in a Next Generation Form Factor (“NGFF”).

[0246] FIG. 13 illustrates an exemplary integrated circuit 1300, in accordance with at least  
 20 one embodiment. In at least one embodiment, exemplary integrated circuit 1300 is an SoC that may be fabricated using one or more IP cores. In at least one embodiment, integrated circuit 1300 includes one or more application processor(s) 1305 (e.g., CPUs), at least one graphics processor 1310, and may additionally include an image processor 1315 and/or a video processor 1320, any of which may be a modular IP core. In at least one embodiment,  
 25 integrated circuit 1300 includes peripheral or bus logic including a USB controller 1325, a UART controller 1330, an SPI/SDIO controller 1335, and an I<sup>2</sup>S/I<sup>2</sup>C controller 1340. In at least one embodiment, integrated circuit 1300 can include a display device 1345 coupled to one or more of a high-definition multimedia interface (“HDMI”) controller 1350 and a mobile industry processor interface (“MIPI”) display interface 1355. In at least one  
 30 embodiment, storage may be provided by a flash memory subsystem 1360 including flash memory and a flash memory controller. In at least one embodiment, a memory interface may be provided via a memory controller 1365 for access to SDRAM or SRAM memory devices. In at least one embodiment, some integrated circuits additionally include an embedded security engine 1370.

35 [0247] FIG. 14 illustrates a computing system 1400, according to at least one embodiment; In at least one embodiment, computing system 1400 includes a processing subsystem 1401 having one or more processor(s) 1402 and a system memory 1404 communicating via an

5 interconnection path that may include a memory hub 1405. In at least one embodiment, memory hub 1405 may be a separate component within a chipset component or may be integrated within one or more processor(s) 1402. In at least one embodiment, memory hub 1405 couples with an I/O subsystem 1411 via a communication link 1406. In at least one embodiment, I/O subsystem 1411 includes an I/O hub 1407 that can enable computing  
10 system 1400 to receive input from one or more input device(s) 1408. In at least one embodiment, I/O hub 1407 can enable a display controller, which may be included in one or more processor(s) 1402, to provide outputs to one or more display device(s) 1410A. In at least one embodiment, one or more display device(s) 1410A coupled with I/O hub 1407 can include a local, internal, or embedded display device.

15 **[0248]** In at least one embodiment, processing subsystem 1401 includes one or more parallel processor(s) 1412 coupled to memory hub 1405 via a bus or other communication link 1413. In at least one embodiment, communication link 1413 may be one of any number of standards based communication link technologies or protocols, such as, but not limited to PCIe, or may be a vendor specific communications interface or communications fabric. In at  
20 least one embodiment, one or more parallel processor(s) 1412 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 processor. In at least one embodiment, one or more parallel processor(s) 1412 form a graphics processing subsystem that can output pixels to one of one or more display device(s) 1410A coupled via I/O Hub  
25 1407. In at least one embodiment, one or more parallel processor(s) 1412 can also include a display controller and display interface (not shown) to enable a direct connection to one or more display device(s) 1410B.

**[0249]** In at least one embodiment, a system storage unit 1414 can connect to I/O hub 1407 to provide a storage mechanism for computing system 1400. In at least one embodiment, an  
30 I/O switch 1416 can be used to provide an interface mechanism to enable connections between I/O hub 1407 and other components, such as a network adapter 1418 and/or wireless network adapter 1419 that may be integrated into a platform, and various other devices that can be added via one or more add-in device(s) 1420. In at least one embodiment, network adapter 1418 can be an Ethernet adapter or another wired network adapter. In at least one  
35 embodiment, wireless network adapter 1419 can include one or more of a Wi-Fi, Bluetooth, NFC, or other network device that includes one or more wireless radios.

5 [0250] In at least one embodiment, computing system 1400 can include other components not explicitly shown, including USB or other port connections, optical storage drives, video capture devices, and the like, that may also be connected to I/O hub 1407. In at least one embodiment, communication paths interconnecting various components in FIG. 14 may be implemented using any suitable protocols, such as PCI based protocols (e.g., PCIe), or other  
10 bus or point-to-point communication interfaces and/or protocol(s), such as NVLink high-speed interconnect, or interconnect protocols.

[0251] In at least one embodiment, one or more parallel processor(s) 1412 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  
15 or more parallel processor(s) 1412 incorporate circuitry optimized for general purpose processing. In at least embodiment, components of computing system 1400 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) 1412, memory hub 1405, processor(s) 1402, and I/O hub 1407 can be integrated into an SoC integrated circuit. In at least one  
20 embodiment, components of computing system 1400 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 the components of computing system 1400 can be integrated into a multi-chip module (“MCM”), which can be interconnected with other multi-chip modules into a modular computing system. In at least one embodiment, I/O subsystem 1411 and display  
25 devices 1410B are omitted from computing system 1400.

### Processing Systems

[0252] The following FIGS. set forth, without limitation, exemplary processing systems that can be used to implement at least one embodiment.

[0253] FIG. 15 illustrates an accelerated processing unit (“APU”) 1500, in accordance with at  
30 least one embodiment. In at least one embodiment, APU 1500 is developed by AMD Corporation of Santa Clara, CA. In at least one embodiment, APU 1500 can be configured to execute an application program, such as a CUDA program. In at least one embodiment, APU 1500 includes, without limitation, a core complex 1510, a graphics complex 1540, fabric 1560, I/O interfaces 1570, memory controllers 1580, a display controller 1592, and a  
35 multimedia engine 1594. In at least one embodiment, APU 1500 may include, without limitation, any number of core complexes 1510, any number of graphics complexes 1550, any number of display controllers 1592, and any number of multimedia engines 1594 in any

5 combination. For explanatory purposes, multiple instances of like objects are denoted herein with reference numbers identifying the object and parenthetical numbers identifying the instance where needed.

[0254] In at least one embodiment, core complex 1510 is a CPU, graphics complex 1540 is a GPU, and APU 1500 is a processing unit that integrates, without limitation, 1510 and 1540  
10 onto a single chip. In at least one embodiment, some tasks may be assigned to core complex 1510 and other tasks may be assigned to graphics complex 1540. In at least one embodiment, core complex 1510 is configured to execute main control software associated with APU 1500, such as an operating system. In at least one embodiment, core complex 1510 is the master processor of APU 1500, controlling and coordinating operations of other processors.  
15 In at least one embodiment, core complex 1510 issues commands that control the operation of graphics complex 1540. In at least one embodiment, core complex 1510 can be configured to execute host executable code derived from CUDA source code, and graphics complex 1540 can be configured to execute device executable code derived from CUDA source code.

[0255] In at least one embodiment, core complex 1510 includes, without limitation, cores  
20 1520(1)-1520(4) and an L3 cache 1530. In at least one embodiment, core complex 1510 may include, without limitation, any number of cores 1520 and any number and type of caches in any combination. In at least one embodiment, cores 1520 are configured to execute instructions of a particular instruction set architecture ("ISA"). In at least one embodiment, each core 1520 is a CPU core.

[0256] In at least one embodiment, each core 1520 includes, without limitation, a  
25 fetch/decode unit 1522, an integer execution engine 1524, a floating point execution engine 1526, and an L2 cache 1528. In at least one embodiment, fetch/decode unit 1522 fetches instructions, decodes such instructions, generates micro-operations, and dispatches separate micro-instructions to integer execution engine 1524 and floating point execution engine 1526.  
30 In at least one embodiment, fetch/decode unit 1522 can concurrently dispatch one micro-instruction to integer execution engine 1524 and another micro-instruction to floating point execution engine 1526. In at least one embodiment, integer execution engine 1524 executes, without limitation, integer and memory operations. In at least one embodiment, floating point engine 1526 executes, without limitation, floating point and vector operations. In at least one  
35 embodiment, fetch-decode unit 1522 dispatches micro-instructions to a single execution engine that replaces both integer execution engine 1524 and floating point execution engine 1526.

5 [0257] In at least one embodiment, each core 1520(i), where i is an integer representing a particular instance of core 1520, may access L2 cache 1528(i) included in core 1520(i). In at least one embodiment, each core 1520 included in core complex 1510(j), where j is an integer representing a particular instance of core complex 1510, is connected to other cores 1520 included in core complex 1510(j) via L3 cache 1530(j) included in core complex 1510(j). In  
10 at least one embodiment, cores 1520 included in core complex 1510(j), where j is an integer representing a particular instance of core complex 1510, can access all of L3 cache 1530(j) included in core complex 1510(j). In at least one embodiment, L3 cache 1530 may include, without limitation, any number of slices.

[0258] In at least one embodiment, graphics complex 1540 can be configured to perform  
15 compute operations in a highly-parallel fashion. In at least one embodiment, graphics complex 1540 is configured to execute graphics pipeline operations such as draw commands, pixel operations, geometric computations, and other operations associated with rendering an image to a display. In at least one embodiment, graphics complex 1540 is configured to execute operations unrelated to graphics. In at least one embodiment, graphics complex 1540  
20 is configured to execute both operations related to graphics and operations unrelated to graphics.

[0259] In at least one embodiment, graphics complex 1540 includes, without limitation, any number of compute units 1550 and an L2 cache 1542. In at least one embodiment, compute units 1550 share L2 cache 1542. In at least one embodiment, L2 cache 1542 is partitioned. In  
25 at least one embodiment, graphics complex 1540 includes, without limitation, any number of compute units 1550 and any number (including zero) and type of caches. In at least one embodiment, graphics complex 1540 includes, without limitation, any amount of dedicated graphics hardware.

[0260] In at least one embodiment, each compute unit 1550 includes, without limitation, any  
30 number of SIMD units 1552 and a shared memory 1554. In at least one embodiment, each SIMD unit 1552 implements a SIMD architecture and is configured to perform operations in parallel. In at least one embodiment, each compute unit 1550 may execute any number of thread blocks, but each thread block executes on a single compute unit 1550. In at least one embodiment, a thread block includes, without limitation, any number of threads of execution.  
35 In at least one embodiment, a workgroup is a thread block. In at least one embodiment, each SIMD unit 1552 executes a different warp. In at least one embodiment, a warp is a group of threads (e.g., 16 threads), where each thread in the warp belongs to a single thread block and

5 is configured to process a different set of data based on a single set of instructions. In at least one embodiment, predication can be used to disable one or more threads in a warp. In at least one embodiment, a lane is a thread. In at least one embodiment, a work item is a thread. In at least one embodiment, a wavefront is a warp. In at least one embodiment, different wavefronts in a thread block may synchronize together and communicate via shared memory  
10 1554.

[0261] In at least one embodiment, fabric 1560 is a system interconnect that facilitates data and control transmissions across core complex 1510, graphics complex 1540, I/O interfaces 1570, memory controllers 1580, display controller 1592, and multimedia engine 1594. In at least one embodiment, APU 1500 may include, without limitation, any amount and type of  
15 system interconnect in addition to or instead of fabric 1560 that facilitates data and control transmissions across any number and type of directly or indirectly linked components that may be internal or external to APU 1500. In at least one embodiment, I/O interfaces 1570 are representative of any number and type of I/O interfaces (e.g., PCI, PCI-Extended ("PCI-X"), PCIe, gigabit Ethernet ("GBE"), USB, etc.). In at least one embodiment, various types of  
20 peripheral devices are coupled to I/O interfaces 1570. In at least one embodiment, peripheral devices that are coupled to I/O interfaces 1570 may include, without limitation, keyboards, mice, printers, scanners, joysticks or other types of game controllers, media recording devices, external storage devices, network interface cards, and so forth.

[0262] In at least one embodiment, display controller AMD92 displays images on one or  
25 more display device(s), such as a liquid crystal display ("LCD") device. In at least one embodiment, multimedia engine 240 includes, without limitation, any amount and type of circuitry that is related to multimedia, such as a video decoder, a video encoder, an image signal processor, etc. In at least one embodiment, memory controllers 1580 facilitate data transfers between APU 1500 and a unified system memory 1590. In at least one embodiment,  
30 core complex 1510 and graphics complex 1540 share unified system memory 1590.

[0263] In at least one embodiment, APU 1500 implements a memory subsystem that includes, without limitation, any amount and type of memory controllers 1580 and memory devices (e.g., shared memory 1554) that may be dedicated to one component or shared among multiple components. In at least one embodiment, APU 1500 implements a cache  
35 subsystem that includes, without limitation, one or more cache memories (e.g., L2 caches 1628, L3 cache 1530, and L2 cache 1542) that may each be private to or shared between any

5 number of components (e.g., cores 1520, core complex 1510, SIMD units 1552, compute units 1550, and graphics complex 1540).

[0264] FIG. 16 illustrates a CPU 1600, in accordance with at least one embodiment. In at least one embodiment, CPU 1600 is developed by AMD Corporation of Santa Clara, CA. In at least one embodiment, CPU 1600 can be configured to execute an application program. In  
10 at least one embodiment, CPU 1600 is configured to execute main control software, such as an operating system. In at least one embodiment, CPU 1600 issues commands that control the operation of an external GPU (not shown). In at least one embodiment, CPU 1600 can be configured to execute host executable code derived from CUDA source code, and an external GPU can be configured to execute device executable code derived from such CUDA source  
15 code. In at least one embodiment, CPU 1600 includes, without limitation, any number of core complexes 1610, fabric 1660, I/O interfaces 1670, and memory controllers 1680.

[0265] In at least one embodiment, core complex 1610 includes, without limitation, cores 1620(1)-1620(4) and an L3 cache 1630. In at least one embodiment, core complex 1610 may include, without limitation, any number of cores 1620 and any number and type of caches in  
20 any combination. In at least one embodiment, cores 1620 are configured to execute instructions of a particular ISA. In at least one embodiment, each core 1620 is a CPU core.

[0266] In at least one embodiment, each core 1620 includes, without limitation, a fetch/decode unit 1622, an integer execution engine 1624, a floating point execution engine 1626, and an L2 cache 1628. In at least one embodiment, fetch/decode unit 1622 fetches  
25 instructions, decodes such instructions, generates micro-operations, and dispatches separate micro-instructions to integer execution engine 1624 and floating point execution engine 1626. In at least one embodiment, fetch/decode unit 1622 can concurrently dispatch one micro-instruction to integer execution engine 1624 and another micro-instruction to floating point execution engine 1626. In at least one embodiment, integer execution engine 1624 executes,  
30 without limitation, integer and memory operations. In at least one embodiment, floating point engine 1626 executes, without limitation, floating point and vector operations. In at least one embodiment, fetch-decode unit 1622 dispatches micro-instructions to a single execution engine that replaces both integer execution engine 1624 and floating point execution engine 1626.

35 [0267] In at least one embodiment, each core 1620(i), where i is an integer representing a particular instance of core 1620, may access L2 cache 1628(i) included in core 1620(i). In at

5 least one embodiment, each core 1620 included in core complex 1610(j), where j is an integer representing a particular instance of core complex 1610, is connected to other cores 1620 in core complex 1610(j) via L3 cache 1630(j) included in core complex 1610(j). In at least one embodiment, cores 1620 included in core complex 1610(j), where j is an integer representing a particular instance of core complex 1610, can access all of L3 cache 1630(j) included in  
10 core complex 1610(j). In at least one embodiment, L3 cache 1630 may include, without limitation, any number of slices.

[0268] In at least one embodiment, fabric 1660 is a system interconnect that facilitates data and control transmissions across core complexes 1610(1)-1610(N) (where N is an integer greater than zero), I/O interfaces 1670, and memory controllers 1680. In at least one  
15 embodiment, CPU 1600 may include, without limitation, any amount and type of system interconnect in addition to or instead of fabric 1660 that facilitates data and control transmissions across any number and type of directly or indirectly linked components that may be internal or external to CPU 1600. In at least one embodiment, I/O interfaces 1670 are representative of any number and type of I/O interfaces (e.g., PCI, PCI-X, PCIe, GBE, USB,  
20 etc.). In at least one embodiment, various types of peripheral devices are coupled to I/O interfaces 1670. In at least one embodiment, peripheral devices that are coupled to I/O interfaces 1670 may include, without limitation, displays, keyboards, mice, printers, scanners, joysticks or other types of game controllers, media recording devices, external storage devices, network interface cards, and so forth.

25 [0269] In at least one embodiment, memory controllers 1680 facilitate data transfers between CPU 1600 and a system memory 1690. In at least one embodiment, core complex 1610 and graphics complex 1640 share system memory 1690. In at least one embodiment, CPU 1600 implements a memory subsystem that includes, without limitation, any amount and type of memory controllers 1680 and memory devices that may be dedicated to one  
30 component or shared among multiple components. In at least one embodiment, CPU 1600 implements a cache subsystem that includes, without limitation, one or more cache memories (e.g., L2 caches 1628 and L3 caches 1630) that may each be private to or shared between any number of components (e.g., cores 1620 and core complexes 1610).

[0270] FIG. 17 illustrates an exemplary accelerator integration slice 1790, in accordance with  
35 at least one embodiment. As used herein, a “slice” comprises a specified portion of processing resources of an accelerator integration circuit. In at least one embodiment, the accelerator integration circuit provides cache management, memory access, context

5 management, and interrupt management services on behalf of multiple graphics processing engines included in a graphics acceleration module. The graphics processing engines may each comprise a separate GPU. Alternatively, the graphics processing engines may comprise different types of graphics processing engines within a GPU such as graphics execution units, media processing engines (e.g., video encoders/decoders), samplers, and blit engines. In at  
10 least one embodiment, the graphics acceleration module may be a GPU with multiple graphics processing engines. In at least one embodiment, the graphics processing engines may be individual GPUs integrated on a common package, line card, or chip.

[0271] An application effective address space 1782 within system memory 1714 stores process elements 1783. In one embodiment, process elements 1783 are stored in response to  
15 GPU invocations 1781 from applications 1780 executed on processor 1707. A process element 1783 contains process state for corresponding application 1780. A work descriptor (“WD”) 1784 contained in process element 1783 can be a single job requested by an application or may contain a pointer to a queue of jobs. In at least one embodiment, WD 1784 is a pointer to a job request queue in application effective address space 1782.

20 [0272] Graphics acceleration module 1746 and/or individual graphics processing engines can be shared by all or a subset of processes in a system. In at least one embodiment, an infrastructure for setting up process state and sending WD 1784 to graphics acceleration module 1746 to start a job in a virtualized environment may be included.

[0273] In at least one embodiment, a dedicated-process programming model is  
25 implementation-specific. In this model, a single process owns graphics acceleration module 1746 or an individual graphics processing engine. Because graphics acceleration module 1746 is owned by a single process, a hypervisor initializes an accelerator integration circuit for an owning partition and an operating system initializes accelerator integration circuit for an owning process when graphics acceleration module 1746 is assigned.

30 [0274] In operation, a WD fetch unit 1791 in accelerator integration slice 1790 fetches next WD 1784 which includes an indication of work to be done by one or more graphics processing engines of graphics acceleration module 1746. Data from WD 1784 may be stored in registers 1745 and used by a memory management unit (“MMU”) 1739, interrupt management circuit 1747 and/or context management circuit 1748 as illustrated. For  
35 example, one embodiment of MMU 1739 includes segment/page walk circuitry for accessing segment/page tables 1786 within OS virtual address space 1785. Interrupt management

- 5 circuit 1747 may process interrupt events (“INT”) 1792 received from graphics acceleration module 1746. When performing graphics operations, an effective address 1793 generated by a graphics processing engine is translated to a real address by MMU 1739.

- [0275] In one embodiment, a same set of registers 1745 are duplicated for each graphics processing engine and/or graphics acceleration module 1746 and may be initialized by a hypervisor or operating system. Each of these duplicated registers may be included in accelerator integration slice 1790. Exemplary registers that may be initialized by a hypervisor are shown in Table 1.

**Table 1 –Hypervisor Initialized Registers**

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

- 15 [0276] Exemplary registers that may be initialized by an operating system are shown in Table 2.

**Table 2 –Operating System Initialized Registers**

|   |                                                     |
|---|-----------------------------------------------------|
| 1 | Process and Thread Identification                   |
| 2 | Effective Address (EA) Context Save/Restore Pointer |

|   |                                                             |
|---|-------------------------------------------------------------|
| 3 | Virtual Address (VA) Accelerator Utilization Record Pointer |
| 4 | Virtual Address (VA) Storage Segment Table Pointer          |
| 5 | Authority Mask                                              |
| 6 | Work descriptor                                             |

5

[0277] In one embodiment, each WD 1784 is specific to a particular graphics acceleration module 1746 and/or a particular graphics processing engine. It contains all information required by a graphics processing engine to do work or it can be a pointer to a memory location where an application has set up a command queue of work to be completed.

10 [0278] FIGS. 18A and 18B illustrate exemplary graphics processors, in accordance with at least one embodiment. In at least one embodiment, any of the exemplary graphics processors may be fabricated using one or more IP cores. 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. In at  
15 least one embodiment, the exemplary graphics processors are for use within an SoC.

[0279] FIG. 18A illustrates an exemplary graphics processor 1810 of an SoC integrated circuit that may be fabricated using one or more IP cores, in accordance with at least one embodiment. FIG. 18B illustrates an additional exemplary graphics processor 1840 of an SoC integrated circuit that may be fabricated using one or more IP cores, in accordance with at  
20 least one embodiment. In at least one embodiment, graphics processor 1810 of FIG. 18A is a low power graphics processor core. In at least one embodiment, graphics processor 1840 of FIG. 18B is a higher performance graphics processor core. In at least one embodiment, each of graphics processors 1810, 1840 can be variants of graphics processor 1310 of FIG. 13.

[0280] In at least one embodiment, graphics processor 1810 includes a vertex processor 1805  
25 and one or more fragment processor(s) 1815A-1815N (e.g., 1815A, 1815B, 1815C, 1815D, through 1815N-1, and 1815N). In at least one embodiment, graphics processor 1810 can execute different shader programs via separate logic, such that vertex processor 1805 is optimized to execute operations for vertex shader programs, while one or more fragment processor(s) 1815A-1815N execute fragment (e.g., pixel) shading operations for fragment or  
30 pixel shader programs. In at least one embodiment, vertex processor 1805 performs a vertex

5 processing stage of a 3D graphics pipeline and generates primitives and vertex data. In at least one embodiment, fragment processor(s) 1815A-1815N use primitive and vertex data generated by vertex processor 1805 to produce a framebuffer that is displayed on a display device. In at least one embodiment, fragment processor(s) 1815A-1815N are optimized to execute fragment shader programs as provided for in an OpenGL API, which may be used to  
10 perform similar operations as a pixel shader program as provided for in a Direct 3D API.

[0281] In at least one embodiment, graphics processor 1810 additionally includes one or more MMU(s) 1820A-1820B, cache(s) 1825A-1825B, and circuit interconnect(s) 1830A-1830B. In at least one embodiment, one or more MMU(s) 1820A-1820B provide for virtual to physical address mapping for graphics processor 1810, including for vertex processor 1805  
15 and/or fragment processor(s) 1815A-1815N, 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) 1825A-1825B. In at least one embodiment, one or more MMU(s) 1820A-1820B may be synchronized with other MMUs within a system, including one or more MMUs associated with one or more application processor(s) 1305, image processors 1315, and/or  
20 video processors 1320 of FIG. 13, such that each processor 1305-1320 can participate in a shared or unified virtual memory system. In at least one embodiment, one or more circuit interconnect(s) 1830A-1830B enable graphics processor 1810 to interface with other IP cores within an SoC, either via an internal bus of the SoC or via a direct connection.

[0282] In at least one embodiment, graphics processor 1840 includes one or more MMU(s)  
25 1820A-1820B, caches 1825A-1825B, and circuit interconnects 1830A-1830B of graphics processor 1810 of FIG. 18A. In at least one embodiment, graphics processor 1840 includes one or more shader core(s) 1855A-1855N (e.g., 1855A, 1855B, 1855C, 1855D, 1855E, 1855F, through 1855N-1, and 1855N), 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,  
30 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 1840 includes an inter-core task manager 1845, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores 1855A-1855N and a tiling unit 1858 to accelerate tiling operations for tile-based rendering, in which  
35 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.

5     **[0283]** FIG. 19A illustrates a graphics core 1900, in accordance with at least one embodiment. In at least one embodiment, graphics core 1900 may be included within graphics processor 1310 of FIG. 13. In at least one embodiment, graphics core 1900 may be a unified shader core 1855A-1855N as in FIG. 18B. In at least one embodiment, graphics core 1900 includes a shared instruction cache 1902, a texture unit 1918, and a cache/shared  
 10     memory 1920 that are common to execution resources within graphics core 1900. In at least one embodiment, graphics core 1900 can include multiple slices 1901A-1901N or partition for each core, and a graphics processor can include multiple instances of graphics core 1900. Slices 1901A-1901N can include support logic including a local instruction cache 1904A-1904N, a thread scheduler 1906A-1906N, a thread dispatcher 1908A-1908N, and a set of  
 15     registers 1910A-1910N. In at least one embodiment, slices 1901A-1901N can include a set of additional function units (“AFUs”) 1912A-1912N, floating-point units (“FPUs”) 1914A-1914N, integer arithmetic logic units (“ALUs”) 1916-1916N, address computational units (“ACUs”) 1913A-1913N, double-precision floating-point units (“DPFPUs”) 1915A-1915N, and matrix processing units (“MPUs”) 1917A-1917N.

20     **[0284]** In at least one embodiment, FPUs 1914A-1914N can perform single-precision (32-bit) and half-precision (16-bit) floating point operations, while DPFPUs 1915A-1915N perform double precision (64-bit) floating point operations. In at least one embodiment, ALUs 1916A-1916N can perform variable precision integer operations at 8-bit, 16-bit, and 32-bit precision, and can be configured for mixed precision operations. In at least one  
 25     embodiment, MPUs 1917A-1917N can also be configured for mixed precision matrix operations, including half-precision floating point and 8-bit integer operations. In at least one embodiment, MPUs 1917-1917N can perform a variety of matrix operations to accelerate CUDA programs, including enabling support for accelerated general matrix to matrix multiplication (“GEMM”). In at least one embodiment, AFUs 1912A-1912N can perform  
 30     additional logic operations not supported by floating-point or integer units, including trigonometric operations (e.g., Sine, Cosine, etc.).

35     **[0285]** FIG. 19B illustrates a general-purpose graphics processing unit (“GPGPU”) 1930, in accordance with at least one embodiment. In at least one embodiment, GPGPU 1930 is highly-parallel and suitable for deployment on a multi-chip module. In at least one embodiment, GPGPU 1930 can be configured to enable highly-parallel compute operations to be performed by an array of GPUs. In at least one embodiment, GPGPU 1930 can be linked directly to other instances of GPGPU 1930 to create a multi-GPU cluster to improve

5 execution time for CUDA programs. In at least one embodiment, GPGPU 1930 includes a host interface 1932 to enable a connection with a host processor. In at least one embodiment, host interface 1932 is a PCIe interface. In at least one embodiment, host interface 1932 can be a vendor specific communications interface or communications fabric. In at least one embodiment, GPGPU 1930 receives commands from a host processor and uses a global  
10 scheduler 1934 to distribute execution threads associated with those commands to a set of compute clusters 1936A-1936H. In at least one embodiment, compute clusters 1936A-1936H share a cache memory 1938. In at least one embodiment, cache memory 1938 can serve as a higher-level cache for cache memories within compute clusters 1936A-1936H.

[0286] In at least one embodiment, GPGPU 1930 includes memory 1944A-1944B coupled  
15 with compute clusters 1936A-1936H via a set of memory controllers 1942A-1942B. In at least one embodiment, memory 1944A-1944B can include various types of memory devices including DRAM or graphics random access memory, such as synchronous graphics random access memory (“SGRAM”), including graphics double data rate (“GDDR”) memory.

[0287] In at least one embodiment, compute clusters 1936A-1936H each include a set of  
20 graphics cores, such as graphics core 1900 of FIG. 19A, 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 computations associated with CUDA programs. For example, in at least one embodiment, at least a subset of floating point units in each of compute clusters 1936A-1936H can be configured to perform 16-bit or 32-bit floating point  
25 operations, while a different subset of floating point units can be configured to perform 64-bit floating point operations.

[0288] In at least one embodiment, multiple instances of GPGPU 1930 can be configured to operate as a compute cluster. Compute clusters 1936A-1936H may implement any technically feasible communication techniques for synchronization and data exchange. In at  
30 least one embodiment, multiple instances of GPGPU 1930 communicate over host interface 1932. In at least one embodiment, GPGPU 1930 includes an I/O hub 1939 that couples GPGPU 1930 with a GPU link 1940 that enables a direct connection to other instances of GPGPU 1930. In at least one embodiment, GPU link 1940 is coupled to a dedicated GPU-to-GPU bridge that enables communication and synchronization between multiple instances of  
35 GPGPU 1930. In at least one embodiment GPU link 1940 couples with a high speed interconnect to transmit and receive data to other GPGPUs 1930 or parallel processors. In at least one embodiment, multiple instances of GPGPU 1930 are located in separate data

5 processing systems and communicate via a network device that is accessible via host interface 1932. In at least one embodiment GPU link 1940 can be configured to enable a connection to a host processor in addition to or as an alternative to host interface 1932. In at least one embodiment, GPGPU 1930 can be configured to execute a CUDA program.

[0289] FIG. 20A illustrates a parallel processor 2000, in accordance with at least one  
10 embodiment. In at least one embodiment, various components of parallel processor 2000 may be implemented using one or more integrated circuit devices, such as programmable processors, application specific integrated circuits (“ASICs”), or FPGAs.

[0290] In at least one embodiment, parallel processor 2000 includes a parallel processing unit 2002. In at least one embodiment, parallel processing unit 2002 includes an I/O unit 2004 that  
15 enables communication with other devices, including other instances of parallel processing unit 2002. In at least one embodiment, I/O unit 2004 may be directly connected to other devices. In at least one embodiment, I/O unit 2004 connects with other devices via use of a hub or switch interface, such as memory hub 2005. In at least one embodiment, connections between memory hub 2005 and I/O unit 2004 form a communication link. In at least one  
20 embodiment, I/O unit 2004 connects with a host interface 2006 and a memory crossbar 2016, where host interface 2006 receives commands directed to performing processing operations and memory crossbar 2016 receives commands directed to performing memory operations.

[0291] In at least one embodiment, when host interface 2006 receives a command buffer via I/O unit 2004, host interface 2006 can direct work operations to perform those commands to a  
25 front end 2008. In at least one embodiment, front end 2008 couples with a scheduler 2010, which is configured to distribute commands or other work items to a processing array 2012. In at least one embodiment, scheduler 2010 ensures that processing array 2012 is properly configured and in a valid state before tasks are distributed to processing array 2012. In at least one embodiment, scheduler 2010 is implemented via firmware logic executing on a  
30 microcontroller. In at least one embodiment, microcontroller implemented scheduler 2010 is configurable to perform complex scheduling and work distribution operations at coarse and fine granularity, enabling rapid preemption and context switching of threads executing on processing array 2012. In at least one embodiment, host software can prove workloads for scheduling on processing array 2012 via one of multiple graphics processing doorbells. In at  
35 least one embodiment, workloads can then be automatically distributed across processing array 2012 by scheduler 2010 logic within a microcontroller including scheduler 2010.

5 [0292] In at least one embodiment, processing array 2012 can include up to “N” clusters (e.g., cluster 2014A, cluster 2014B, through cluster 2014N). In at least one embodiment, each cluster 2014A-2014N of processing array 2012 can execute a large number of concurrent threads. In at least one embodiment, scheduler 2010 can allocate work to clusters 2014A-2014N of processing array 2012 using various scheduling and/or work distribution  
10 algorithms, which may vary depending on the workload arising for each type of program or computation. In at least one embodiment, scheduling can be handled dynamically by scheduler 2010, or can be assisted in part by compiler logic during compilation of program logic configured for execution by processing array 2012. In at least one embodiment, different clusters 2014A-2014N of processing array 2012 can be allocated for processing  
15 different types of programs or for performing different types of computations.

[0293] In at least one embodiment, processing array 2012 can be configured to perform various types of parallel processing operations. In at least one embodiment, processing array 2012 is configured to perform general-purpose parallel compute operations. For example, in at least one embodiment, processing array 2012 can include logic to execute processing tasks  
20 including filtering of video and/or audio data, performing modeling operations, including physics operations, and performing data transformations.

[0294] In at least one embodiment, processing array 2012 is configured to perform parallel graphics processing operations. In at least one embodiment, processing array 2012 can include additional logic to support execution of such graphics processing operations,  
25 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 array 2012 can be configured to execute graphics processing related shader programs such as, but not limited to vertex shaders, tessellation shaders, geometry shaders, and pixel shaders. In at least one embodiment, parallel processing unit 2002 can transfer data from system memory  
30 via I/O unit 2004 for processing. In at least one embodiment, during processing, transferred data can be stored to on-chip memory (e.g., a parallel processor memory 2022) during processing, then written back to system memory.

[0295] In at least one embodiment, when parallel processing unit 2002 is used to perform graphics processing, scheduler 2010 can be configured to divide a processing workload into  
35 approximately equal sized tasks, to better enable distribution of graphics processing operations to multiple clusters 2014A-2014N of processing array 2012. In at least one embodiment, portions of processing array 2012 can be configured to perform different types

5 of processing. For example, in at least one embodiment, a first portion may be configured to perform vertex shading and topology generation, a second portion may be configured to perform tessellation and geometry shading, and a third portion may be configured to perform pixel shading or other screen space operations, to produce a rendered image for display. In at least one embodiment, intermediate data produced by one or more of clusters 2014A-2014N  
10 may be stored in buffers to allow intermediate data to be transmitted between clusters 2014A-2014N for further processing.

[0296] In at least one embodiment, processing array 2012 can receive processing tasks to be executed via scheduler 2010, which receives commands defining processing tasks from front end 2008. In at least one embodiment, processing tasks can include indices of data to be  
15 processed, e.g., surface (patch) data, primitive data, vertex data, and/or pixel data, as well as state parameters and commands defining how data is to be processed (e.g., what program is to be executed). In at least one embodiment, scheduler 2010 may be configured to fetch indices corresponding to tasks or may receive indices from front end 2008. In at least one embodiment, front end 2008 can be configured to ensure processing array 2012 is configured  
20 to a valid state before a workload specified by incoming command buffers (e.g., batch-buffers, push buffers, etc.) is initiated.

[0297] In at least one embodiment, each of one or more instances of parallel processing unit 2002 can couple with parallel processor memory 2022. In at least one embodiment, parallel processor memory 2022 can be accessed via memory crossbar 2016, which can receive  
25 memory requests from processing array 2012 as well as I/O unit 2004. In at least one embodiment, memory crossbar 2016 can access parallel processor memory 2022 via a memory interface 2018. In at least one embodiment, memory interface 2018 can include multiple partition units (e.g., a partition unit 2020A, partition unit 2020B, through partition unit 2020N) that can each couple to a portion (e.g., memory unit) of parallel processor  
30 memory 2022. In at least one embodiment, a number of partition units 2020A-2020N is configured to be equal to a number of memory units, such that a first partition unit 2020A has a corresponding first memory unit 2024A, a second partition unit 2020B has a corresponding memory unit 2024B, and an Nth partition unit 2020N has a corresponding Nth memory unit 2024N. In at least one embodiment, a number of partition units 2020A-2020N may not be  
35 equal to a number of memory devices.

[0298] In at least one embodiment, memory units 2024A-2024N can include various types of memory devices, including DRAM or graphics random access memory, such as SGRAM,

5 including GDDR memory. In at least one embodiment, memory units 2024A-2024N may also include 3D stacked memory, including but not limited to high bandwidth memory (“HBM”). In at least one embodiment, render targets, such as frame buffers or texture maps may be stored across memory units 2024A-2024N, allowing partition units 2020A-2020N to write portions of each render target in parallel to efficiently use available bandwidth of parallel processor memory 2022. In at least one embodiment, a local instance of parallel processor memory 2022 may be excluded in favor of a unified memory design that utilizes system memory in conjunction with local cache memory.

[0299] In at least one embodiment, any one of clusters 2014A-2014N of processing array 2012 can process data that will be written to any of memory units 2024A-2024N within parallel processor memory 2022. In at least one embodiment, memory crossbar 2016 can be configured to transfer an output of each cluster 2014A-2014N to any partition unit 2020A-2020N or to another cluster 2014A-2014N, which can perform additional processing operations on an output. In at least one embodiment, each cluster 2014A-2014N can communicate with memory interface 2018 through memory crossbar 2016 to read from or write to various external memory devices. In at least one embodiment, memory crossbar 2016 has a connection to memory interface 2018 to communicate with I/O unit 2004, as well as a connection to a local instance of parallel processor memory 2022, enabling processing units within different clusters 2014A-2014N to communicate with system memory or other memory that is not local to parallel processing unit 2002. In at least one embodiment, memory crossbar 2016 can use virtual channels to separate traffic streams between clusters 2014A-2014N and partition units 2020A-2020N.

[0300] In at least one embodiment, multiple instances of parallel processing unit 2002 can be provided on a single add-in card, or multiple add-in cards can be interconnected. In at least one embodiment, different instances of parallel processing unit 2002 can be configured to 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 2002 can include higher precision floating point units relative to other instances. In at least one embodiment, systems incorporating one or more instances of parallel processing unit 2002 or parallel processor 2000 can be implemented in a variety of configurations and form factors, including but not limited to desktop, laptop, or handheld personal computers, servers, workstations, game consoles, and/or embedded systems.

5     **[0301]** FIG. 20B illustrates a processing cluster 2094, in accordance with at least one embodiment. In at least one embodiment, processing cluster 2094 is included within a parallel processing unit. In at least one embodiment, processing cluster 2094 is one of processing clusters 2014A-2014N of FIG. 20. In at least one embodiment, processing cluster 2094 can be configured to execute many threads in parallel, where the term “thread” refers to  
10     an instance of a particular program executing on a particular set of input data. In at least one embodiment, single instruction, multiple data (“SIMD”) instruction issue techniques are used to support parallel execution of a large number of threads without providing multiple independent instruction units. In at least one embodiment, single instruction, multiple thread (“SIMT”) techniques are used to support parallel execution of a large number of generally  
15     synchronized threads, using a common instruction unit configured to issue instructions to a set of processing engines within each processing cluster 2094.

**[0302]** In at least one embodiment, operation of processing cluster 2094 can be controlled via a pipeline manager 2032 that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager 2032 receives instructions from scheduler 2010 of  
20     FIG. 20 and manages execution of those instructions via a graphics multiprocessor 2034 and/or a texture unit 2036. In at least one embodiment, graphics multiprocessor 2034 is an exemplary instance of a SIMT parallel processor. However, in at least one embodiment, various types of SIMT parallel processors of differing architectures may be included within processing cluster 2094. In at least one embodiment, one or more instances of graphics  
25     multiprocessor 2034 can be included within processing cluster 2094. In at least one embodiment, graphics multiprocessor 2034 can process data and a data crossbar 2040 can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager 2032 can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data  
30     crossbar 2040.

**[0303]** In at least one embodiment, each graphics multiprocessor 2034 within processing cluster 2094 can include an identical set of functional execution logic (e.g., arithmetic logic units, load/store units (“LSUs”), etc.). In at least one embodiment, functional execution logic can be configured in a pipelined manner in which new instructions can be issued before  
35     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.

5 In at least one embodiment, same functional-unit hardware can be leveraged to perform different operations and any combination of functional units may be present.

[0304] In at least one embodiment, instructions transmitted to processing cluster 2094 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  
10 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 graphics multiprocessor 2034. In at least one embodiment, a thread group may include fewer threads than a number of processing engines within graphics multiprocessor 2034. In at least one embodiment, when a thread group includes fewer threads than a number of processing engines, one or more of the  
15 processing engines may be idle during cycles in which that thread group is being processed. In at least one embodiment, a thread group may also include more threads than a number of processing engines within graphics multiprocessor 2034. In at least one embodiment, when a thread group includes more threads than the number of processing engines within graphics multiprocessor 2034, processing can be performed over consecutive clock cycles. In at least  
20 one embodiment, multiple thread groups can be executed concurrently on graphics multiprocessor 2034.

[0305] In at least one embodiment, graphics multiprocessor 2034 includes an internal cache memory to perform load and store operations. In at least one embodiment, graphics multiprocessor 2034 can forego an internal cache and use a cache memory (e.g., L1 cache  
25 2048) within processing cluster 2094. In at least one embodiment, each graphics multiprocessor 2034 also has access to Level 2 (“L2”) caches within partition units (e.g., partition units 2020A-2020N of FIG. 20A) that are shared among all processing clusters 2094 and may be used to transfer data between threads. In at least one embodiment, graphics multiprocessor 2034 may also access off-chip global memory, which can include one or more  
30 of local parallel processor memory and/or system memory. In at least one embodiment, any memory external to parallel processing unit 2002 may be used as global memory. In at least one embodiment, processing cluster 2094 includes multiple instances of graphics multiprocessor 2034 that can share common instructions and data, which may be stored in L1 cache 2048.

35 [0306] In at least one embodiment, each processing cluster 2094 may include an MMU 2045 that is configured to map virtual addresses into physical addresses. In at least one embodiment, one or more instances of MMU 2045 may reside within memory interface 2018

5 of FIG. 20. In at least one embodiment, MMU 2045 includes a set of page table entries (“PTEs”) used to map a virtual address to a physical address of a tile and optionally a cache line index. In at least one embodiment, MMU 2045 may include address translation lookaside buffers (“TLBs”) or caches that may reside within graphics multiprocessor 2034 or L1 cache 2048 or processing cluster 2094. In at least one embodiment, a physical address is processed  
10 to distribute surface data access locality to allow efficient request interleaving among partition units. In at least one embodiment, a cache line index may be used to determine whether a request for a cache line is a hit or miss.

[0307] In at least one embodiment, processing cluster 2094 may be configured such that each graphics multiprocessor 2034 is coupled to a texture unit 2036 for performing texture  
15 mapping operations, e.g., determining texture sample positions, reading texture data, and filtering texture data. In at least one embodiment, texture data is read from an internal texture L1 cache (not shown) or from an L1 cache within graphics multiprocessor 2034 and is fetched from an L2 cache, local parallel processor memory, or system memory, as needed. In at least one embodiment, each graphics multiprocessor 2034 outputs a processed task to data  
20 crossbar 2040 to provide the processed task to another processing cluster 2094 for further processing or to store the processed task in an L2 cache, a local parallel processor memory, or a system memory via memory crossbar 2016. In at least one embodiment, a pre-raster operations unit (“preROP”) 2042 is configured to receive data from graphics multiprocessor 2034, direct data to ROP units, which may be located with partition units as described herein  
25 (e.g., partition units 2020A-2020N of FIG. 20). In at least one embodiment, PreROP 2042 can perform optimizations for color blending, organize pixel color data, and perform address translations.

[0308] FIG. 20C illustrates a graphics multiprocessor 2096, in accordance with at least one embodiment. In at least one embodiment, graphics multiprocessor 2096 is graphics  
30 multiprocessor 2034 of FIG. 20B. In at least one embodiment, graphics multiprocessor 2096 couples with pipeline manager 2032 of processing cluster 2094. In at least one embodiment, graphics multiprocessor 2096 has an execution pipeline including but not limited to an instruction cache 2052, an instruction unit 2054, an address mapping unit 2056, a register file 2058, one or more GPGPU cores 2062, and one or more LSUs 2066. GPGPU cores 2062 and  
35 LSUs 2066 are coupled with cache memory 2072 and shared memory 2070 via a memory and cache interconnect 2068.

5     **[0309]** In at least one embodiment, instruction cache 2052 receives a stream of instructions to execute from pipeline manager 2032. In at least one embodiment, instructions are cached in instruction cache 2052 and dispatched for execution by instruction unit 2054. In at least one embodiment, instruction unit 2054 can dispatch instructions as thread groups (e.g., warps), with each thread of a thread group assigned to a different execution unit within GPGPU core 10     2062. In at least one embodiment, an instruction can access any of a local, shared, or global address space by specifying an address within a unified address space. In at least one embodiment, address mapping unit 2056 can be used to translate addresses in a unified address space into a distinct memory address that can be accessed by LSUs 2066.

15     **[0310]** In at least one embodiment, register file 2058 provides a set of registers for functional units of graphics multiprocessor 2096. In at least one embodiment, register file 2058 provides temporary storage for operands connected to data paths of functional units (e.g., GPGPU cores 2062, LSUs 2066) of graphics multiprocessor 2096. In at least one embodiment, register file 2058 is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file 2058. In at least one embodiment, register file 20     2058 is divided between different thread groups being executed by graphics multiprocessor 2096.

25     **[0311]** In at least one embodiment, GPGPU cores 2062 can each include FPUs and/or integer ALUs that are used to execute instructions of graphics multiprocessor 2096. GPGPU cores 2062 can be similar in architecture or can differ in architecture. In at least one embodiment, a first portion of GPGPU cores 2062 include a single precision FPU and an integer ALU while a second portion of GPGPU cores 2062 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 2096 can additionally include one or more fixed function or special function 30     units to perform specific functions such as copy rectangle or pixel blending operations. In at least one embodiment one or more of GPGPU cores 2062 can also include fixed or special function logic.

35     **[0312]** In at least one embodiment, GPGPU cores 2062 include SIMD logic capable of performing a single instruction on multiple sets of data. In at least one embodiment GPGPU cores 2062 can physically execute SIMD4, SIMD8, and SIMD16 instructions and logically execute SIMD1, SIMD2, and SIMD32 instructions. In at least one embodiment, SIMD instructions for GPGPU cores 2062 can be generated at compile time by a shader compiler or

- 5 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 the same or similar operations can be executed in parallel via a single SIMD8 logic unit.
- 10 **[0313]** In at least one embodiment, memory and cache interconnect 2068 is an interconnect network that connects each functional unit of graphics multiprocessor 2096 to register file 2058 and to shared memory 2070. In at least one embodiment, memory and cache interconnect 2068 is a crossbar interconnect that allows LSU 2066 to implement load and store operations between shared memory 2070 and register file 2058. In at least one
- 15 embodiment, register file 2058 can operate at a same frequency as GPGPU cores 2062, thus data transfer between GPGPU cores 2062 and register file 2058 is very low latency. In at least one embodiment, shared memory 2070 can be used to enable communication between threads that execute on functional units within graphics multiprocessor 2096. In at least one embodiment, cache memory 2072 can be used as a data cache for example, to cache texture
- 20 data communicated between functional units and texture unit 2036. In at least one embodiment, shared memory 2070 can also be used as a program managed cache. In at least one embodiment, threads executing on GPGPU cores 2062 can programmatically store data within shared memory in addition to automatically cached data that is stored within cache memory 2072.
- 25 **[0314]** In at least one embodiment, a parallel processor or GPGPU as described herein is communicatively coupled to host/processor cores to accelerate graphics operations, machine-learning operations, pattern analysis operations, and various general purpose GPU (GPGPU) functions. In at least one embodiment, a GPU may be communicatively coupled to host processor/cores over a bus or other interconnect (e.g., a high speed interconnect such as PCIe
- 30 or NVLink). In at least one embodiment, a GPU may be integrated on the same package or chip as cores and communicatively coupled to cores over a processor bus/interconnect that is internal to a package or a chip. In at least one embodiment, regardless of the manner in which a GPU is connected, processor cores may allocate work to the GPU in the form of sequences of commands/instructions contained in a WD. In at least one embodiment, the GPU then uses
- 35 dedicated circuitry/logic for efficiently processing these commands/instructions.
- [0315]** FIG. 21 illustrates a graphics processor 2100, in accordance with at least one embodiment. In at least one embodiment, graphics processor 2100 includes a ring

- 5 interconnect 2102, a pipeline front-end 2104, a media engine 2137, and graphics cores 2180A-2180N. In at least one embodiment, ring interconnect 2102 couples graphics processor 2100 to other processing units, including other graphics processors or one or more general-purpose processor cores. In at least one embodiment, graphics processor 2100 is one of many processors integrated within a multi-core processing system.
- 10 **[0316]** In at least one embodiment, graphics processor 2100 receives batches of commands via ring interconnect 2102. In at least one embodiment, incoming commands are interpreted by a command streamer 2103 in pipeline front-end 2104. In at least one embodiment, graphics processor 2100 includes scalable execution logic to perform 3D geometry processing and media processing via graphics core(s) 2180A-2180N. In at least one
- 15 embodiment, for 3D geometry processing commands, command streamer 2103 supplies commands to geometry pipeline 2136. In at least one embodiment, for at least some media processing commands, command streamer 2103 supplies commands to a video front end 2134, which couples with a media engine 2137. In at least one embodiment, media engine 2137 includes a Video Quality Engine (“VQE”) 2130 for video and image post-processing
- 20 and a multi-format encode/decode (“MFX”) engine 2133 to provide hardware-accelerated media data encode and decode. In at least one embodiment, geometry pipeline 2136 and media engine 2137 each generate execution threads for thread execution resources provided by at least one graphics core 2180A.
- [0317]** In at least one embodiment, graphics processor 2100 includes scalable thread
- 25 execution resources featuring modular graphics cores 2180A-2180N (sometimes referred to as core slices), each having multiple sub-cores 2150A-2150N, 2160A-2160N (sometimes referred to as core sub-slices). In at least one embodiment, graphics processor 2100 can have any number of graphics cores 2180A through 2180N. In at least one embodiment, graphics processor 2100 includes a graphics core 2180A having at least a first sub-core 2150A and a
- 30 second sub-core 2160A. In at least one embodiment, graphics processor 2100 is a low power processor with a single sub-core (e.g., sub-core 2150A). In at least one embodiment, graphics processor 2100 includes multiple graphics cores 2180A-2180N, each including a set of first sub-cores 2150A-2150N and a set of second sub-cores 2160A-2160N. In at least one
- embodiment, each sub-core in first sub-cores 2150A-2150N includes at least a first set of
- 35 execution units (“EUs”) 2152A-2152N and media/texture samplers 2154A-2154N. In at least one embodiment, each sub-core in second sub-cores 2160A-2160N includes at least a second set of execution units 2162A-2162N and samplers 2164A-2164N. In at least one

5     embodiment, each sub-core 2150A-2150N, 2160A-2160N shares a set of shared resources 2170A-2170N. In at least one embodiment, shared resources 2170 include shared cache memory and pixel operation logic.

[0318] FIG. 22 illustrates a processor 2200, in accordance with at least one embodiment. In at least one embodiment, processor 2200 may include, without limitation, logic circuits to  
10     perform instructions. In at least one embodiment, processor 2200 may perform instructions, including x86 instructions, ARM instructions, specialized instructions for ASICs, etc. In at least one embodiment, processor 2210 may include registers to store packed data, such as 64-bit wide MMXTM registers in microprocessors enabled with MMX technology from Intel Corporation of Santa Clara, Calif. In at least one embodiment, MMX registers, available in  
15     both integer and floating point forms, may operate with packed data elements that accompany 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 2210 may perform instructions to accelerate CUDA programs.

20     [0319] In at least one embodiment, processor 2200 includes an in-order front end (“front end”) 2201 to fetch instructions to be executed and prepare instructions to be used later in processor pipeline. In at least one embodiment, front end 2201 may include several units. In at least one embodiment, an instruction prefetcher 2226 fetches instructions from memory and feeds instructions to an instruction decoder 2228 which in turn decodes or interprets  
25     instructions. For example, in at least one embodiment, instruction decoder 2228 decodes a received instruction into one or more operations called “micro-instructions” or “micro-operations” (also called “micro ops” or “uops”) for execution. In at least one embodiment, instruction decoder 2228 parses instruction into an opcode and corresponding data and control fields that may be used by micro-architecture to perform operations. In at least one  
30     embodiment, a trace cache 2230 may assemble decoded uops into program ordered sequences or traces in a uop queue 2234 for execution. In at least one embodiment, when trace cache 2230 encounters a complex instruction, a microcode ROM 2232 provides uops needed to complete an operation.

[0320] In at least one embodiment, some instructions may be converted into a single micro-  
35     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 2228 may access microcode ROM 2232 to perform instruction. In at least one

5     embodiment, an instruction may be decoded into a small number of micro-ops for processing at instruction decoder 2228. In at least one embodiment, an instruction may be stored within microcode ROM 2232 should a number of micro-ops be needed to accomplish operation. In at least one embodiment, trace cache 2230 refers to an entry point programmable logic array (“PLA”) to determine a correct micro-instruction pointer for reading microcode sequences to  
10    complete one or more instructions from microcode ROM 2232. In at least one embodiment, after microcode ROM 2232 finishes sequencing micro-ops for an instruction, front end 2201 of machine may resume fetching micro-ops from trace cache 2230.

[0321] In at least one embodiment, out-of-order execution engine (“out of order engine”) 2203 may prepare instructions for execution. In at least one embodiment, out-of-order  
15    execution logic has a number of buffers to smooth out and re-order the flow of instructions to optimize performance as they go down a pipeline and get scheduled for execution. Out-of-order execution engine 2203 includes, without limitation, an allocator/register renamer 2240, a memory uop queue 2242, an integer/floating point uop queue 2244, a memory scheduler 2246, a fast scheduler 2202, a slow/general floating point scheduler (“slow/general FP  
20    scheduler”) 2204, and a simple floating point scheduler (“simple FP scheduler”) 2206. In at least one embodiment, fast scheduler 2202, slow/general floating point scheduler 2204, and simple floating point scheduler 2206 are also collectively referred to herein as “uop schedulers 2202, 2204, 2206.” Allocator/register renamer 2240 allocates machine buffers and resources that each uop needs in order to execute. In at least one embodiment,  
25    allocator/register renamer 2240 renames logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer 2240 also allocates an entry for each uop in one of two uop queues, memory uop queue 2242 for memory operations and integer/floating point uop queue 2244 for non-memory operations, in front of memory scheduler 2246 and uop schedulers 2202, 2204, 2206. In at least one embodiment, uop schedulers 2202, 2204,  
30    2206, 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 2202 of at least one embodiment may schedule on each half of main clock cycle while slow/general floating point scheduler 2204 and simple floating point scheduler 2206 may schedule once per main processor clock cycle.  
35    In at least one embodiment, uop schedulers 2202, 2204, 2206 arbitrate for dispatch ports to schedule uops for execution.

5     **[0322]** In at least one embodiment, execution block 2211 includes, without limitation, an integer register file/bypass network 2208, a floating point register file/bypass network (“FP register file/bypass network”) 2210, address generation units (“AGUs”) 2212 and 2214, fast ALUs 2216 and 2218, a slow ALU 2220, a floating point ALU (“FP”) 2222, and a floating point move unit (“FP move”) 2224. In at least one embodiment, integer register file/bypass network 2208 and floating point register file/bypass network 2210 are also referred to herein as “register files 2208, 2210.” In at least one embodiment, AGUs 2212 and 2214, fast ALUs 2216 and 2218, slow ALU 2220, floating point ALU 2222, and floating point move unit 2224 are also referred to herein as “execution units 2212, 2214, 2216, 2218, 2220, 2222, and 2224.” In at least one embodiment, an execution block may include, without limitation, any number (including zero) and type of register files, bypass networks, address generation units, and execution units, in any combination.

10     **[0323]** In at least one embodiment, register files 2208, 2210 may be arranged between uop schedulers 2202, 2204, 2206, and execution units 2212, 2214, 2216, 2218, 2220, 2222, and 2224. In at least one embodiment, integer register file/bypass network 2208 performs integer operations. In at least one embodiment, floating point register file/bypass network 2210 performs floating point operations. In at least one embodiment, each of register files 2208, 2210 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 2208, 2210 may communicate data with each other. In at least one embodiment, integer register file/bypass network 2208 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 2210 may include, without limitation, 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

25     **[0324]** In at least one embodiment, execution units 2212, 2214, 2216, 2218, 2220, 2222, 2224 may execute instructions. In at least one embodiment, register files 2208, 2210 store integer and floating point data operand values that micro-instructions need to execute. In at least one embodiment, processor 2200 may include, without limitation, any number and combination of execution units 2212, 2214, 2216, 2218, 2220, 2222, 2224. In at least one embodiment, floating point ALU 2222 and floating point move unit 2224 may execute floating point, MMX, SIMD, AVX and SSE, or other operations. In at least one embodiment,

5 floating point ALU 2222 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 2216, 2218. In at least one embodiment, fast ALUS 2216, 2218 may execute fast operations with an effective  
10 latency of half a clock cycle. In at least one embodiment, most complex integer operations go to slow ALU 2220 as slow ALU 2220 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 2212, 2214. In at least one embodiment, fast ALU 2216, fast ALU 2218,  
15 and slow ALU 2220 may perform integer operations on 64-bit data operands. In at least one embodiment, fast ALU 2216, fast ALU 2218, and slow ALU 2220 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 2222 and floating point move unit 2224 may be implemented to support a range of operands having bits of various widths. In at least one  
20 embodiment, floating point ALU 2222 and floating point move unit 2224 may operate on 128-bit wide packed data operands in conjunction with SIMD and multimedia instructions.

[0325] In at least one embodiment, uop schedulers 2202, 2204, 2206 dispatch dependent operations before parent load has finished executing. In at least one embodiment, as uops may be speculatively scheduled and executed in processor 2200, processor 2200 may also  
25 include logic to handle memory misses. In at least one embodiment, if a data load misses in a data cache, there may be dependent operations in flight in pipeline that have left a scheduler with temporarily incorrect data. In at least one embodiment, a replay mechanism tracks and re-executes instructions that use incorrect data. In at least one embodiment, dependent operations might need to be replayed and independent ones may be allowed to complete. In at  
30 least one embodiment, schedulers and replay mechanisms of at least one embodiment of a processor may also be designed to catch instruction sequences for text string comparison operations.

[0326] In at least one embodiment, the 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  
35 embodiment, registers may be those that may be usable from outside of a processor (from a programmer's perspective). In at least one embodiment, registers might not be limited to a particular type of circuit. Rather, in at least one embodiment, a register may store data,

5 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.  
10 A register file of at least one embodiment also contains eight multimedia SIMD registers for packed data.

[0327] FIG. 23 illustrates a processor 2300, in accordance with at least one embodiment. In at least one embodiment, processor 2300 includes, without limitation, one or more processor cores (“cores”) 2302A-2302N, an integrated memory controller 2314, and an integrated  
15 graphics processor 2308. In at least one embodiment, processor 2300 can include additional cores up to and including additional processor core 2302N represented by dashed lined boxes. In at least one embodiment, each of processor cores 2302A-2302N includes one or more internal cache units 2304A-2304N. In at least one embodiment, each processor core also has access to one or more shared cached units 2306.

20 [0328] In at least one embodiment, internal cache units 2304A-2304N and shared cache units 2306 represent a cache memory hierarchy within processor 2300. In at least one embodiment, cache memory units 2304A-2304N may include at least one level of instruction and data cache within each processor core and one or more levels of shared mid-level cache, such as an L2, L3, Level 4 (“L4”), or other levels of cache, where a highest level of cache before  
25 external memory is classified as an LLC. In at least one embodiment, cache coherency logic maintains coherency between various cache units 2306 and 2304A-2304N.

[0329] In at least one embodiment, processor 2300 may also include a set of one or more bus controller units 2316 and a system agent core 2310. In at least one embodiment, one or more bus controller units 2316 manage a set of peripheral buses, such as one or more PCI or PCI  
30 express buses. In at least one embodiment, system agent core 2310 provides management functionality for various processor components. In at least one embodiment, system agent core 2310 includes one or more integrated memory controllers 2314 to manage access to various external memory devices (not shown).

[0330] In at least one embodiment, one or more of processor cores 2302A-2302N include  
35 support for simultaneous multi-threading. In at least one embodiment, system agent core 2310 includes components for coordinating and operating processor cores 2302A-2302N

5 during multi-threaded processing. In at least one embodiment, system agent core 2310 may additionally include a power control unit (“PCU”), which includes logic and components to regulate one or more power states of processor cores 2302A-2302N and graphics processor 2308.

[0331] In at least one embodiment, processor 2300 additionally includes graphics processor  
10 2308 to execute graphics processing operations. In at least one embodiment, graphics processor 2308 couples with shared cache units 2306, and system agent core 2310, including one or more integrated memory controllers 2314. In at least one embodiment, system agent core 2310 also includes a display controller 2311 to drive graphics processor output to one or more coupled displays. In at least one embodiment, display controller 2311 may also be a  
15 separate module coupled with graphics processor 2308 via at least one interconnect, or may be integrated within graphics processor 2308.

[0332] In at least one embodiment, a ring based interconnect unit 2312 is used to couple internal components of processor 2300. In at least one embodiment, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect,  
20 or other techniques. In at least one embodiment, graphics processor 2308 couples with ring interconnect 2312 via an I/O link 2313.

[0333] In at least one embodiment, I/O link 2313 represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded  
25 memory module 2318, such as an eDRAM module. In at least one embodiment, each of processor cores 2302A-2302N and graphics processor 2308 use embedded memory modules 2318 as a shared LLC.

[0334] In at least one embodiment, processor cores 2302A-2302N are homogeneous cores executing a common instruction set architecture. In at least one embodiment, processor cores  
30 2302A-2302N are heterogeneous in terms of ISA, where one or more of processor cores 2302A-2302N execute a common instruction set, while one or more other cores of processor cores 2302A-23-02N executes a subset of a common instruction set or a different instruction set. In at least one embodiment, processor cores 2302A-2302N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption  
35 couple with one or more cores having a lower power consumption. In at least one

5 embodiment, processor 2300 can be implemented on one or more chips or as an SoC integrated circuit.

[0335] FIG. 24 illustrates a graphics processor core 2400, in accordance with at least one embodiment described. In at least one embodiment, graphics processor core 2400 is included within a graphics core array. In at least one embodiment, graphics processor core 2400,  
10 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 2400 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 2400 can include a fixed function block 2430 coupled with  
15 multiple sub-cores 2401A-2401F, also referred to as sub-slices, that include modular blocks of general-purpose and fixed function logic.

[0336] In at least one embodiment, fixed function block 2430 includes a geometry/fixed function pipeline 2436 that can be shared by all sub-cores in graphics processor 2400, for example, in lower performance and/or lower power graphics processor implementations. In at  
20 least one embodiment, geometry/fixed function pipeline 2436 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.

[0337] In at least one embodiment, fixed function block 2430 also includes a graphics SoC interface 2437, a graphics microcontroller 2438, and a media pipeline 2439. Graphics SoC  
25 interface 2437 provides an interface between graphics core 2400 and other processor cores within an SoC integrated circuit. In at least one embodiment, graphics microcontroller 2438 is a programmable sub-processor that is configurable to manage various functions of graphics processor 2400, including thread dispatch, scheduling, and pre-emption. In at least one embodiment, media pipeline 2439 includes logic to facilitate decoding, encoding, pre-  
30 processing, and/or post-processing of multimedia data, including image and video data. In at least one embodiment, media pipeline 2439 implements media operations via requests to compute or sampling logic within sub-cores 2401-2401F.

[0338] In at least one embodiment, SoC interface 2437 enables graphics core 2400 to communicate with general-purpose application processor cores (e.g., CPUs) and/or other  
35 components within an SoC, including memory hierarchy elements such as a shared LLC memory, system RAM, and/or embedded on-chip or on-package DRAM. In at least one

5 embodiment, SoC interface 2437 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 2400 and CPUs within an SoC. In at least one embodiment, SoC interface 2437 can also implement power management controls for graphics core 2400 and enable an interface between a clock domain of graphic  
10 core 2400 and other clock domains within an SoC. In at least one embodiment, SoC interface 2437 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 2439, when media operations are to be  
15 performed, or a geometry and fixed function pipeline (e.g., geometry and fixed function pipeline 2436, geometry and fixed function pipeline 2414) when graphics processing operations are to be performed.

[0339] In at least one embodiment, graphics microcontroller 2438 can be configured to perform various scheduling and management tasks for graphics core 2400. In at least one  
20 embodiment, graphics microcontroller 2438 can perform graphics and/or compute workload scheduling on various graphics parallel engines within execution unit (EU) arrays 2402A-2402F, 2404A-2404F within sub-cores 2401A-2401F. In at least one embodiment, host software executing on a CPU core of an SoC including graphics core 2400 can submit workloads one of multiple graphic processor doorbells, which invokes a scheduling operation  
25 on an appropriate graphics engine. In at least one embodiment, scheduling operations include determining which workload to run next, submitting a workload to a command streamer, pre-empting existing workloads running on an engine, monitoring progress of a workload, and notifying host software when a workload is complete. In at least one embodiment, graphics microcontroller 2438 can also facilitate low-power or idle states for graphics core 2400,  
30 providing graphics core 2400 with an ability to save and restore registers within graphics core 2400 across low-power state transitions independently from an operating system and/or graphics driver software on a system.

[0340] In at least one embodiment, graphics core 2400 may have greater than or fewer than illustrated sub-cores 2401A-2401F, up to N modular sub-cores. For each set of N sub-cores,  
35 in at least one embodiment, graphics core 2400 can also include shared function logic 2410, shared and/or cache memory 2412, a geometry/fixed function pipeline 2414, as well as additional fixed function logic 2416 to accelerate various graphics and compute processing

5 operations. In at least one embodiment, shared function logic 2410 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 2400. Shared and/or cache memory 2412 can be an LLC for N sub-cores 2401A-2401F within graphics core 2400 and can also serve as shared memory that is accessible by multiple sub-cores. In at least one embodiment, geometry/fixed function  
10 pipeline 2414 can be included instead of geometry/fixed function pipeline 2436 within fixed function block 2430 and can include same or similar logic units.

[0341] In at least one embodiment, graphics core 2400 includes additional fixed function logic 2416 that can include various fixed function acceleration logic for use by graphics core 2400. In at least one embodiment, additional fixed function logic 2416 includes an additional  
15 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 2416, 2436, and a cull pipeline, which is an additional geometry pipeline which may be included within additional fixed function logic 2416. In at least one embodiment, cull pipeline is a trimmed down version of a full geometry pipeline. In at least one embodiment, a  
20 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 2416 can execute position shaders in parallel with a main application and  
25 generally generates critical results faster than a full pipeline, as a 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, a cull pipeline can use generated critical results to compute visibility information for all triangles without regard to whether those triangles are culled. In at least one embodiment, a full pipeline (which in this instance may be referred  
30 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.

[0342] In at least one embodiment, additional fixed function logic 2416 can also include general purpose processing acceleration logic, such as fixed function matrix multiplication logic, for accelerating CUDA programs.

35 [0343] In at least one embodiment, each graphics sub-core 2401A-2401F 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

5 embodiment, graphics sub-cores 2401A-2401F include multiple EU arrays 2402A-2402F, 2404A-2404F, thread dispatch and inter-thread communication (“TD/IC”) logic 2403A-2403F, a 3D (e.g., texture) sampler 2405A-2405F, a media sampler 2406A-2406F, a shader processor 2407A-2407F, and shared local memory (“SLM”) 2408A-2408F. EU arrays 2402A-2402F, 2404A-2404F each include multiple execution units, which are GPGPUs  
 10 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 2403A-2403F 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  
 15 embodiment, 3D sampler 2405A-2405F 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 2406A-2406F can perform similar read operations based on a type and format associated with media data. In at least one embodiment, each graphics sub-  
 20 core 2401A-2401F can alternately include a unified 3D and media sampler. In at least one embodiment, threads executing on execution units within each of sub-cores 2401A-2401F can make use of shared local memory 2408A-2408F within each sub-core, to enable threads executing within a thread group to execute using a common pool of on-chip memory.

[0344] FIG. 25 illustrates a parallel processing unit (“PPU”) 2500, in accordance with at least  
 25 one embodiment. In at least one embodiment, PPU 2500 is configured with machine-readable code that, if executed by PPU 2500, causes PPU 2500 to perform some or all of processes and techniques described herein. In at least one embodiment, PPU 2500 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  
 30 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 2500. In at least one embodiment, PPU 2500 is a GPU configured to implement a graphics rendering pipeline for processing three-dimensional (“3D”) graphics data in order to generate  
 35 two-dimensional (“2D”) image data for display on a display device such as an LCD device. In at least one embodiment, PPU 2500 is utilized to perform computations such as linear algebra operations and machine-learning operations. FIG. 25 illustrates an example parallel

5 processor for illustrative purposes only and should be construed as a non-limiting example of a processor architecture that may be implemented in at least one embodiment.

[0345] In at least one embodiment, one or more PPUs 2500 are configured to accelerate High Performance Computing (“HPC”), data center, and machine learning applications. In at least one embodiment, one or more PPUs 2500 are configured to accelerate CUDA programs. In at least one embodiment, PPU 2500 includes, without limitation, an I/O unit 2506, a front-end unit 2510, a scheduler unit 2512, a work distribution unit 2514, a hub 2516, a crossbar (“Xbar”) 2520, one or more general processing clusters (“GPCs”) 2518, and one or more partition units (“memory partition units”) 2522. In at least one embodiment, PPU 2500 is connected to a host processor or other PPUs 2500 via one or more high-speed GPU interconnects (“GPU interconnects”) 2508. In at least one embodiment, PPU 2500 is connected to a host processor or other peripheral devices via a system bus or interconnect 2502. In at least one embodiment, PPU 2500 is connected to a local memory comprising one or more memory devices (“memory”) 2504. In at least one embodiment, memory devices 2504 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.

[0346] In at least one embodiment, high-speed GPU interconnect 2508 may refer to a wire-based multi-lane communications link that is used by systems to scale and include one or more PPUs 2500 combined with one or more CPUs, supports cache coherence between PPUs 2500 and CPUs, and CPU mastering. In at least one embodiment, data and/or commands are transmitted by high-speed GPU interconnect 2508 through hub 2516 to/from other units of PPU 2500 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. 25.

[0347] In at least one embodiment, I/O unit 2506 is configured to transmit and receive communications (e.g., commands, data) from a host processor (not illustrated in FIG. 25) over system bus 2502. In at least one embodiment, I/O unit 2506 communicates with host processor directly via system bus 2502 or through one or more intermediate devices such as a memory bridge. In at least one embodiment, I/O unit 2506 may communicate with one or more other processors, such as one or more of PPUs 2500 via system bus 2502. In at least one embodiment, I/O unit 2506 implements a PCIe interface for communications over a PCIe

5 bus. In at least one embodiment, I/O unit 2506 implements interfaces for communicating with external devices.

[0348] In at least one embodiment, I/O unit 2506 decodes packets received via system bus 2502. In at least one embodiment, at least some packets represent commands configured to cause PPU 2500 to perform various operations. In at least one embodiment, I/O unit 2506  
10 transmits decoded commands to various other units of PPU 2500 as specified by commands. In at least one embodiment, commands are transmitted to front-end unit 2510 and/or transmitted to hub 2516 or other units of PPU 2500 such as one or more copy engines, a video encoder, a video decoder, a power management unit, etc. (not explicitly illustrated in FIG. 25). In at least one embodiment, I/O unit 2506 is configured to route communications  
15 between and among various logical units of PPU 2500.

[0349] In at least one embodiment, a program executed by host processor encodes a command stream in a buffer that provides workloads to PPU 2500 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  
20 (e.g., read/write) by both a host processor and PPU 2500 — a host interface unit may be configured to access buffer in a system memory connected to system bus 2502 via memory requests transmitted over system bus 2502 by I/O unit 2506. In at least one embodiment, a host processor writes a command stream to a buffer and then transmits a pointer to the start of the command stream to PPU 2500 such that front-end unit 2510 receives pointers to one or  
25 more command streams and manages one or more command streams, reading commands from command streams and forwarding commands to various units of PPU 2500.

[0350] In at least one embodiment, front-end unit 2510 is coupled to scheduler unit 2512 that configures various GPCs 2518 to process tasks defined by one or more command streams. In at least one embodiment, scheduler unit 2512 is configured to track state information related  
30 to various tasks managed by scheduler unit 2512 where state information may indicate which of GPCs 2518 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 2512 manages execution of a plurality of tasks on one or more of GPCs 2518.

[0351] In at least one embodiment, scheduler unit 2512 is coupled to work distribution unit  
35 2514 that is configured to dispatch tasks for execution on GPCs 2518. In at least one embodiment, work distribution unit 2514 tracks a number of scheduled tasks received from

5 scheduler unit 2512 and work distribution unit 2514 manages a pending task pool and an active task pool for each of GPCs 2518. 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 2518; active task pool may comprise a number of slots (e.g., 4 slots) for tasks that are actively being processed by GPCs 2518 such that as one of GPCs 2518 completes  
10 execution of a task, that task is evicted from active task pool for GPC 2518 and one of other tasks from pending task pool is selected and scheduled for execution on GPC 2518. In at least one embodiment, if an active task is idle on GPC 2518, such as while waiting for a data dependency to be resolved, then the active task is evicted from GPC 2518 and returned to a pending task pool while another task in the pending task pool is selected and scheduled for  
15 execution on GPC 2518.

[0352] In at least one embodiment, work distribution unit 2514 communicates with one or more GPCs 2518 via XBar 2520. In at least one embodiment, XBar 2520 is an interconnect network that couples many units of PPU 2500 to other units of PPU 2500 and can be configured to couple work distribution unit 2514 to a particular GPC 2518. In at least one  
20 embodiment, one or more other units of PPU 2500 may also be connected to XBar 2520 via hub 2516.

[0353] In at least one embodiment, tasks are managed by scheduler unit 2512 and dispatched to one of GPCs 2518 by work distribution unit 2514. GPC 2518 is configured to process task and generate results. In at least one embodiment, results may be consumed by other tasks  
25 within GPC 2518, routed to a different GPC 2518 via XBar 2520, or stored in memory 2504. In at least one embodiment, results can be written to memory 2504 via partition units 2522, which implement a memory interface for reading and writing data to/from memory 2504. In at least one embodiment, results can be transmitted to another PPU 2504 or CPU via high-speed GPU interconnect 2508. In at least one embodiment, PPU 2500 includes, without  
30 limitation, a number U of partition units 2522 that is equal to number of separate and distinct memory devices 2504 coupled to PPU 2500.

[0354] 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  
35 executing on host processor to schedule operations for execution on PPU 2500. In at least one embodiment, multiple compute applications are simultaneously executed by PPU 2500 and PPU 2500 provides isolation, quality of service (“QoS”), and independent address spaces for multiple compute applications. In at least one embodiment, an application generates

5 instructions (e.g., in the form of API calls) that cause a driver kernel to generate one or more tasks for execution by PPU 2500 and the driver kernel outputs tasks to one or more streams being processed by PPU 2500. 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  
10 parallel. In at least one embodiment, cooperating threads can refer to a plurality of threads including instructions to perform a task and that exchange data through shared memory.

[0355] FIG. 26 illustrates a GPC 2600, in accordance with at least one embodiment. In at least one embodiment, GPC 2600 is GPC 2518 of FIG. 25. In at least one embodiment, each GPC 2600 includes, without limitation, a number of hardware units for processing tasks and  
15 each GPC 2600 includes, without limitation, a pipeline manager 2602, a pre-raster operations unit (“PROP”) 2604, a raster engine 2608, a work distribution crossbar (“WDX”) 2616, an MMU 2618, one or more Data Processing Clusters (“DPCs”) 2606, and any suitable combination of parts.

[0356] In at least one embodiment, operation of GPC 2600 is controlled by pipeline  
20 manager 2602. In at least one embodiment, pipeline manager 2602 manages configuration of one or more DPCs 2606 for processing tasks allocated to GPC 2600. In at least one embodiment, pipeline manager 2602 configures at least one of one or more DPCs 2606 to implement at least a portion of a graphics rendering pipeline. In at least one embodiment, DPC 2606 is configured to execute a vertex shader program on a programmable streaming  
25 multiprocessor (“SM”) 2614. In at least one embodiment, pipeline manager 2602 is configured to route packets received from a work distribution unit to appropriate logical units within GPC 2600 and, in at least one embodiment, some packets may be routed to fixed function hardware units in PROP 2604 and/or raster engine 2608 while other packets may be routed to DPCs 2606 for processing by a primitive engine 2612 or SM 2614. In at least one  
30 embodiment, pipeline manager 2602 configures at least one of DPCs 2606 to implement a computing pipeline. In at least one embodiment, pipeline manager 2602 configures at least one of DPCs 2606 to execute at least a portion of a CUDA program.

[0357] In at least one embodiment, PROP unit 2604 is configured to route data generated by raster engine 2608 and DPCs 2606 to a Raster Operations (“ROP”) unit in a partition unit,  
35 such as memory partition unit 2522 described in more detail above in conjunction with FIG. 25. In at least one embodiment, PROP unit 2604 is configured to perform optimizations for color blending, organize pixel data, perform address translations, and more. In at least one

5 embodiment, raster engine 2608 includes, without limitation, a number of fixed function hardware units configured to perform various raster operations and, in at least one embodiment, raster engine 2608 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, a setup engine receives  
10 transformed vertices and generates plane equations associated with geometric primitive defined by vertices; plane equations are transmitted to a coarse raster engine to generate coverage information (e.g., an x, y coverage mask for a tile) for a primitive; the output of the coarse raster engine is transmitted to a culling engine where fragments associated with a primitive that fail a z-test are culled, and transmitted to a clipping engine where fragments  
15 lying outside a viewing frustum are clipped. In at least one embodiment, fragments that survive clipping and culling are passed to a fine raster engine to generate attributes for pixel fragments based on plane equations generated by a setup engine. In at least one embodiment, the output of raster engine 2608 comprises fragments to be processed by any suitable entity such as by a fragment shader implemented within DPC 2606.

20 **[0358]** In at least one embodiment, each DPC 2606 included in GPC 2600 comprise, without limitation, an M-Pipe Controller (“MPC”) 2610; primitive engine 2612; one or more SMs 2614; and any suitable combination thereof. In at least one embodiment, MPC 2610 controls operation of DPC 2606, routing packets received from pipeline manager 2602 to appropriate units in DPC 2606. In at least one embodiment, packets associated with a vertex are routed to  
25 primitive engine 2612, 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 2614.

**[0359]** In at least one embodiment, SM 2614 comprises, without limitation, a programmable streaming processor that is configured to process tasks represented by a number of threads. In  
30 at least one embodiment, SM 2614 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 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  
35 2614 implements a 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

5 embodiment, a program counter, a call stack, and an execution state is maintained for each warp, enabling concurrency between warps and serial execution within warps when threads within a warp diverge. In another embodiment, a program counter, a call stack, and an execution state is maintained for each individual thread, enabling equal concurrency between all threads, within and between warps. In at least one embodiment, an execution state is  
10 maintained for each individual thread and threads executing the same instructions may be converged and executed in parallel for better efficiency. At least one embodiment of SM 2614 is described in more detail in conjunction with FIG. 27.

[0360] In at least one embodiment, MMU 2618 provides an interface between GPC 2600 and a memory partition unit (e.g., partition unit 2522 of FIG. 25) and MMU 2618 provides  
15 translation of virtual addresses into physical addresses, memory protection, and arbitration of memory requests. In at least one embodiment, MMU 2618 provides one or more translation lookaside buffers (TLBs) for performing translation of virtual addresses into physical addresses in memory.

[0361] FIG. 27 illustrates a streaming multiprocessor ("SM") 2700, in accordance with at  
20 least one embodiment. In at least one embodiment, SM 2700 is SM 2614 of FIG. 26. In at least one embodiment, SM 2700 includes, without limitation, an instruction cache 2702; one or more scheduler units 2704; a register file 2708; one or more processing cores ("cores") 2710; one or more special function units ("SFUs") 2712; one or more LSUs 2714; an interconnect network 2716; a shared memory/L1 cache 2718; and any suitable combination  
25 thereof. In at least one embodiment, a work distribution unit dispatches tasks for execution on GPCs of parallel processing units (PPUs) and each task is allocated to a particular Data Processing Cluster (DPC) within a GPC and, if a task is associated with a shader program, then the task is allocated to one of SMs 2700. In at least one embodiment, scheduler unit 2704 receives tasks from a work distribution unit and manages instruction scheduling for one  
30 or more thread blocks assigned to SM 2700. In at least one embodiment, scheduler unit 2704 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 2704 manages a plurality of different thread blocks, allocating warps to different thread blocks and then dispatching instructions from a plurality  
35 of different cooperative groups to various functional units (e.g., processing cores 2710, SFUs 2712, and LSUs 2714) during each clock cycle.

- 5     **[0362]** 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
- 10    embodiment, APIs of conventional programming models provide a single, simple construct for synchronizing cooperating threads: a barrier across all threads of a thread block (e.g., `syncthreads()` function). However, in at least one embodiment, programmers may define groups of threads at smaller than thread block granularities and synchronize within defined groups to enable greater performance, design flexibility, and software reuse in the form of
- 15    collective group-wide function interfaces. In at least one embodiment, cooperative groups enable 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. In at least one embodiment, a sub-block granularity is as small as a single thread. In at least one embodiment, a programming model supports clean composition across
- 20    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 group 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.
- 25    **[0363]** In at least one embodiment, a dispatch unit 2706 is configured to transmit instructions to one or more of functional units and scheduler unit 2704 includes, without limitation, two dispatch units 2706 that enable two different instructions from same warp to be dispatched during each clock cycle. In at least one embodiment, each scheduler unit 2704 includes a single dispatch unit 2706 or additional dispatch units 2706.
- 30    **[0364]** In at least one embodiment, each SM 2700, in at least one embodiment, includes, without limitation, register file 2708 that provides a set of registers for functional units of SM 2700. In at least one embodiment, register file 2708 is divided between each of the functional units such that each functional unit is allocated a dedicated portion of register file 2708. In at least one embodiment, register file 2708 is divided between different warps being executed
- 35    by SM 2700 and register file 2708 provides temporary storage for operands connected to data paths of functional units. In at least one embodiment, each SM 2700 comprises, without limitation, a plurality of L processing cores 2710. In at least one embodiment, SM 2700

5 includes, without limitation, a large number (e.g., 128 or more) of distinct processing cores 2710. In at least one embodiment, each processing core 2710 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  
 10 implement IEEE 754-2008 standard for floating point arithmetic. In at least one embodiment, processing cores 2710 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.

[0365] In at least one embodiment, tensor cores are configured to perform matrix operations. In at least one embodiment, one or more tensor cores are included in processing cores 2710.

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

[0366] In at least one embodiment, matrix multiply inputs A and B are 16-bit floating point  
 20 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  
 25 multiply. Tensor cores are used to perform much larger two-dimensional or higher dimensional matrix operations, built up from these smaller elements, in at least one embodiment. In at least one embodiment, an API, such as a CUDA-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 the  
 30 CUDA level, a warp-level interface assumes 16x16 size matrices spanning all 32 threads of a warp.

[0367] In at least one embodiment, each SM 2700 comprises, without limitation, M SFUs 2712 that perform special functions (e.g., attribute evaluation, reciprocal square root, and like). In at least one embodiment, SFUs 2712 include, without limitation, a tree traversal unit  
 35 configured to traverse a hierarchical tree data structure. In at least one embodiment, SFUs 2712 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

5 (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 2700. In at least one embodiment, texture maps are stored in shared memory/L1 cache 2718. 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 at least one embodiment, each SM 2700 includes, without  
10 limitation, two texture units.

[0368] In at least one embodiment, each SM 2700 comprises, without limitation, N LSUs 2714 that implement load and store operations between shared memory/L1 cache 2718 and register file 2708. In at least one embodiment, each SM 2700 includes, without limitation, interconnect network 2716 that connects each of the functional units to register file 2708 and  
15 LSU 2714 to register file 2708 and shared memory/ L1 cache 2718. In at least one embodiment, interconnect network 2716 is a crossbar that can be configured to connect any of the functional units to any of the registers in register file 2708 and connect LSUs 2714 to register file 2708 and memory locations in shared memory/L1 cache 2718.

[0369] In at least one embodiment, shared memory/L1 cache 2718 is an array of on-chip  
20 memory that allows for data storage and communication between SM 2700 and a primitive engine and between threads in SM 2700. In at least one embodiment, shared memory/L1 cache 2718 comprises, without limitation, 128KB of storage capacity and is in a path from SM 2700 to a partition unit. In at least one embodiment, shared memory/L1 cache 2718 is used to cache reads and writes. In at least one embodiment, one or more of shared  
25 memory/L1 cache 2718, L2 cache, and memory are backing stores.

[0370] In at least one embodiment, 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, 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  
30 capacity, texture and load/store operations can use remaining capacity. In at least one embodiment, integration within shared memory/L1 cache 2718 enables shared memory/L1 cache 2718 to function as a high-throughput conduit for streaming data while simultaneously providing high-bandwidth and low-latency access to frequently reused data. In at least one embodiment, when configured for general purpose parallel computation, a simpler  
35 configuration can be used compared with graphics processing. In at least one embodiment, fixed function GPUs are bypassed, creating a much simpler programming model. In at least one embodiment and in a general purpose parallel computation configuration, a work

5 distribution unit assigns and distributes blocks of threads directly to DPCs. In at least one embodiment, threads in a block execute the same program, using a unique thread ID in a calculation to ensure each thread generates unique results, using SM 2700 to execute a program and perform calculations, shared memory/L1 cache 2718 to communicate between threads, and LSU 2714 to read and write global memory through shared memory/L1 cache  
 10 2718 and a memory partition unit. In at least one embodiment, when configured for general purpose parallel computation, SM 2700 writes commands that scheduler unit 2704 can use to launch new work on DPCs.

[0371] 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,  
 15 hand-held device), a 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 an SoC along with one or more other devices such as additional PPUs, memory, a RISC CPU, an MMU, a digital-to-analog converter (“DAC”), and like.

20 [0372] In at least one embodiment, PPU may be included on a graphics card that includes one or more memory devices. In at least one embodiment, 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 GPU (“iGPU”) included in chipset of motherboard.

#### **Software Constructions for General-Purpose Computing**

25 [0373] The following FIGS. set forth, without limitation, exemplary software constructs for implementing at least one embodiment.

[0374] FIG. 28 illustrates a software stack of a programming platform, in accordance with at least one embodiment. In at least one embodiment, a programming platform is a platform for leveraging hardware on a computing system to accelerate computational tasks. A  
 30 programming platform may be accessible to software developers through libraries, compiler directives, and/or extensions to programming languages, in at least one embodiment. In at least one embodiment, a programming platform may be, but is not limited to, CUDA, Radeon Open Compute Platform (“ROCm”), OpenCL (OpenCL™ is developed by Khronos group), SYCL, or Intel One API.

35 [0375] In at least one embodiment, a software stack 2800 of a programming platform provides an execution environment for an application 2801. In at least one embodiment,

- 5 application 2801 may include any computer software capable of being launched on software stack 2800. In at least one embodiment, application 2801 may include, but is not limited to, an artificial intelligence (“AI”)/machine learning (“ML”) application, a high performance computing (“HPC”) application, a virtual desktop infrastructure (“VDI”), or a data center workload.
- 10 **[0376]** In at least one embodiment, application 2801 and software stack 2800 run on hardware 2807. Hardware 2807 may include one or more GPUs, CPUs, FPGAs, AI engines, and/or other types of compute devices that support a programming platform, in at least one embodiment. In at least one embodiment, such as with CUDA, software stack 2800 may be vendor specific and compatible with only devices from particular vendor(s). In at least one
- 15 embodiment, such as in with OpenCL, software stack 2800 may be used with devices from different vendors. In at least one embodiment, hardware 2807 includes a host connected to one more devices that can be accessed to perform computational tasks via application programming interface (“API”) calls. A device within hardware 2807 may include, but is not limited to, a GPU, FPGA, AI engine, or other compute device (but may also include a CPU)
- 20 and its memory, as opposed to a host within hardware 2807 that may include, but is not limited to, a CPU (but may also include a compute device) and its memory, in at least one embodiment.
- [0377]** In at least one embodiment, software stack 2800 of a programming platform includes, without limitation, a number of libraries 2803, a runtime 2805, and a device kernel driver
- 25 2806. Each of libraries 2803 may include data and programming code that can be used by computer programs and leveraged during software development, in at least one embodiment. In at least one embodiment, libraries 2803 may include, but are not limited to, pre-written code and subroutines, classes, values, type specifications, configuration data, documentation, help data, and/or message templates. In at least one embodiment, libraries 2803 include
- 30 functions that are optimized for execution on one or more types of devices. In at least one embodiment, libraries 2803 may include, but are not limited to, functions for performing mathematical, deep learning, and/or other types of operations on devices. In at least one embodiment, libraries 2803 are associated with corresponding APIs 2802, which may include one or more APIs, that expose functions implemented in libraries 2803.
- 35 **[0378]** In at least one embodiment, application 2801 is written as source code that is compiled into executable code, as discussed in greater detail below in conjunction with FIGS. 33 – 35. Executable code of application 2801 may run, at least in part, on an execution

5 environment provided by software stack 2800, in at least one embodiment. In at least one embodiment, during execution of application 2801, code may be reached that needs to run on a device, as opposed to a host. In such a case, runtime 2805 may be called to load and launch requisite code on the device, in at least one embodiment. In at least one embodiment, runtime 2805 may include any technically feasible runtime system that is able to support execution of  
10 application S01.

[0379] In at least one embodiment, runtime 2805 is implemented as one or more runtime libraries associated with corresponding APIs, which are shown as API(s) 2804. One or more of such runtime libraries may include, without limitation, functions for memory management, execution control, device management, error handling, and/or synchronization, among other  
15 things, in at least one embodiment. In at least one embodiment, memory management functions may include, but are not limited to, functions to allocate, deallocate, and copy device memory, as well as transfer data between host memory and device memory. In at least one embodiment, execution control functions may include, but are not limited to, functions to launch a function (sometimes referred to as a “kernel” when a function is a global function  
20 callable from a host) on a device and set attribute values in a buffer maintained by a runtime library for a given function to be executed on a device.

[0380] Runtime libraries and corresponding API(s) 2804 may be implemented in any technically feasible manner, in at least one embodiment. In at least one embodiment, one (or any number of) API may expose a low-level set of functions for fine-grained control of a  
25 device, while another (or any number of) API may expose a higher-level set of such functions. In at least one embodiment, a high-level runtime API may be built on top of a low-level API. In at least one embodiment, one or more of runtime APIs may be language-specific APIs that are layered on top of a language-independent runtime API.

[0381] In at least one embodiment, device kernel driver 2806 is configured to facilitate  
30 communication with an underlying device. In at least one embodiment, device kernel driver 2806 may provide low-level functionalities upon which APIs, such as API(s) 2804, and/or other software relies. In at least one embodiment, device kernel driver 2806 may be configured to compile intermediate representation (“IR”) code into binary code at runtime. For CUDA, device kernel driver 2806 may compile Parallel Thread Execution (“PTX”) IR  
35 code that is not hardware specific into binary code for a specific target device at runtime (with caching of compiled binary code), which is also sometimes referred to as “finalizing” code, in at least one embodiment. Doing so may permit finalized code to run on a target

5 device, which may not have existed when source code was originally compiled into PTX code, in at least one embodiment. Alternatively, in at least one embodiment, device source code may be compiled into binary code offline, without requiring device kernel driver 2806 to compile IR code at runtime.

[0382] FIG. 29 illustrates a CUDA implementation of software stack 2800 of FIG. 28, in  
10 accordance with at least one embodiment. In at least one embodiment, a CUDA software stack 2900, on which an application 2901 may be launched, includes CUDA libraries 2903, a CUDA runtime 2905, a CUDA driver 2907, and a device kernel driver 2908. In at least one embodiment, CUDA software stack 2900 executes on hardware 2909, which may include a GPU that supports CUDA and is developed by NVIDIA Corporation of Santa Clara, CA.

15 [0383] In at least one embodiment, application 2901, CUDA runtime 2905, and device kernel driver 2908 may perform similar functionalities as application 2801, runtime 2805, and device kernel driver 2806, respectively, which are described above in conjunction with FIG. 28. In at least one embodiment, CUDA driver 2907 includes a library (libcuda.so) that implements a CUDA driver API 2906. Similar to a CUDA runtime API 2904 implemented  
20 by a CUDA runtime library (cudart), CUDA driver API 2906 may, without limitation, expose functions for memory management, execution control, device management, error handling, synchronization, and/or graphics interoperability, among other things, in at least one embodiment. In at least one embodiment, CUDA driver API 2906 differs from CUDA runtime API 2904 in that CUDA runtime API 2904 simplifies device code management by  
25 providing implicit initialization, context (analogous to a process) management, and module (analogous to dynamically loaded libraries) management. In contrast to high-level CUDA runtime API 2904, CUDA driver API 2906 is a low-level API providing more fine-grained control of the device, particularly with respect to contexts and module loading, in at least one embodiment. In at least one embodiment, CUDA driver API 2906 may expose functions for  
30 context management that are not exposed by CUDA runtime API 2904. In at least one embodiment, CUDA driver API 2906 is also language-independent and supports, e.g., OpenCL in addition to CUDA runtime API 2904. Further, in at least one embodiment, development libraries, including CUDA runtime 2905, may be considered as separate from driver components, including user-mode CUDA driver 2907 and kernel-mode device driver  
35 2908 (also sometimes referred to as a “display” driver).

[0384] In at least one embodiment, CUDA libraries 2903 may include, but are not limited to, mathematical libraries, deep learning libraries, parallel algorithm libraries, and/or

5 signal/image/video processing libraries, which parallel computing applications such as application 2901 may utilize. In at least one embodiment, CUDA libraries 2903 may include mathematical libraries such as a cuBLAS library that is an implementation of Basic Linear Algebra Subprograms (“BLAS”) for performing linear algebra operations, a cuFFT library for computing fast Fourier transforms (“FFTs”), and a cuRAND library for generating  
 10 random numbers, among others. In at least one embodiment, CUDA libraries 2903 may include deep learning libraries such as a cuDNN library of primitives for deep neural networks and a TensorRT platform for high-performance deep learning inference, among others.

[0385] FIG. 30 illustrates a ROCm implementation of software stack 2800 of FIG. 28, in  
 15 accordance with at least one embodiment. In at least one embodiment, a ROCm software stack 3000, on which an application 3001 may be launched, includes a language runtime 3003, a system runtime 3005, a thunk 3007, and a ROCm kernel driver 3008. In at least one embodiment, ROCm software stack 3000 executes on hardware 3009, which may include a GPU that supports ROCm and is developed by AMD Corporation of Santa Clara, CA.

20 [0386] In at least one embodiment, application 3001 may perform similar functionalities as application 2801 discussed above in conjunction with FIG. 28. In addition, language runtime 3003 and system runtime 3005 may perform similar functionalities as runtime 2805 discussed above in conjunction with FIG. 28, in at least one embodiment. In at least one embodiment, language runtime 3003 and system runtime 3005 differ in that system runtime 3005 is a  
 25 language-independent runtime that implements a ROCr system runtime API 3004 and makes use of a Heterogeneous System Architecture (“HSA”) Runtime API. HSA runtime API is a thin, user-mode API that exposes interfaces to access and interact with an AMD GPU, including functions for memory management, execution control via architected dispatch of kernels, error handling, system and agent information, and runtime initialization and  
 30 shutdown, among other things, in at least one embodiment. In contrast to system runtime 3005, language runtime 3003 is an implementation of a language-specific runtime API 3002 layered on top of ROCr system runtime API 3004, in at least one embodiment. In at least one embodiment, language runtime API may include, but is not limited to, a Heterogeneous compute Interface for Portability (“HIP”) language runtime API, a Heterogeneous Compute  
 35 Compiler (“HCC”) language runtime API, or an OpenCL API, among others. HIP language in particular is an extension of C++ programming language with functionally similar versions of CUDA mechanisms, and, in at least one embodiment, a HIP language runtime API

5 includes functions that are similar to those of CUDA runtime API 2904 discussed above in conjunction with FIG. 29, such as functions for memory management, execution control, device management, error handling, and synchronization, among other things.

[0387] In at least one embodiment, thunk (ROCt) 3007 is an interface 3006 that can be used to interact with underlying ROCm driver 3008. In at least one embodiment, ROCm driver  
10 3008 is a ROCk driver, which is a combination of an AMDGPU driver and a HSA kernel driver (amdkfd). In at least one embodiment, AMDGPU driver is a device kernel driver for GPUs developed by AMD that performs similar functionalities as device kernel driver 2806 discussed above in conjunction with FIG. 28. In at least one embodiment, HSA kernel driver is a driver permitting different types of processors to share system resources more effectively  
15 via hardware features.

[0388] In at least one embodiment, various libraries (not shown) may be included in ROCm software stack 3000 above language runtime 3003 and provide functionality similarity to CUDA libraries 2903, discussed above in conjunction with FIG. 29. In at least one embodiment, various libraries may include, but are not limited to, mathematical, deep  
20 learning, and/or other libraries such as a hipBLAS library that implements functions similar to those of CUDA cuBLAS, a rocFFT library for computing FFTs that is similar to CUDA cuFFT, among others.

[0389] FIG. 31 illustrates an OpenCL implementation of software stack 2800 of FIG. 28, in accordance with at least one embodiment. In at least one embodiment, an OpenCL software stack 3100, on which an application 3101 may be launched, includes an OpenCL framework  
25 3110, an OpenCL runtime 3106, and a driver 3107. In at least one embodiment, OpenCL software stack 3100 executes on hardware 2909 that is not vendor-specific. As OpenCL is supported by devices developed by different vendors, specific OpenCL drivers may be required to interoperate with hardware from such vendors, in at least one embodiment.

30 [0390] In at least one embodiment, application 3101, OpenCL runtime 3106, device kernel driver 3107, and hardware 3108 may perform similar functionalities as application 2801, runtime 2805, device kernel driver 2806, and hardware 2807, respectively, that are discussed above in conjunction with FIG. 28. In at least one embodiment, application 3101 further includes an OpenCL kernel 3102 with code that is to be executed on a device.

35 [0391] In at least one embodiment, OpenCL defines a “platform” that allows a host to control devices connected to the host. In at least one embodiment, an OpenCL framework provides a

5 platform layer API and a runtime API, shown as platform API 3103 and runtime API 3105. In at least one embodiment, runtime API 3105 uses contexts to manage execution of kernels on devices. In at least one embodiment, each identified device may be associated with a respective context, which runtime API 3105 may use to manage command queues, program objects, and kernel objects, share memory objects, among other things, for that device. In at  
10 least one embodiment, platform API 3103 exposes functions that permit device contexts to be used to select and initialize devices, submit work to devices via command queues, and enable data transfer to and from devices, among other things. In addition, OpenCL framework provides various built-in functions (not shown), including math functions, relational functions, and image processing functions, among others, in at least one embodiment.

15 **[0392]** In at least one embodiment, a compiler 3104 is also included in OpenCL framework 3110. Source code may be compiled offline prior to executing an application or online during execution of an application, in at least one embodiment. In contrast to CUDA and ROCm, OpenCL applications in at least one embodiment may be compiled online by compiler 3104, which is included to be representative of any number of compilers that may be used to  
20 compile source code and/or IR code, such as Standard Portable Intermediate Representation (“SPIR-V”) code, into binary code. Alternatively, in at least one embodiment, OpenCL applications may be compiled offline, prior to execution of such applications.

**[0393]** FIG. 32 illustrates software that is supported by a programming platform, in accordance with at least one embodiment. In at least one embodiment, a programming  
25 platform 3204 is configured to support various programming models 3203, middlewares and/or libraries 3202, and frameworks 3201 that an application 3200 may rely upon. In at least one embodiment, application 3200 may be an AI/ML application implemented using, for example, a deep learning framework such as MXNet, PyTorch, or TensorFlow, which may rely on libraries such as cuDNN, NVIDIA Collective Communications Library  
30 (“NCCL”), and/or NVIDIA Developer Data Loading Library (“DALI”) CUDA libraries to provide accelerated computing on underlying hardware.

**[0394]** In at least one embodiment, programming platform 3204 may be one of a CUDA, ROCm, or OpenCL platform described above in conjunction with FIG. 29, FIG. 30, and FIG. 31, respectively. In at least one embodiment, programming platform 3204 supports multiple  
35 programming models 3203, which are abstractions of an underlying computing system permitting expressions of algorithms and data structures. Programming models 3203 may expose features of underlying hardware in order to improve performance, in at least one

embodiment. In at least one embodiment, programming models 3203 may include, but are not limited to, CUDA, HIP, OpenCL, C++ Accelerated Massive Parallelism (“C++AMP”), Open Multi-Processing (“OpenMP”), Open Accelerators (“OpenACC”), and/or Vulkan Compute.

[0395] In at least one embodiment, libraries and/or middlewares 3202 provide implementations of abstractions of programming models 3204. In at least one embodiment, such libraries include data and programming code that may be used by computer programs and leveraged during software development. In at least one embodiment, such middlewares include software that provides services to applications beyond those available from programming platform 3204. In at least one embodiment, libraries and/or middlewares 3202 may include, but are not limited to, cuBLAS, cuFFT, cuRAND, and other CUDA libraries, or rocBLAS, rocFFT, rocRAND, and other ROCm libraries. In addition, in at least one embodiment, libraries and/or middlewares 3202 may include NCCL and ROCm Communication Collectives Library (“RCCL”) libraries providing communication routines for GPUs, a MIOpen library for deep learning acceleration, and/or an Eigen library for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers, and related algorithms.

[0396] In at least one embodiment, application frameworks 3201 depend on libraries and/or middlewares 3202. In at least one embodiment, each of application frameworks 3201 is a software framework used to implement a standard structure of application software.

Returning to the AI/ML example discussed above, an AI/ML application may be implemented using a framework such as Caffe, Caffe2, TensorFlow, Keras, PyTorch, or MxNet deep learning frameworks, in at least one embodiment.

[0397] FIG. 33 illustrates compiling code to execute on one of programming platforms of FIGS. 28 - 31, in accordance with at least one embodiment. In at least one embodiment, a compiler 3301 receives source code 3300 that includes both host code as well as device code.

In at least one embodiment, compiler 3301 is configured to convert source code 3300 into host executable code 3302 for execution on a host and device executable code 3303 for execution on a device. In at least one embodiment, source code 3300 may either be compiled offline prior to execution of an application, or online during execution of an application.

[0398] In at least one embodiment, source code 3300 may include code in any programming language supported by compiler 3301, such as C++, C, Fortran, etc. In at least one embodiment, source code 3300 may be included in a single-source file having a mixture of

- 5 host code and device code, with locations of device code being indicated therein. In at least one embodiment, a single-source file may be a .cu file that includes CUDA code or a .hip.cpp file that includes HIP code. Alternatively, in at least one embodiment, source code 3300 may include multiple source code files, rather than a single-source file, into which host code and device code are separated.
- 10 **[0399]** In at least one embodiment, compiler 3301 is configured to compile source code 3300 into host executable code 3302 for execution on a host and device executable code 3303 for execution on a device. In at least one embodiment, compiler 3301 performs operations including parsing source code 3300 into an abstract system tree (AST), performing optimizations, and generating executable code. In at least one embodiment in which source
- 15 code 3300 includes a single-source file, compiler 3301 may separate device code from host code in such a single-source file, compile device code and host code into device executable code 3303 and host executable code 3302, respectively, and link device executable code 3303 and host executable code 3302 together in a single file, as discussed in greater detail below with respect to FIG. 34.
- 20 **[0400]** In at least one embodiment, host executable code 3302 and device executable code 3303 may be in any suitable format, such as binary code and/or IR code. In the case of CUDA, host executable code 3302 may include native object code and device executable code 3303 may include code in PTX intermediate representation, in at least one embodiment. In the case of ROCm, both host executable code 3302 and device executable code 3303 may
- 25 include target binary code, in at least one embodiment.
- [0401]** FIG. 34 is a more detailed illustration of compiling code to execute on one of programming platforms of FIGS. 28 - 31, in accordance with at least one embodiment. In at least one embodiment, a compiler 3401 is configured to receive source code 3400, compile source code 3400, and output an executable file 3410. In at least one embodiment, source
- 30 code 3400 is a single-source file, such as a .cu file, a .hip.cpp file, or a file in another format, that includes both host and device code. In at least one embodiment, compiler 3401 may be, but is not limited to, an NVIDIA CUDA compiler (“NVCC”) for compiling CUDA code in .cu files, or a HCC compiler for compiling HIP code in .hip.cpp files.
- [0402]** In at least one embodiment, compiler 3401 includes a compiler front end 3402, a host
- 35 compiler 3405, a device compiler 3406, and a linker 3409. In at least one embodiment, compiler front end 3402 is configured to separate device code 3404 from host code 3403 in

5 source code 3400. Device code 3404 is compiled by device compiler 3406 into device executable code 3408, which as described may include binary code or IR code, in at least one embodiment. Separately, host code 3403 is compiled by host compiler 3405 into host executable code 3407, in at least one embodiment. For NVCC, host compiler 3405 may be, but is not limited to, a general purpose C/C++ compiler that outputs native object code, while  
10 device compiler 3406 may be, but is not limited to, a Low Level Virtual Machine (“LLVM”)-based compiler that forks a LLVM compiler infrastructure and outputs PTX code or binary code, in at least one embodiment. For HCC, both host compiler 3405 and device compiler 3406 may be, but are not limited to, LLVM-based compilers that output target binary code, in at least one embodiment.

15 **[0403]** Subsequent to compiling source code 3400 into host executable code 3407 and device executable code 3408, linker 3409 links host and device executable code 3407 and 3408 together in executable file 3410, in at least one embodiment. In at least one embodiment, native object code for a host and PTX or binary code for a device may be linked together in an Executable and Linkable Format (“ELF”) file, which is a container format used to store  
20 object code.

**[0404]** FIG. 35 illustrates translating source code prior to compiling source code, in accordance with at least one embodiment. In at least one embodiment, source code 3500 is passed through a translation tool 3501, which translates source code 3500 into translated source code 3502. In at least one embodiment, a compiler 3503 is used to compile translated  
25 source code 3502 into host executable code 3504 and device executable code 3505 in a process that is similar to compilation of source code 3300 by compiler 3301 into host executable code 3302 and device executable 3303, as discussed above in conjunction with FIG. 33.

**[0405]** In at least one embodiment, a translation performed by translation tool 3501 is used to  
30 port source 3500 for execution in a different environment than that in which it was originally intended to run. In at least one embodiment, translation tool 3501 may include, but is not limited to, a HIP translator that is used to “hipify” CUDA code intended for a CUDA platform into HIP code that can be compiled and executed on a ROCm platform. In at least one embodiment, translation of source code 3500 may include parsing source code 3500 and  
35 converting calls to API(s) provided by one programming model (e.g., CUDA) into corresponding calls to API(s) provided by another programming model (e.g., HIP), as discussed in greater detail below in conjunction with FIGS. 36A - 37. Returning to the

example of hipifying CUDA code, calls to CUDA runtime API, CUDA driver API, and/or CUDA libraries may be converted to corresponding HIP API calls, in at least one embodiment. In at least one embodiment, automated translations performed by translation tool 3501 may sometimes be incomplete, requiring additional, manual effort to fully port source code 3500.

### Configuring GPUs for General-Purpose Computing

[0406] The following FIGS. set forth, without limitation, exemplary architectures for compiling and executing compute source code, in accordance with at least one embodiment.

[0407] FIG. 36A illustrates a system 36A00 configured to compile and execute CUDA source code 3610 using different types of processing units, in accordance with at least one embodiment. In at least one embodiment, system 36A00 includes, without limitation, CUDA source code 3610, a CUDA compiler 3650, host executable code 3670(1), host executable code 3670(2), CUDA device executable code 3684, a CPU 3690, a CUDA-enabled GPU 3694, a GPU 3692, a CUDA to HIP translation tool 3620, HIP source code 3630, a HIP compiler driver 3640, an HCC 3660, and HCC device executable code 3682.

[0408] In at least one embodiment, CUDA source code 3610 is a collection of human-readable code in a CUDA programming language. In at least one embodiment, CUDA code is human-readable code in a CUDA programming language. In at least one embodiment, a CUDA programming language is an extension of the C++ programming language that includes, without limitation, mechanisms to define device code and distinguish between device code and host code. In at least one embodiment, device code is source code that, after compilation, is executable in parallel on a device. In at least one embodiment, a device may be a processor that is optimized for parallel instruction processing, such as CUDA-enabled GPU 3690, GPU 36192, or another GPGPU, etc. In at least one embodiment, host code is source code that, after compilation, is executable on a host. In at least one embodiment, a host is a processor that is optimized for sequential instruction processing, such as CPU 3690.

[0409] In at least one embodiment, CUDA source code 3610 includes, without limitation, any number (including zero) of global functions 3612, any number (including zero) of device functions 3614, any number (including zero) of host functions 3616, and any number (including zero) of host/device functions 3618. In at least one embodiment, global functions 3612, device functions 3614, host functions 3616, and host/device functions 3618 may be mixed in CUDA source code 3610. In at least one embodiment, each of global functions 3612

5 is executable on a device and callable from a host. In at least one embodiment, one or more of global functions 3612 may therefore act as entry points to a device. In at least one embodiment, each of global functions 3612 is a kernel. In at least one embodiment and in a technique known as dynamic parallelism, one or more of global functions 3612 defines a kernel that is executable on a device and callable from such a device. In at least one  
10 embodiment, a kernel is executed N (where N is any positive integer) times in parallel by N different threads on a device during execution.

[0410] In at least one embodiment, each of device functions 3614 is executed on a device and callable from such a device only. In at least one embodiment, each of host functions 3616 is executed on a host and callable from such a host only. In at least one embodiment, each of  
15 host/device functions 3616 defines both a host version of a function that is executable on a host and callable from such a host only and a device version of the function that is executable on a device and callable from such a device only.

[0411] In at least one embodiment, CUDA source code 3610 may also include, without limitation, any number of calls to any number of functions that are defined via a CUDA  
20 runtime API 3602. In at least one embodiment, CUDA runtime API 3602 may include, without limitation, any number of functions that execute on a host to allocate and deallocate device memory, transfer data between host memory and device memory, manage systems with multiple devices, etc. In at least one embodiment, CUDA source code 3610 may also include any number of calls to any number of functions that are specified in any number of  
25 other CUDA APIs. In at least one embodiment, a CUDA API may be any API that is designed for use by CUDA code. In at least one embodiment, CUDA APIs include, without limitation, CUDA runtime API 3602, a CUDA driver API, APIs for any number of CUDA libraries, etc. In at least one embodiment and relative to CUDA runtime API 3602, a CUDA driver API is a lower-level API but provides finer-grained control of a device. In at least one  
30 embodiment, examples of CUDA libraries include, without limitation, cuBLAS, cuFFT, cuRAND, cuDNN, etc.

[0412] In at least one embodiment, CUDA compiler 3650 compiles input CUDA code (e.g., CUDA source code 3610) to generate host executable code 3670(1) and CUDA device executable code 3684. In at least one embodiment, CUDA compiler 3650 is NVCC. In at  
35 least one embodiment, host executable code 3670(1) is a compiled version of host code included in input source code that is executable on CPU 3690. In at least one embodiment, CPU 3690 may be any processor that is optimized for sequential instruction processing.

- 5     **[0413]** In at least one embodiment, CUDA device executable code 3684 is a compiled version of device code included in input source code that is executable on CUDA-enabled GPU 3694. In at least one embodiment, CUDA device executable code 3684 includes, without limitation, binary code. In at least one embodiment, CUDA device executable code 3684 includes, without limitation, IR code, such as PTX code, that is further compiled at  
10     runtime into binary code for a specific target device (e.g., CUDA-enabled GPU 3694) by a device driver. In at least one embodiment, CUDA-enabled GPU 3694 may be any processor that is optimized for parallel instruction processing and that supports CUDA. In at least one embodiment, CUDA-enabled GPU 3694 is developed by NVIDIA Corporation of Santa Clara, CA.
- 15     **[0414]** In at least one embodiment, CUDA to HIP translation tool 3620 is configured to translate CUDA source code 3610 to functionally similar HIP source code 3630. In a least one embodiment, HIP source code 3630 is a collection of human-readable code in a HIP programming language. In at least one embodiment, HIP code is human-readable code in a HIP programming language. In at least one embodiment, a HIP programming language is an  
20     extension of the C++ programming language that includes, without limitation, functionally similar versions of CUDA mechanisms to define device code and distinguish between device code and host code. In at least one embodiment, a HIP programming language may include a subset of functionality of a CUDA programming language. In at least one embodiment, for example, a HIP programming language includes, without limitation, mechanism(s) to define  
25     global functions 3612, but such a HIP programming language may lack support for dynamic parallelism and therefore global functions 3612 defined in HIP code may be callable from a host only.
- 30     **[0415]** In at least one embodiment, HIP source code 3630 includes, without limitation, any number (including zero) of global functions 3612, any number (including zero) of device functions 3614, any number (including zero) of host functions 3616, and any number (including zero) of host/device functions 3618. In at least one embodiment, HIP source code 3630 may also include any number of calls to any number of functions that are specified in a HIP runtime API 3632. In at least one embodiment, HIP runtime API 3632 includes, without limitation, functionally similar versions of a subset of functions included in CUDA runtime  
35     API 3602. In at least one embodiment, HIP source code 3630 may also include any number of calls to any number of functions that are specified in any number of other HIP APIs. In at least one embodiment, a HIP API may be any API that is designed for use by HIP code

5 and/or ROCm. In at least one embodiment, HIP APIs include, without limitation, HIP runtime API 3632, a HIP driver API, APIs for any number of HIP libraries, APIs for any number of ROCm libraries, etc.

[0416] In at least one embodiment, CUDA to HIP translation tool 3620 converts each kernel call in CUDA code from a CUDA syntax to a HIP syntax and converts any number of other  
10 CUDA calls in CUDA code to any number of other functionally similar HIP calls. In at least one embodiment, a CUDA call is a call to a function specified in a CUDA API, and a HIP call is a call to a function specified in a HIP API. In at least one embodiment, CUDA to HIP translation tool 3620 converts any number of calls to functions specified in CUDA runtime API 3602 to any number of calls to functions specified in HIP runtime API 3632.

15 [0417] In at least one embodiment, CUDA to HIP translation tool 3620 is a tool known as hipify-perl that executes a text-based translation process. In at least one embodiment, CUDA to HIP translation tool 3620 is a tool known as hipify-clang that, relative to hipify-perl, executes a more complex and more robust translation process that involves parsing CUDA code using clang (a compiler front-end) and then translating resulting symbols. In at least one  
20 embodiment, properly converting CUDA code to HIP code may require modifications (e.g., manual edits) in addition to those performed by CUDA to HIP translation tool 3620.

[0418] In at least one embodiment, HIP compiler driver 3640 is a front end that determines a target device 3646 and then configures a compiler that is compatible with target device 3646 to compile HIP source code 3630. In at least one embodiment, target device 3646 is a  
25 processor that is optimized for parallel instruction processing. In at least one embodiment, HIP compiler driver 3640 may determine target device 3646 in any technically feasible fashion.

[0419] In at least one embodiment, if target device 3646 is compatible with CUDA (e.g., CUDA-enabled GPU 3694), then HIP compiler driver 3640 generates a HIP/NVCC  
30 compilation command 3642. In at least one embodiment and as described in greater detail in conjunction with FIG. 36B, HIP/NVCC compilation command 3642 configures CUDA compiler 3650 to compile HIP source code 3630 using, without limitation, a HIP to CUDA translation header and a CUDA runtime library. In at least one embodiment and in response to HIP/NVCC compilation command 3642, CUDA compiler 3650 generates host executable  
35 code 3670(1) and CUDA device executable code 3684.

5     **[0420]** In at least one embodiment, if target device 3646 is not compatible with CUDA, then  
HIP compiler driver 3640 generates a HIP/HCC compilation command 3644. In at least one  
embodiment and as described in greater detail in conjunction with FIG. 36C, HIP/HCC  
compilation command 3644 configures HCC 3660 to compile HIP source code 3630 using,  
without limitation, an HCC header and a HIP/HCC runtime library. In at least one  
10   embodiment and in response to HIP/HCC compilation command 3644, HCC 3660 generates  
host executable code 3670(2) and HCC device executable code 3682. In at least one  
embodiment, HCC device executable code 3682 is a compiled version of device code  
included in HIP source code 3630 that is executable on GPU 3692. In at least one  
embodiment, GPU 3692 may be any processor that is optimized for parallel instruction  
15   processing, is not compatible with CUDA, and is compatible with HCC. In at least one  
embodiment, GPU 3692 is developed by AMD Corporation of Santa Clara, CA. In at least  
one embodiment GPU, 3692 is a non-CUDA-enabled GPU 3692.

**[0421]** For explanatory purposes only, three different flows that may be implemented in at  
least one embodiment to compile CUDA source code 3610 for execution on CPU 3690 and  
20   different devices are depicted in FIG. 36A. In at least one embodiment, a direct CUDA flow  
compiles CUDA source code 3610 for execution on CPU 3690 and CUDA-enabled GPU  
3694 without translating CUDA source code 3610 to HIP source code 3630. In at least one  
embodiment, an indirect CUDA flow translates CUDA source code 3610 to HIP source code  
3630 and then compiles HIP source code 3630 for execution on CPU 3690 and CUDA-  
25   enabled GPU 3694. In at least one embodiment, a CUDA/HCC flow translates CUDA source  
code 3610 to HIP source code 3630 and then compiles HIP source code 3630 for execution  
on CPU 3690 and GPU 3692.

**[0422]** A direct CUDA flow that may be implemented in at least one embodiment is depicted  
via dashed lines and a series of bubbles annotated A1-A3. In at least one embodiment and as  
30   depicted with bubble annotated A1, CUDA compiler 3650 receives CUDA source code 3610  
and a CUDA compile command 3648 that configures CUDA compiler 3650 to compile  
CUDA source code 3610. In at least one embodiment, CUDA source code 3610 used in a  
direct CUDA flow is written in a CUDA programming language that is based on a  
programming language other than C++ (e.g., C, Fortran, Python, Java, etc.). In at least one  
35   embodiment and in response to CUDA compile command 3648, CUDA compiler 3650  
generates host executable code 3670(1) and CUDA device executable code 3684 (depicted  
with bubble annotated A2). In at least one embodiment and as depicted with bubble annotated

5 A3, host executable code 3670(1) and CUDA device executable code 3684 may be executed on, respectively, CPU 3690 and CUDA-enabled GPU 3694. In at least one embodiment, CUDA device executable code 3684 includes, without limitation, binary code. In at least one embodiment, CUDA device executable code 3684 includes, without limitation, PTX code and is further compiled into binary code for a specific target device at runtime.

10 **[0423]** An indirect CUDA flow that may be implemented in at least one embodiment is depicted via dotted lines and a series of bubbles annotated B1-B6. In at least one embodiment and as depicted with bubble annotated B1, CUDA to HIP translation tool 3620 receives CUDA source code 3610. In at least one embodiment and as depicted with bubble annotated B2, CUDA to HIP translation tool 3620 translates CUDA source code 3610 to HIP source  
15 code 3630. In at least one embodiment and as depicted with bubble annotated B3, HIP compiler driver 3640 receives HIP source code 3630 and determines that target device 3646 is CUDA-enabled.

**[0424]** In at least one embodiment and as depicted with bubble annotated B4, HIP compiler driver 3640 generates HIP/NVCC compilation command 3642 and transmits both HIP/NVCC  
20 compilation command 3642 and HIP source code 3630 to CUDA compiler 3650. In at least one embodiment and as described in greater detail in conjunction with FIG. 36B, HIP/NVCC compilation command 3642 configures CUDA compiler 3650 to compile HIP source code 3630 using, without limitation, a HIP to CUDA translation header and a CUDA runtime library. In at least one embodiment and in response to HIP/NVCC compilation command  
25 3642, CUDA compiler 3650 generates host executable code 3670(1) and CUDA device executable code 3684 (depicted with bubble annotated B5). In at least one embodiment and as depicted with bubble annotated B6, host executable code 3670(1) and CUDA device executable code 3684 may be executed on, respectively, CPU 3690 and CUDA-enabled GPU 3694. In at least one embodiment, CUDA device executable code 3684 includes, without  
30 limitation, binary code. In at least one embodiment, CUDA device executable code 3684 includes, without limitation, PTX code and is further compiled into binary code for a specific target device at runtime.

**[0425]** A CUDA/HCC flow that may be implemented in at least one embodiment is depicted via solid lines and a series of bubbles annotated C1-C6. In at least one embodiment and as  
35 depicted with bubble annotated C1, CUDA to HIP translation tool 3620 receives CUDA source code 3610. In at least one embodiment and as depicted with bubble annotated C2, CUDA to HIP translation tool 3620 translates CUDA source code 3610 to HIP source code

5 3630. In at least one embodiment and as depicted with bubble annotated C3, HIP compiler driver 3640 receives HIP source code 3630 and determines that target device 3646 is not CUDA-enabled.

[0426] In at least one embodiment, HIP compiler driver 3640 generates HIP/HCC compilation command 3644 and transmits both HIP/HCC compilation command 3644 and  
10 HIP source code 3630 to HCC 3660 (depicted with bubble annotated C4). In at least one embodiment and as described in greater detail in conjunction with FIG. 36C, HIP/HCC compilation command 3644 configures HCC 3660 to compile HIP source code 3630 using, without limitation, an HCC header and a HIP/HCC runtime library. In at least one embodiment and in response to HIP/HCC compilation command 3644, HCC 3660 generates  
15 host executable code 3670(2) and HCC device executable code 3682 (depicted with bubble annotated C5). In at least one embodiment and as depicted with bubble annotated C6, host executable code 3670(2) and HCC device executable code 3682 may be executed on, respectively, CPU 3690 and GPU 3692.

[0427] In at least one embodiment, after CUDA source code 3610 is translated to HIP source  
20 code 3630, HIP compiler driver 3640 may subsequently be used to generate executable code for either CUDA-enabled GPU 3694 or GPU 3692 without re-executing CUDA to HIP translation tool 3620. In at least one embodiment, CUDA to HIP translation tool 3620 translates CUDA source code 3610 to HIP source code 3630 that is then stored in memory. In at least one embodiment, HIP compiler driver 3640 then configures HCC 3660 to generate  
25 host executable code 3670(2) and HCC device executable code 3682 based on HIP source code 3630. In at least one embodiment, HIP compiler driver 3640 subsequently configures CUDA compiler 3650 to generate host executable code 3670(1) and CUDA device executable code 3684 based on stored HIP source code 3630.

[0428] FIG. 36B illustrates a system 3604 configured to compile and execute CUDA source  
30 code 3610 of FIG. 36A using CPU 3690 and CUDA-enabled GPU 3694, in accordance with at least one embodiment. In at least one embodiment, system 3604 includes, without limitation, CUDA source code 3610, CUDA to HIP translation tool 3620, HIP source code 3630, HIP compiler driver 3640, CUDA compiler 3650, host executable code 3670(1), CUDA device executable code 3684, CPU 3690, and CUDA-enabled GPU 3694.

35 [0429] In at least one embodiment and as described previously herein in conjunction with FIG. 36A, CUDA source code 3610 includes, without limitation, any number (including zero)

- 5 of global functions 3612, any number (including zero) of device functions 3614, any number (including zero) of host functions 3616, and any number (including zero) of host/device functions 3618. In at least one embodiment, CUDA source code 3610 also includes, without limitation, any number of calls to any number of functions that are specified in any number of CUDA APIs.
- 10 **[0430]** In at least one embodiment, CUDA to HIP translation tool 3620 translates CUDA source code 3610 to HIP source code 3630. In at least one embodiment, CUDA to HIP translation tool 3620 converts each kernel call in CUDA source code 3610 from a CUDA syntax to a HIP syntax and converts any number of other CUDA calls in CUDA source code 3610 to any number of other functionally similar HIP calls.
- 15 **[0431]** In at least one embodiment, HIP compiler driver 3640 determines that target device 3646 is CUDA-enabled and generates HIP/NVCC compilation command 3642. In at least one embodiment, HIP compiler driver 3640 then configures CUDA compiler 3650 via HIP/NVCC compilation command 3642 to compile HIP source code 3630. In at least one embodiment, HIP compiler driver 3640 provides access to a HIP to CUDA translation header
- 20 3652 as part of configuring CUDA compiler 3650. In at least one embodiment, HIP to CUDA translation header 3652 translates any number of mechanisms (e.g., functions) specified in any number of HIP APIs to any number of mechanisms specified in any number of CUDA APIs. In at least one embodiment, CUDA compiler 3650 uses HIP to CUDA translation header 3652 in conjunction with a CUDA runtime library 3654 corresponding to CUDA
- 25 runtime API 3602 to generate host executable code 3670(1) and CUDA device executable code 3684. In at least one embodiment, host executable code 3670(1) and CUDA device executable code 3684 may then be executed on, respectively, CPU 3690 and CUDA-enabled GPU 3694. In at least one embodiment, CUDA device executable code 3684 includes, without limitation, binary code. In at least one embodiment, CUDA device executable code
- 30 3684 includes, without limitation, PTX code and is further compiled into binary code for a specific target device at runtime.
- [0432]** FIG. 36C illustrates a system 3606 configured to compile and execute CUDA source code 3610 of FIG. 36A using CPU 3690 and non-CUDA-enabled GPU 3692, in accordance with at least one embodiment. In at least one embodiment, system 3606 includes, without
- 35 limitation, CUDA source code 3610, CUDA to HIP translation tool 3620, HIP source code 3630, HIP compiler driver 3640, HCC 3660, host executable code 3670(2), HCC device executable code 3682, CPU 3690, and GPU 3692.

5     **[0433]** In at least one embodiment and as described previously herein in conjunction with FIG. 36A, CUDA source code 3610 includes, without limitation, any number (including zero) of global functions 3612, any number (including zero) of device functions 3614, any number (including zero) of host functions 3616, and any number (including zero) of host/device functions 3618. In at least one embodiment, CUDA source code 3610 also includes, without  
10    limitation, any number of calls to any number of functions that are specified in any number of CUDA APIs.

**[0434]** In at least one embodiment, CUDA to HIP translation tool 3620 translates CUDA source code 3610 to HIP source code 3630. In at least one embodiment, CUDA to HIP translation tool 3620 converts each kernel call in CUDA source code 3610 from a CUDA  
15    syntax to a HIP syntax and converts any number of other CUDA calls in source code 3610 to any number of other functionally similar HIP calls.

**[0435]** In at least one embodiment, HIP compiler driver 3640 subsequently determines that target device 3646 is not CUDA-enabled and generates HIP/HCC compilation command 3644. In at least one embodiment, HIP compiler driver 3640 then configures HCC 3660 to  
20    execute HIP/HCC compilation command 3644 to compile HIP source code 3630. In at least one embodiment, HIP/HCC compilation command 3644 configures HCC 3660 to use, without limitation, a HIP/HCC runtime library 3658 and an HCC header 3656 to generate host executable code 3670(2) and HCC device executable code 3682. In at least one  
25    embodiment, HIP/HCC runtime library 3658 corresponds to HIP runtime API 3632. In at least one embodiment, HCC header 3656 includes, without limitation, any number and type of interoperability mechanisms for HIP and HCC. In at least one embodiment, host executable code 3670(2) and HCC device executable code 3682 may be executed on, respectively, CPU 3690 and GPU 3692.

**[0436]** FIG. 37 illustrates an exemplary kernel translated by CUDA-to-HIP translation tool  
30    3620 of FIG. 36C, in accordance with at least one embodiment. In at least one embodiment, CUDA source code 3610 partitions an overall problem that a given kernel is designed to solve into relatively coarse sub-problems that can independently be solved using thread blocks. In at least one embodiment, each thread block includes, without limitation, any  
35    number of threads. In at least one embodiment, each sub-problem is partitioned into relatively fine pieces that can be solved cooperatively in parallel by threads within a thread block. In at least one embodiment, threads within a thread block can cooperate by sharing data through shared memory and by synchronizing execution to coordinate memory accesses.

- 5     **[0437]** In at least one embodiment, CUDA source code 3610 organizes thread blocks associated with a given kernel into a one-dimensional, a two-dimensional, or a three-dimensional grid of thread blocks. In at least one embodiment, each thread block includes, without limitation, any number of threads, and a grid includes, without limitation, any number of thread blocks.
- 10    **[0438]** In at least one embodiment, a kernel is a function in device code that is defined using a “\_\_global\_\_” declaration specifier. In at least one embodiment, the dimension of a grid that executes a kernel for a given kernel call and associated streams are specified using a CUDA kernel launch syntax 3710. In at least one embodiment, CUDA kernel launch syntax 3710 is specified as “KernelName<<<GridSize, BlockSize, SharedMemorySize,
- 15    Stream>>>(KernelArguments);”. In at least one embodiment, an execution configuration syntax is a “<<<...>>>” construct that is inserted between a kernel name (“KernelName”) and a parenthesized list of kernel arguments (“KernelArguments”). In at least one embodiment, CUDA kernel launch syntax 3710 includes, without limitation, a CUDA launch function syntax instead of an execution configuration syntax.
- 20    **[0439]** In at least one embodiment, “GridSize” is of a type dim3 and specifies the dimension and size of a grid. In at least one embodiment, type dim3 is a CUDA-defined structure that includes, without limitation, unsigned integers x, y, and z. In at least one embodiment, if z is not specified, then z defaults to one. In at least one embodiment, if y is not specified, then y defaults to one. In at least one embodiment, the number of thread blocks in a grid is equal to
- 25    the product of GridSize.x, GridSize.y, and GridSize.z. In at least one embodiment, “BlockSize” is of type dim3 and specifies the dimension and size of each thread block. In at least one embodiment, the number of threads per thread block is equal to the product of BlockSize.x, BlockSize.y, and BlockSize.z. In at least one embodiment, each thread that executes a kernel is given a unique thread ID that is accessible within the kernel through a
- 30    built-in variable (e.g., “threadIdx”).
- [0440]** In at least one embodiment and with respect to CUDA kernel launch syntax 3710, “SharedMemorySize” is an optional argument that specifies a number of bytes in a shared memory that is dynamically allocated per thread block for a given kernel call in addition to statically allocated memory. In at least one embodiment and with respect to CUDA kernel
- 35    launch syntax 3710, SharedMemorySize defaults to zero. In at least one embodiment and with respect to CUDA kernel launch syntax 3710, “Stream” is an optional argument that specifies an associated stream and defaults to zero to specify a default stream. In at least one

5 embodiment, a stream is a sequence of commands (possibly issued by different host threads) that execute in order. In at least one embodiment, different streams may execute commands out of order with respect to one another or concurrently.

[0441] In at least one embodiment, CUDA source code 3610 includes, without limitation, a kernel definition for an exemplary kernel “MatAdd” and a main function. In at least one  
 10 embodiment, main function is host code that executes on a host and includes, without limitation, a kernel call that causes kernel MatAdd to execute on a device. In at least one embodiment and as shown, kernel MatAdd adds two matrices A and B of size  $N \times N$ , where N is a positive integer, and stores the result in a matrix C. In at least one embodiment, main function defines a threadsPerBlock variable as 16 by 16 and a numBlocks variable as  $N/16$  by  $N/16$ . In at least one embodiment, main function then specifies kernel call  
 15 “MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);”. In at least one embodiment and as per CUDA kernel launch syntax 3710, kernel MatAdd is executed using a grid of thread blocks having a dimension  $N/16$  by  $N/16$ , where each thread block has a dimension of 16 by 16. In at least one embodiment, each thread block includes 256 threads, a grid is created with  
 20 enough blocks to have one thread per matrix element, and each thread in such a grid executes kernel MatAdd to perform one pair-wise addition.

[0442] In at least one embodiment, while translating CUDA source code 3610 to HIP source code 3630, CUDA to HIP translation tool 3620 translates each kernel call in CUDA source code 3610 from CUDA kernel launch syntax 3710 to a HIP kernel launch syntax 3720 and  
 25 converts any number of other CUDA calls in source code 3610 to any number of other functionally similar HIP calls. In at least one embodiment, HIP kernel launch syntax 3720 is specified as “hipLaunchKernelGGL(KernelName, GridSize, BlockSize, SharedMemorySize, Stream, KernelArguments);”. In at least one embodiment, each of KernelName, GridSize, BlockSize, ShareMemorySize, Stream, and KernelArguments has the same meaning in HIP  
 30 kernel launch syntax 3720 as in CUDA kernel launch syntax 3710 (described previously herein). In at least one embodiment, arguments SharedMemorySize and Stream are required in HIP kernel launch syntax 3720 and are optional in CUDA kernel launch syntax 3710.

[0443] In at least one embodiment, a portion of HIP source code 3630 depicted in FIG. 37 is identical to a portion of CUDA source code 3610 depicted in FIG. 37 except for a kernel call  
 35 that causes kernel MatAdd to execute on a device. In at least one embodiment, kernel MatAdd is defined in HIP source code 3630 with the same “\_\_global\_\_” declaration specifier with which kernel MatAdd is defined in CUDA source code 3610. In at least one

5     embodiment, a kernel call in HIP source code 3630 is “hipLaunchKernelGGL(MatAdd, numBlocks, threadsPerBlock, 0, 0, A, B, C);”, while a corresponding kernel call in CUDA source code 3610 is “MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);”.

[0444] FIG. 38 illustrates non-CUDA-enabled GPU 3692 of FIG. 36C in greater detail, in accordance with at least one embodiment. In at least one embodiment, GPU 3692 is  
10     developed by AMD corporation of Santa Clara. In at least one embodiment, GPU 3692 can be configured to perform compute operations in a highly-parallel fashion. In at least one embodiment, GPU 3692 is configured to execute graphics pipeline operations such as draw commands, pixel operations, geometric computations, and other operations associated with rendering an image to a display. In at least one embodiment, GPU 3692 is configured to  
15     execute operations unrelated to graphics. In at least one embodiment, GPU 3692 is configured to execute both operations related to graphics and operations unrelated to graphics. In at least one embodiment, GPU 3692 can be configured to execute device code included in HIP source code 3630.

[0445] In at least one embodiment, GPU 3692 includes, without limitation, any number of  
20     programmable processing units 3820, a command processor 3810, an L2 cache 3822, memory controllers 3870, DMA engines 3880(1), system memory controllers 3882, DMA engines 3880(2), and GPU controllers 3884. In at least one embodiment, each programmable processing unit 3820 includes, without limitation, a workload manager 3830 and any number of compute units 3840. In at least one embodiment, command processor 3810 reads  
25     commands from one or more command queues (not shown) and distributes commands to workload managers 3830. In at least one embodiment, for each programmable processing unit 3820, associated workload manager 3830 distributes work to compute units 3840 included in programmable processing unit 3820. In at least one embodiment, each compute unit 3840 may execute any number of thread blocks, but each thread block executes on a single  
30     compute unit 3840. In at least one embodiment, a workgroup is a thread block.

[0446] In at least one embodiment, each compute unit 3840 includes, without limitation, any number of SIMD units 3850 and a shared memory 3860. In at least one embodiment, each SIMD unit 3850 implements a SIMD architecture and is configured to perform operations in parallel. In at least one embodiment, each SIMD unit 3850 includes, without limitation, a  
35     vector ALU 3852 and a vector register file 3854. In at least one embodiment, each SIMD unit 3850 executes a different warp. In at least one embodiment, a warp is a group of threads (e.g., 16 threads), where each thread in the warp belongs to a single thread block and is configured

- 5 to process a different set of data based on a single set of instructions. In at least one embodiment, predication can be used to disable one or more threads in a warp. In at least one embodiment, a lane is a thread. In at least one embodiment, a work item is a thread. In at least one embodiment, a wavefront is a warp. In at least one embodiment, different wavefronts in a thread block may synchronize together and communicate via shared memory 3860.
- 10 [0447] In at least one embodiment, programmable processing units 3820 are referred to as “shader engines.” In at least one embodiment, each programmable processing unit 3820 includes, without limitation, any amount of dedicated graphics hardware in addition to compute units 3840. In at least one embodiment, each programmable processing unit 3820 includes, without limitation, any number (including zero) of geometry processors, any
- 15 number (including zero) of rasterizers, any number (including zero) of render back ends, workload manager 3830, and any number of compute units 3840.
- [0448] In at least one embodiment, compute units 3840 share L2 cache 3822. In at least one embodiment, L2 cache 3822 is partitioned. In at least one embodiment, a GPU memory 3890 is accessible by all compute units 3840 in GPU 3692. In at least one embodiment, memory
- 20 controllers 3870 and system memory controllers 3882 facilitate data transfers between GPU 3692 and a host, and DMA engines 3880(1) enable asynchronous memory transfers between GPU 3692 and such a host. In at least one embodiment, memory controllers 3870 and GPU controllers 3884 facilitate data transfers between GPU 3692 and other GPUs 3692, and DMA engines 3880(2) enable asynchronous memory transfers between GPU 3692 and other GPUs
- 25 3692.
- [0449] In at least one embodiment, GPU 3692 includes, without limitation, any amount and type of system interconnect that facilitates data and control transmissions across any number and type of directly or indirectly linked components that may be internal or external to GPU 3692. In at least one embodiment, GPU 3692 includes, without limitation, any number and
- 30 type of I/O interfaces (e.g., PCIe) that are coupled to any number and type of peripheral devices. In at least one embodiment, GPU 3692 may include, without limitation, any number (including zero) of display engines and any number (including zero) of multimedia engines. In at least one embodiment, GPU 3692 implements a memory subsystem that includes, without limitation, any amount and type of memory controllers (e.g., memory controllers
- 35 3870 and system memory controllers 3882) and memory devices (e.g., shared memories 3860) that may be dedicated to one component or shared among multiple components. In at least one embodiment, GPU 3692 implements a cache subsystem that includes, without

5 limitation, one or more cache memories (e.g., L2 cache 3822) that may each be private to or shared between any number of components (e.g., SIMD units 3850, compute units 3840, and programmable processing units 3820).

[0450] FIG. 39 illustrates how threads of an exemplary CUDA grid 3920 are mapped to different compute units 3840 of FIG. 38, in accordance with at least one embodiment. In at least one embodiment and for explanatory purposes only, grid 3920 has a GridSize of BX by BY by 1 and a BlockSize of TX by TY by 1. In at least one embodiment, grid 3920 therefore includes, without limitation, (BX \* BY) thread blocks 3930 and each thread block 3930 includes, without limitation, (TX \* TY) threads 3940. Threads 3940 are depicted in FIG. 39 as squiggly arrows.

15 [0451] In at least one embodiment, grid 3920 is mapped to programmable processing unit 3820(1) that includes, without limitation, compute units 3840(1)-3840(C). In at least one embodiment and as shown, (BJ \* BY) thread blocks 3930 are mapped to compute unit 3840(1), and the remaining thread blocks 3930 are mapped to compute unit 3840(2). In at least one embodiment, each thread block 3930 may include, without limitation, any number of warps, and each warp is mapped to a different SIMD unit 3850 of FIG. 38.

[0452] In at least one embodiment, warps in a given thread block 3930 may synchronize together and communicate through shared memory 3860 included in associated compute unit 3840. For example and in at least one embodiment, warps in thread block 3930(BJ,1) can synchronize together and communicate through shared memory 3860(1). For example and in at least one embodiment, warps in thread block 3930(BJ+1,1) can synchronize together and communicate through shared memory 3860(2).

[0453] FIG. 40 illustrates how to migrate existing CUDA code to Data Parallel C++ code, in accordance with at least one embodiment. Data Parallel C++ (DPC++) may refer to an open, standards-based alternative to single-architecture proprietary languages that allows developers to reuse code across hardware targets (CPUs and accelerators such as GPUs and FPGAs) and also perform custom tuning for a specific accelerator. DPC++ use similar and/or identical C and C++ constructs in accordance with ISO C++ which developers may be familiar with. DPC++ incorporates standard SYCL from The Khronos Group to support data parallelism and heterogeneous programming. SYCL refers to a cross-platform abstraction layer that builds on underlying concepts, portability and efficiency of OpenCL that enables code for heterogeneous processors to be written in a “single-source” style using standard

5 C++. SYCL may enable single source development where C++ template functions can contain both host and device code to construct complex algorithms that use OpenCL acceleration, and then re-use them throughout their source code on different types of data.

[0454] In at least one embodiment, a DPC++ compiler is used to compile DPC++ source code which can be deployed across diverse hardware targets. In at least one embodiment, a  
10 DPC++ compiler is used to generate DPC++ applications that can be deployed across diverse hardware targets and a DPC++ compatibility tool can be used to migrate CUDA applications to a multiplatform program in DPC++. In at least one embodiment, a DPC++ base tool kit includes a DPC++ compiler to deploy applications across diverse hardware targets; a DPC++ library to increase productivity and performance across CPUs, GPUs, and FPGAs; a DPC++  
15 compatibility tool to migrate CUDA applications to multi-platform applications; and any suitable combination thereof.

[0455] In at least one embodiment, a DPC++ programming model is utilized to simplify one or more aspects relating to programming CPUs and accelerators by using modern C++ features to express parallelism with a programming language called Data Parallel C++. DPC++  
20 programming language may be utilized to code reuse for hosts (e.g., a CPU) and accelerators (e.g., a GPU or FPGA) using a single source language, with execution and memory dependencies being clearly communicated. Mappings within DPC++ code can be used to transition an application to run on a hardware or set of hardware devices that best accelerates a workload. A host may be available to simplify development and debugging of device code,  
25 even on platforms that do not have an accelerator available.

[0456] In at least one embodiment, CUDA source code 4000 is provided as an input to a DPC++ compatibility tool 4002 to generate human readable DPC++ 4004. In at least one embodiment, human readable DPC++ 4004 includes inline comments generated by DPC++ compatibility tool 4002 that guides a developer on how and/or where to modify DPC++ code  
30 to complete coding and tuning to desired performance 4006, thereby generating DPC++ source code 4008.

[0457] In at least one embodiment, CUDA source code 4000 is or includes a collection of human-readable source code in a CUDA programming language. In at least one embodiment, CUDA source code 4000 is human-readable source code in a CUDA programming language.  
35 In at least one embodiment, a CUDA programming language is an extension of the C++ programming language that includes, without limitation, mechanisms to define device code

5 and distinguish between device code and host code. In at least one embodiment, device code is source code that, after compilation, is executable on a device (e.g., GPU or FPGA) and may include or more parallelizable workflows that can be executed on one or more processor cores of a device. In at least one embodiment, a device may be a processor that is optimized for parallel instruction processing, such as CUDA-enabled GPU, GPU, or another GPGPU,  
10 etc. In at least one embodiment, host code is source code that, after compilation, is executable on a host. In at least one embodiment, some or all of host code and device code can be executed in parallel across a CPU and GPU/FPGA. In at least one embodiment, a host is a processor that is optimized for sequential instruction processing, such as CPU. CUDA source code 4000 described in connection with FIG. 40 may be in accordance with those discussed  
15 elsewhere in this document.

[0458] In at least one embodiment, DPC++ compatibility tool 4002 refers to an executable tool, program, application, or any other suitable type of tool that is used to facilitate migration of CUDA source code 4000 to DPC++ source code 4008. In at least one embodiment, DPC++ compatibility tool 4002 is a command-line-based code migration tool available as  
20 part of a DPC++ tool kit that is used to port existing CUDA sources to DPC++. In at least one embodiment, DPC++ compatibility tool 4002 converts some or all source code of a CUDA application from CUDA to DPC++ and generates a resulting file that is written at least partially in DPC++, referred to as human readable DPC++ 4004. In at least one embodiment, human readable DPC++ 4004 includes comments that are generated by DPC++ compatibility  
25 tool 4002 to indicate where user intervention may be necessary. In at least one embodiment, user intervention is necessary when CUDA source code 4000 calls a CUDA API that has no analogous DPC++ API; other examples where user intervention is required are discussed later in greater detail.

[0459] In at least one embodiment, a workflow for migrating CUDA source code 4000 (e.g.,  
30 application or portion thereof) includes creating one or more compilation database files; migrating CUDA to DPC++ using a DPC++ compatibility tool 4002; completing migration and verifying correctness, thereby generating DPC++ source code 4008; and compiling DPC++ source code 4008 with a DPC++ compiler to generate a DPC++ application. In at least one embodiment, a compatibility tool provides a utility that intercepts commands used  
35 when Makefile executes and stores them in a compilation database file. In at least one embodiment, a file is stored in JSON format. In at least one embodiment, an intercept-built command converts Makefile command to a DPC compatibility command.

5     **[0460]** In at least one embodiment, intercept-build is a utility script that intercepts a build process to capture compilation options, macro defs, and include paths, and writes this data to a compilation database file. In at least one embodiment, a compilation database file is a JSON file. In at least one embodiment, DPC++ compatibility tool 4002 parses a compilation database and applies options when migrating input sources. In at least one embodiment, use of intercept-build is optional, but highly recommended for Make or CMake based environments. In at least one embodiment, a migration database includes commands, directories, and files: command may include necessary compilation flags; directory may include paths to header files; file may include paths to CUDA files.

15     **[0461]** In at least one embodiment, DPC++ compatibility tool 4002 migrates CUDA code (e.g., applications) written in CUDA to DPC++ by generating DPC++ wherever possible. In at least one embodiment, DPC++ compatibility tool 4002 is available as part of a tool kit. In at least one embodiment, a DPC++ tool kit includes an intercept-build tool. In at least one embodiment, an intercept-built tool creates a compilation database that captures compilation commands to migrate CUDA files. In at least one embodiment, a compilation database generated by an intercept-built tool is used by DPC++ compatibility tool 4002 to migrate CUDA code to DPC++. In at least one embodiment, non-CUDA C++ code and files are migrated as is. In at least one embodiment, DPC++ compatibility tool 4002 generates human readable DPC++ 4004 which may be DPC++ code that, as generated by DPC++ compatibility tool 4002, cannot be compiled by DPC++ compiler and requires additional plumbing for verifying portions of code that were not migrated correctly, and may involve manual intervention, such as by a developer. In at least one embodiment, DPC++ compatibility tool 4002 provides hints or tools embedded in code to help developers manually migrate additional code that could not be migrated automatically. In at least one embodiment, migration is a one-time activity for a source file, project, or application.

30     **[0462]** In at least one embodiment, DPC++ compatibility tool 40002 is able to successfully migrate all portions of CUDA code to DPC++ and there may simply be an optional step for manually verifying and tuning performance of DPC++ source code that was generated. In at least one embodiment, DPC++ compatibility tool 4002 directly generates DPC++ source code 4008 which is compiled by a DPC++ compiler without requiring or utilizing human intervention to modify DPC++ code generated by DPC++ compatibility tool 4002. In at least one embodiment, DPC++ compatibility tool generates compile-able DPC++ code which can

5 be optionally tuned by a developer for performance, readability, maintainability, other various considerations; or any combination thereof.

[0463] In at least one embodiment, one or more CUDA source files are migrated to DPC++ source files at least partially using DPC++ compatibility tool 4002. In at least one embodiment, CUDA source code includes one or more header files which may include  
 10 CUDA header files. In at least one embodiment, a CUDA source file includes a <cuda.h> header file and a <stdio.h> header file which can be used to print text. In at least one embodiment, a portion of a vector addition kernel CUDA source file may be written as or related to:

```

15      #include <cuda.h>
      #include <stdio.h>
      #define VECTOR_SIZE 256

      [] global __ void VectorAddKernel(float* A, float* B, float* C)
      {
20          A[threadIdx.x] = threadIdx.x + 1.0f;
          B[threadIdx.x] = threadIdx.x + 1.0f;
          C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];
      }

25      int main()
      {
          float *d_A, *d_B, *d_C;

          cudaMalloc(&d_A, VECTOR_SIZE*sizeof(float));
          cudaMalloc(&d_B, VECTOR_SIZE*sizeof(float));
30          cudaMalloc(&d_C, VECTOR_SIZE*sizeof(float));

          VectorAddKernel<<<1, VECTOR_SIZE>>>>(d_A, d_B, d_C);

35          float Result[VECTOR_SIZE] = { };
          cudaMemcpy(Result, d_C, VECTOR_SIZE*sizeof(float),
          cudaMemcpyDeviceToHost);

          cudaFree(d_A);
          cudaFree(d_B);
40          cudaFree(d_C);

          for (int i=0; i<VECTOR_SIZE; i++ {
              if (i % 16 == 0) {
45                  printf("\n");
              }
              printf("%f ", Result[i]);
          }
  
```

5           return 0;  
          }

[0464] In at least one embodiment and in connection with CUDA source file presented above, DPC++ compatibility tool 4002 parses a CUDA source code and replaces header files  
10   with appropriate DPC++ and SYCL header files. In at least one embodiment, DPC++ header files includes helper declarations. In CUDA, there is a concept of a thread ID and correspondingly, in DPC++ or SYCL, for each element there is a local identifier.

[0465] In at least one embodiment and in connection with CUDA source file presented above, there are two vectors A and B which are initialized and a vector addition result is put  
15   into vector C as part of VectorAddKernel(). In at least one embodiment, DPC++ compatibility tool 4002 converts CUDA thread IDs used to index work elements to SYCL standard addressing for work elements via a local ID as part of migrating CUDA code to DPC++ code. In at least one embodiment, DPC++ code generated by DPC++ compatibility tool 4002 can be optimized – for example, by reducing dimensionality of an nd\_item, thereby  
20   increasing memory and/or processor utilization.

[0466] In at least one embodiment and in connection with CUDA source file presented above, memory allocation is migrated. In at least one embodiment, cudaMalloc() is migrated to a unified shared memory SYCL call malloc\_device() to which a device and context is passed, relying on SYCL concepts such as platform, device, context, and queue. In at least  
25   one embodiment, a SYCL platform can have multiple devices (e.g., host and GPU devices); a device may have multiple queues to which jobs can be submitted; each device may have a context; and a context may have multiple devices and manage shared memory objects.

[0467] In at least one embodiment and in connection with CUDA source file presented above, a main() function invokes or calls VectorAddKernel() to add two vectors A and B  
30   together and store result in vector C. In at least one embodiment, CUDA code to invoke VectorAddKernel() is replaced by DPC++ code to submit a kernel to a command queue for execution. In at least one embodiment, a command group handler cgh passes data, synchronization, and computation that is submitted to the queue, parallel\_for is called for a number of global elements and a number of work items in that work group where  
35   VectorAddKernel() is called.

[0468] In at least one embodiment and in connection with CUDA source file presented above, CUDA calls to copy device memory and then free memory for vectors A, B, and C are

5 migrated to corresponding DPC++ calls. In at least one embodiment, C++ code (e.g., standard ISO C++ code for printing a vector of floating point variables) is migrated as is, without being modified by DPC++ compatibility tool 4002. In at least one embodiment, DPC++ compatibility tool 4002 modify CUDA APIs for memory setup and/or host calls to execute kernel on the acceleration device. In at least one embodiment and in connection with  
 10 CUDA source file presented above, a corresponding human readable DPC++ 4004 (e.g., which can be compiled) is written as or related to:

```

#include <CL/sycl.hpp>
#include <dpct/dpct.hpp>
#define VECTOR_SIZE 256
15 void VectorAddKernel(float* A, float* B, float* C,
                                sycl::nd_item<3> item_ct1)
    {
        A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
20        B[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
        C[item_ct1.get_local_id(2)] =
            A[item_ct1.get_local_id(2)] + B[item_ct1.get_local_id(2)];
    }

25 int main()
    {
        float *d_A, *d_B, *d_C;

        d_A = (float *)sycl::malloc_device(VECTOR_SIZE * sizeof(float),
30                                dpct::get_current_device(),
                                dpct::get_default_context());
        d_B = (float *)sycl::malloc_device(VECTOR_SIZE * sizeof(float),
                                dpct::get_current_device(),
                                dpct::get_default_context());
35        d_C = (float *)sycl::malloc_device(VECTOR_SIZE * sizeof(float),
                                dpct::get_current_device(),
                                dpct::get_default_context());

        dpct::get_default_queue_wait().submit([&](sycl::handler &cgh) {
40            cgh.parallel_for(
                sycl::nd_range<3>(sycl::range<3>(1, 1, 1) *
                                sycl::range<3>(1, 1, VECTOR_SIZE) *
                                sycl::range<3>(1, 1, VECTOR_SIZE)),
                [=](sycl::nd_items<3> item_ct1) {
45                    VectorAddKernel(d_A, d_B, d_C, item_ct1);
                });
        });

        float Result[VECTOR_SIZE] = { };
50        dpct::get_default_queue_wait()

```

```

5      .memcpy(Result, d_C, VECTOR_SIZE * sizeof(float))
      .wait();

      sycl::free(d_A, dpct::get_default_context());
      sycl::free(d_B, dpct::get_default_context());
10     sycl::free(d_C, dpct::get_default_context());

      for (int i=0; i<VECTOR_SIZE; i++ {
          if (i % 16 == 0) {
              printf("\n");
15         }
          printf("%f ", Result[i]);
      }

      return 0;
20  }

```

[0469] In at least one embodiment, human readable DPC++ 4004 refers to output generated by DPC++ compatibility tool 4002 and may be optimized in one manner or another. In at least one embodiment, human readable DPC++ 4004 generated by DPC++ compatibility tool 4002 can be manually edited by a developer after migration to make it more maintainable, performance, or other considerations. In at least one embodiment, DPC++ code generated by DPC++ compatibility tool 40002 such as DPC++ disclosed can be optimized by removing repeat calls to `get_current_device()` and/or `get_default_context()` for each `malloc_device()` call. In at least one embodiment, DPC++ code generated above uses a 3 dimensional

30 `nd_range` which can be refactored to use only a single dimension, thereby reducing memory usage. In at least one embodiment, a developer can manually edit DPC++ code generated by DPC++ compatibility tool 4002 replace uses of unified shared memory with accessors. In at least one embodiment, DPC++ compatibility tool 4002 has an option to change how it migrates CUDA code to DPC++ code. In at least one embodiment, DPC++ compatibility tool

35 4002 is verbose because it is using a general template to migrate CUDA code to DPC++ code that works for a large number of cases.

[0470] In at least one embodiment, a CUDA to DPC++ migration workflow includes steps to: prepare for migration using intercept-build script; perform migration of CUDA projects to DPC++ using DPC++ compatibility tool 4002; review and edit migrated source files

40 manually for completion and correctness; and compile final DPC++ code to generate a DPC++ application. In at least one embodiment, manual review of DPC++ source code may be required in one or more scenarios including but not limited to: migrated API does not

5 return error code (CUDA code can return an error code which can then be consumed by the application but SYCL uses exceptions to report errors, and therefore does not use error codes to surface errors); CUDA compute capability dependent logic is not supported by DPC++; statement could not be removed. In at least one embodiment, scenarios in which DPC++ code requires manual intervention may include, without limitation: error code logic replaced with  
10 (\*,0) code or commented out; equivalent DPC++ API not available; CUDA compute capability-dependent logic; hardware-dependent API (clock()); missing features unsupported API; execution time measurement logic; handling built-in vector type conflicts; migration of cuBLAS API; and more.

[0471] At least one embodiment of the disclosure can be described in view of the following  
15 clauses:

Clause 1. A processor comprising: one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images.

20 Clause 2. The processor of clause 1, wherein the one or more circuits are to:

use a first processor resource to generate the first one or more images based at least in part on the three-dimensional representation;

provide the first one or more images and the three-dimensional representation  
25 to a second processor resource; and

use the second processor resource to train the one or more neural network to generate the one or more second versions of the one or more images using the first version of the one or more images and the three-dimensional representation.

Clause 3. The processor of clause 1 or 2, wherein the one or more circuits  
30 are to:

provide the one or more images and the three-dimensional representation to the second processor resource via a buffer, wherein:

the first processor resource is to write the first version of the one or more images and the three-dimensional representation to the buffer; and

35 the second processor resource is to read the first version of the one or more images and the three-dimensional representation from the buffer; and

5                    wherein the buffer is allocated in memory shared between the first processor resource and the second processor resource.

                  Clause 4.        The processor of any of clauses 1-3, wherein the second processor resource is to read the first version of the one or more images and three-dimensional representation from the buffer with a four-frame delay after the first processor  
10        resource writes the first version of the one or more images to the ring buffer.

                  Clause 5.        The processor of any of clauses 1-4, wherein the buffer is a ring buffer.

                  Clause 6.        The processor of any of clauses 1-5, wherein additional image data includes depth data, normal data, albedo data, roughness data, or motion vector data.

15                    Clause 7.        The processor of any of clauses 1-6, wherein the first processor resource comprises a first processor core and the second processor resource comprises a second processor core.

                  Clause 8.        A system comprising: one or more processors to use one or more neural networks to generate one or more second versions of one or more images based,  
20        at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images.

                  Clause 9.        The system of clause 8, wherein the one or more processors comprises a first processor resource to execute a software application comprising a plugin that provides the three-dimensional representation to a second processor resource that  
25        generates the first version of the one or more images and controls training of the one or more neural networks on a third processor resource.

                  Clause 10.       The system of clause 8 or 9, wherein the second processor resource is connected to a display device for presenting a version of the one or more images.

                  Clause 11.       The system of any of clauses 8-10, where the plugin controls  
30        whether the first version or the second version of the one or more images is to be presented on the display device.

                  Clause 12.       The system of any of clauses 8-11, wherein the plugin is to:

5                   determine a set of parameters from training the one or more neural networks;  
and

                  update a different one or more neural networks used by the first processor  
resource to generate the first version of the one or more images to use the set of parameters.

                  Clause 13.     The system of any of clauses 8-12, wherein:  
10                   the plugin is to receive training information from the third processor resource;  
and

                  the first processors resource provides the training information to the second  
processors resource to be presented using the display device.

                  Clause 14.     A machine-readable medium having stored thereon a set of  
15                   instructions, which if performed by one or more processor resources, cause the one or more  
processor resources to at least: use one or more neural networks to generate a second version  
of one or more images based, at least in part, on a first version of the one or more images and  
a three-dimensional representation of the one or more first versions of the one or more  
images.

20                   Clause 15.     The machine-readable medium of clause 14, wherein the set of  
instructions include instructions to:

                  use a first processor resource of the one or more processor resources to render  
the first version of the one or more images based at least in part on the three-dimensional  
representation and a second one or more neural networks;

25                   provide the first version of the one or more images and the three-dimensional  
representation to a second processor resource of the one or more processor resources; and

                  use the second processor resource to train the one or more neural network to  
generate the second version of the one or more images using the first version of the one or  
more images and the three-dimensional representation.

30                   Clause 16.     The machine-readable medium of clause 14 or 15, wherein the  
first processor resource to

                  render a noisy version of the one or more images using the three-dimensional  
representation; and

5                    use the second one or more neural networks to generate a denoised version of the one or more images from the noisy version of the one or more images, wherein the denoised version is the first version.

                    Clause 17.     The machine-readable medium of any of clauses 14-16,  
                    wherein the noisy version of the one or more images are to be rendered using a non-  
10     deterministic algorithm.

                    Clause 18.     The machine-readable medium of any of clauses 14-17,  
                    wherein the non-deterministic algorithm is a Monte Carlo path tracing algorithm.

                    Clause 19.     The machine-readable medium of any of clauses 14-18,  
                    wherein first processor resource is to render the first version of the one or more images using  
15     a first number of samples and the second processor resource is to render the second version of the one or more images using a second number of samples that is greater than the first number.

                    Clause 20.     The machine-readable medium of any of clauses 14-19,  
                    wherein the second version of the one or more images are used as ground truth data to train  
20     the one or more neural networks.

                    Clause 21.     The machine-readable medium of any of clauses 14-20,  
                    wherein the first processor resource is a graphics processing unit (GPU) and the second processor resource comprises a plurality of GPUs to collectively train the one or more neural networks.

25                    Clause 22.     A processor comprising: two or more processor resources having different computational capabilities to perform an inference operation using a first version of one or more neural networks and a first processor resource of the two or more processor resources and to train a second version of the one or more neural networks using a second processor resource of the two or more processor resources.

30                    Clause 23.     The processor of clause 22, wherein:  
                    the first processor resource is to:

                                render a first image based, at least in part, on a three-dimensional model;

5                   generate a second image from the first image using the first version of  
the one or more neural networks; and  
                    provide the second image and the three-dimensional model to the  
second processor resource; and  
                    the second processor resource is to use the second image and the three-  
10 dimensional model to train the second version of the one or more neural networks.

Clause 24.     The processor of clause 22 or 23, wherein the first processor  
resource is to push at least the second image to end of a queue and the second processor  
resource is to pop at least the second image from front of the queue.

Clause 25.     The processor of any of clauses 22-24, wherein the first  
15 processor resource is to push images to the queue at a rate of 60 images per second or more.

Clause 26.     The processor of any of clauses 22-25, wherein the first  
processor resource pushes additional data associated with the second image to the queue that  
includes depth data, normal data, albedo data, roughness data, or motion vector data.

Clause 27.     The processor of any of clauses 22-26, wherein the three-  
20 dimensional model is to be used by the second processor resource to generate a ground truth  
image that is to be compared against the second image as part of training the second version  
of the one or more neural networks.

Clause 28.     The processor of any of clauses 22-27, wherein the one or more  
neural networks is a denoiser neural network, further wherein the first version and the second  
25 version of the one or more neural networks have differing weights.

Clause 29.     A system comprising: two or more processor resources having  
different computational capabilities to perform an inference operation using a first version of  
one or more neural networks and a first processor resource of the two or more processor  
resources and to train a second version of the one or more neural networks using a second  
30 processor resource of the two or more processor resources.

Clause 30.     The system of clause 29, further comprising a third processor  
resource of the two or more processor resources to execute a software application comprising  
a plugin that provides a three-dimensional representation to the first processor resource to

5 generate one or more images using the first version of the one or more neural networks and controls training of the second version of the one or more neural networks on the second processor resource.

Clause 31. The system of clause 29 or 30, wherein the first processor resource is connected to a display device for presenting images generated by the first  
10 processor resource.

Clause 32. The system of any of clauses 29-31, where the first processor resource is to generate a first version of an image using the three-dimensional representation and generate a second version of the image using the first version of the one or more neural networks.

15 Clause 33. The system of any of clauses 29-32, wherein the plugin controls whether the first version or the second version of the image is to be presented on the display device.

Clause 34. The system of any of clauses 29-33, wherein the plugin is to:  
obtain a set of parameters from training the second version of the one or more  
20 neural networks; and  
update the first version of the one or more neural networks to use the set of parameters.

Clause 35. The system of any of clauses 29-34, wherein the plugin is to provide the set of parameters to the first processor resource to render on a display device.

25 Clause 36. The system of any of clauses 29-35, wherein the plugin is an optional component of the software application.

Clause 37. The system of any of clauses 29-26, wherein the first processor resource comprises a first graphics processing unit (GPU) and the second processor resource comprises a second GPU.

30 Clause 38. A machine-readable medium having stored thereon a set of instructions, which if performed by two or more processor resources having different computational capabilities, cause the two or more processor resources to at least: perform an inference operation using a first version of one or more neural networks and a first processor

5 resource of the two or more processor resources and to train a second version of the one or more neural networks using a second processor resource of the two or more processor resources.

Clause 39. The machine-readable medium of clause 38, wherein the set of instructions include instructions to:

10 use the inference operation to generate a first version of one or more images based at least in part on a three-dimensional representation;  
provide the first version of the one or more images and the three-dimensional representation to the second processor resource; and  
train the second version of the one or more neural network to generate a  
15 second version of the one or more images using the first version of the one or more images and the three-dimensional representation.

Clause 40. The machine-readable medium of clause 38 or 39, wherein the first processor resource is to:

20 render a noisy version of the one or more images using the three-dimensional representation; and  
use the first version of the one or more neural networks to generate a denoised version of the one or more images from the noisy version of the one or more images, wherein the denoised version of the one or more images is the first version of the one or more images.

Clause 41. The machine-readable medium of any of clauses 38-40,  
25 wherein the noisy version of the one or more images are to be rendered using a non-deterministic algorithm.

Clause 42. The machine-readable medium of any of clauses 38-41,  
wherein the non-deterministic algorithm is a Monte Carlo path tracing algorithm.

30 **[0472]** 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,

5 alternative constructions, and equivalents falling within spirit and scope of disclosure, as defined in appended claims.

[0473] Use of terms “a” and “an” and “the” and similar referents in context of describing disclosed embodiments (especially in context of following claims) are to be construed to cover both singular and plural, unless otherwise indicated herein or clearly contradicted by context, and not as a definition of a term. Terms “comprising,” “having,” “including,” and “containing” are to be construed as open-ended terms (meaning “including, but not limited to,”) unless otherwise noted. 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 individually to each separate value falling within range, unless otherwise indicated herein and each separate value is incorporated into specification as if it were individually recited herein. 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.

[0474] Conjunctive language, such as phrases of form “at least one of A, B, and C,” or “at least one of A, B and C,” unless specifically stated otherwise or otherwise clearly contradicted by context, is otherwise understood with context as used in general to present that an item, term, etc., may be either A or B or C, or any nonempty subset of set of A and B and C. For instance, in illustrative example of a set having three members, conjunctive phrases “at least one of A, B, and C” and “at least one of A, B and C” refer to any of following sets: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}. Thus, such conjunctive language is not generally intended to imply that certain embodiments require at least one of A, at least one of B and at least one of C each to be present. In addition, unless otherwise noted or contradicted by context, term “plurality” indicates a state of being plural (e.g., “a plurality of items” indicates multiple items). 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.”

5 [0475] Operations of processes described herein can be performed in any suitable order unless otherwise indicated herein or otherwise clearly contradicted by context. In at least one embodiment, a process such as those processes described herein (or variations and/or combinations thereof) is performed under control of one or more computer systems configured with executable instructions and is implemented as code (e.g., executable instructions, one or more computer programs or one or more applications) executing collectively on one or more processors, by hardware or combinations thereof. In at least one embodiment, code is stored on a computer-readable storage medium, for example, in form of a computer program comprising a plurality of instructions executable by one or more processors. In at least one embodiment, a computer-readable storage medium is a non-transitory computer-readable storage medium that excludes transitory signals (e.g., a propagating transient electric or electromagnetic transmission) but includes non-transitory data storage circuitry (e.g., buffers, cache, and queues) within transceivers of transitory signals. In at least one embodiment, code (e.g., executable code or source code) is stored on a set of one or more non-transitory computer-readable storage media having stored thereon executable instructions (or other memory to store executable instructions) that, when executed (i.e., as a result of being executed) by one or more processors of a computer system, cause computer system to perform operations described herein. 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.

[0476] 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,

5 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.

[0477] 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  
10 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.

[0478] All references, including publications, patent applications, and patents, cited herein are hereby incorporated by reference to same extent as if each reference were individually  
15 and specifically indicated to be incorporated by reference and were set forth in its entirety herein.

[0479] 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  
20 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.

[0480] Unless specifically stated otherwise, it may be appreciated that throughout specification terms such as “processing,” “computing,” “calculating,” “determining,” or like,  
25 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.

[0481] 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  
30 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

5 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.

[0482] 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  
10 machine. 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  
15 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  
20 input or output parameter of a function call, a parameter of an application programming interface or interprocess communication mechanism.

[0483] 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  
25 responsibilities are defined above for purposes of discussion, various functions and responsibilities might be distributed and divided in different ways, depending on circumstances.

[0484] 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  
30 claimed in appended claims is not necessarily limited to specific features or acts described. Rather, specific features and acts are disclosed as exemplary forms of implementing the claims.

PATENT

Attorney Docket No. 0112912-147WO0

Client Reference No. 20-SC-0226WO01

CLAIMS

## WHAT IS CLAIMED IS:

1. A processor comprising: one or more circuits to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images.
2. The processor of claim 1, wherein the one or more circuits are to:
  - use a first processor resource to generate the first one or more images based at least in part on the three-dimensional representation;
  - provide the first one or more images and the three-dimensional representation to a second processor resource; and
  - use the second processor resource to train the one or more neural network to generate the one or more second versions of the one or more images using the first version of the one or more images and the three-dimensional representation.
3. The processor of claim 2, wherein the one or more circuits are to:
  - provide the one or more images and the three-dimensional representation to the second processor resource via a buffer, wherein:
    - the first processor resource is to write the first version of the one or more images and the three-dimensional representation to the buffer; and
    - the second processor resource is to read the first version of the one or more images and the three-dimensional representation from the buffer; and
    - wherein the buffer is allocated in memory shared between the first processor resource and the second processor resource.
4. The processor of claim 3, wherein the second processor resource is to read the first version of the one or more images and three-dimensional representation from the buffer with a four-frame delay after the first processor resource writes the first version of the one or more images to the buffer.

5. The processor of claim 3, wherein the buffer is a ring buffer.
6. The processor of claim 5, wherein additional image data includes depth data, normal data, albedo data, roughness data, or motion vector data.
7. The processor of claim 2, wherein the first processor resource comprises a first processor core and the second processor resource comprises a second processor core.
8. A system comprising: one or more processors to use one or more neural networks to generate one or more second versions of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images.
9. The system of claim 8, wherein the one or more processors comprises a first processor resource to execute a software application comprising a plugin that provides the three-dimensional representation to a second processor resource that generates the first version of the one or more images and controls training of the one or more neural networks on a third processor resource.
10. The system of claim 9, wherein the second processor resource is connected to a display device for presenting a version of the one or more images.
11. The system of claim 10, where the plugin controls whether the first version or the second version of the one or more images is to be presented on the display device.
12. The system of claim 11, wherein the plugin is to:  
determine a set of parameters from training the one or more neural networks;  
and  
update a different one or more neural networks used by the first processor resource to generate the first version of the one or more images to use the set of parameters.
13. The system of claim 10, wherein:  
the plugin is to receive training information from the third processor resource;  
and  
the first processors resource provides the training information to the second processors resource to be presented using the display device.

14. A machine-readable medium having stored thereon a set of instructions, which if performed by one or more processor resources, cause the one or more processor resources to at least: use one or more neural networks to generate a second version of one or more images based, at least in part, on a first version of the one or more images and a three-dimensional representation of the one or more first versions of the one or more images.

15. The machine-readable medium of claim 14, wherein the set of instructions include instructions to:

use a first processor resource of the one or more processor resources to render the first version of the one or more images based at least in part on the three-dimensional representation and a second one or more neural networks;

provide the first version of the one or more images and the three-dimensional representation to a second processor resource of the one or more processor resources; and

use the second processor resource to train the one or more neural network to generate the second version of the one or more images using the first version of the one or more images and the three-dimensional representation.

16. The machine-readable medium of claim 15, wherein the first processor resource to

render a noisy version of the one or more images using the three-dimensional representation; and

use the second one or more neural networks to generate a denoised version of the one or more images from the noisy version of the one or more images, wherein the denoised version is the first version.

17. The machine-readable medium of claim 16, wherein the noisy version of the one or more images are to be rendered using a non-deterministic algorithm.

18. The machine-readable medium of claim 17, wherein the non-deterministic algorithm is a Monte Carlo path tracing algorithm.

19. The machine-readable medium of claim 15, wherein first processor resource is to render the first version of the one or more images using a first number of samples and the second processor resource is to render the second version of the one or more images using a second number of samples that is greater than the first number.

20. The machine-readable medium of claim 15, wherein the second version of the one or more images are used as ground truth data to train the one or more neural networks.

21. The machine-readable medium of claim 15, wherein the first processor resource is a graphics processing unit (GPU) and the second processor resource comprises a plurality of GPUs to collectively train the one or more neural networks.

22. A processor comprising: two or more processor resources having different computational capabilities to perform an inference operation using a first version of one or more neural networks and a first processor resource of the two or more processor resources and to train a second version of the one or more neural networks using a second processor resource of the two or more processor resources.

23. The processor of claim 22, wherein:

the first processor resource is to:

render a first image based, at least in part, on a three-dimensional model;

generate a second image from the first image using the first version of the one or more neural networks; and

provide the second image and the three-dimensional model to the second processor resource; and

the second processor resource is to use the second image and the three-dimensional model to train the second version of the one or more neural networks.

24. The processor of claim 23, wherein the first processor resource is to push at least the second image to end of a queue and the second processor resource is to pop at least the second image from front of the queue.

25. The processor of claim 24, wherein the first processor resource is to push images to the queue at a rate of 60 images per second or more.

26. The processor of claim 24, wherein the first processor resource pushes additional data associated with the second image to the queue that includes depth data, normal data, albedo data, roughness data, or motion vector data.

27. The processor of claim 23, wherein the three-dimensional model is to be used by the second processor resource to generate a ground truth image that is to be compared against the second image as part of training the second version of the one or more neural networks.

28. The processor of claim 22, wherein the one or more neural networks is a denoiser neural network, further wherein the first version and the second version of the one or more neural networks have differing weights.

29. A system comprising: two or more processor resources having different computational capabilities to perform an inference operation using a first version of one or more neural networks and a first processor resource of the two or more processor resources and to train a second version of the one or more neural networks using a second processor resource of the two or more processor resources.

30. The system of claim 30, further comprising a third processor resource of the two or more processor resources to execute a software application comprising a plugin that provides a three-dimensional representation to the first processor resource to generate one or more images using the first version of the one or more neural networks and controls training of the second version of the one or more neural networks on the second processor resource.

31. The system of claim 31, wherein the first processor resource is connected to a display device for presenting images generated by the first processor resource.

32. The system of claim 32, where the first processor resource is to generate a first version of an image using the three-dimensional representation and generate a second version of the image using the first version of the one or more neural networks.

33. The system of claim 33, wherein the plugin controls whether the first version or the second version of the image is to be presented on the display device.

34. The system of claim 31, wherein the plugin is to:  
obtain a set of parameters from training the second version of the one or more neural networks; and

update the first version of the one or more neural networks to use the set of parameters.

35. The system of claim 35, wherein the plugin is to provide the set of parameters to the first processor resource to render on a display device.

36. The system of claim 31, wherein the plugin is an optional component of the software application.

37. The system of claim 29, wherein the first processor resource comprises a first graphics processing unit (GPU) and the second processor resource comprises a second GPU.

38. A machine-readable medium having stored thereon a set of instructions, which if performed by two or more processor resources having different computational capabilities, cause the two or more processor resources to at least: perform an inference operation using a first version of one or more neural networks and a first processor resource of the two or more processor resources and to train a second version of the one or more neural networks using a second processor resource of the two or more processor resources.

39. The machine-readable medium of claim 38, wherein the set of instructions include instructions to:

use the inference operation to generate a first version of one or more images based at least in part on a three-dimensional representation;

provide the first version of the one or more images and the three-dimensional representation to the second processor resource; and

train the second version of the one or more neural network to generate a second version of the one or more images using the first version of the one or more images and the three-dimensional representation.

40. The machine-readable medium of claim 39, wherein the first processor resource is to:

render a noisy version of the one or more images using the three-dimensional representation; and

use the first version of the one or more neural networks to generate a denoised version of the one or more images from the noisy version of the one or more images, wherein the denoised version of the one or more images is the first version of the one or more images.

41. The machine-readable medium of claim 40, wherein the noisy version of the one or more images are to be rendered using a non-deterministic algorithm.

42. The machine-readable medium of claim 41, wherein the non-deterministic algorithm is a Monte Carlo path tracing algorithm.

100

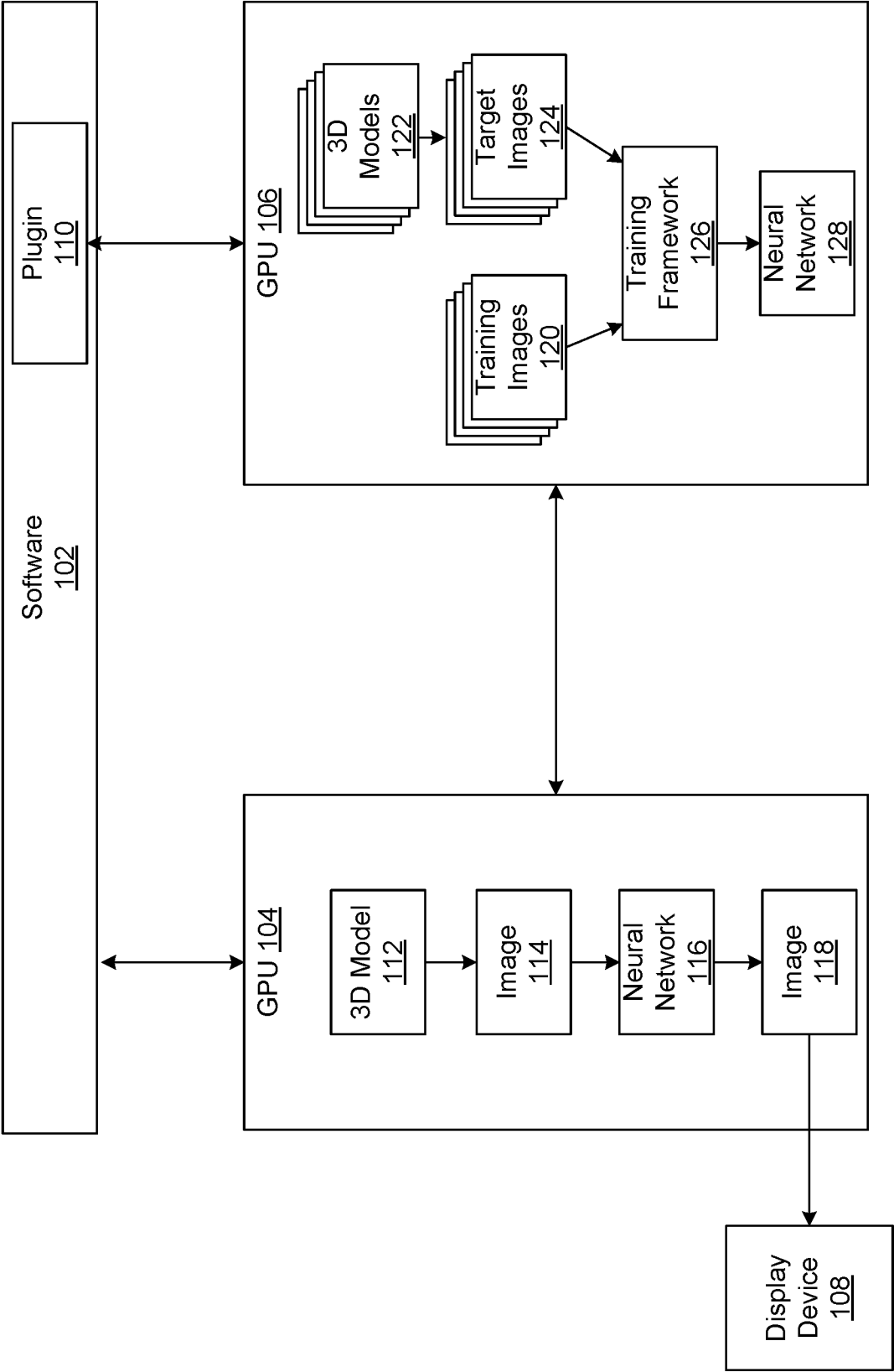


FIG. 1

200

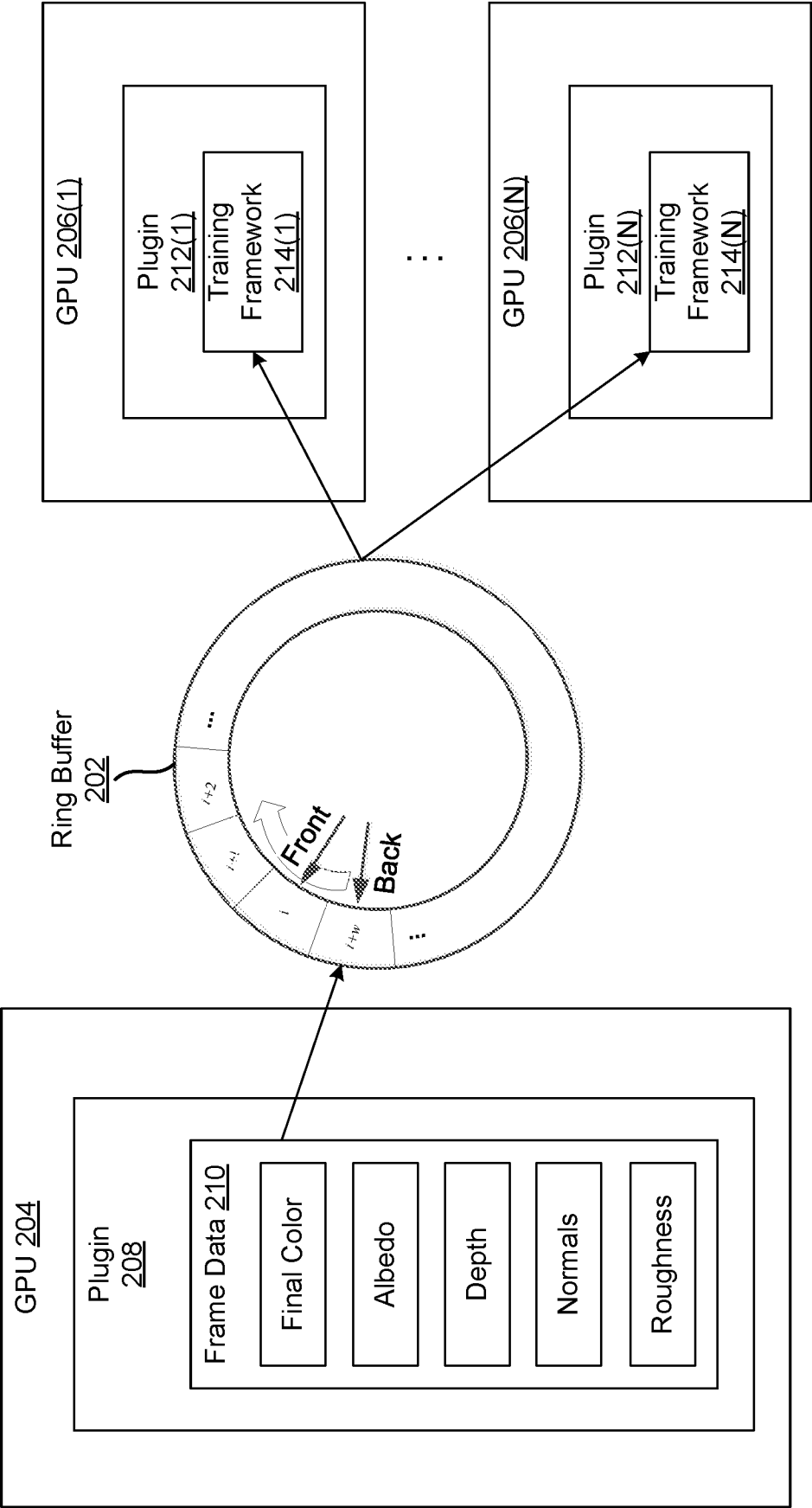


FIG. 2

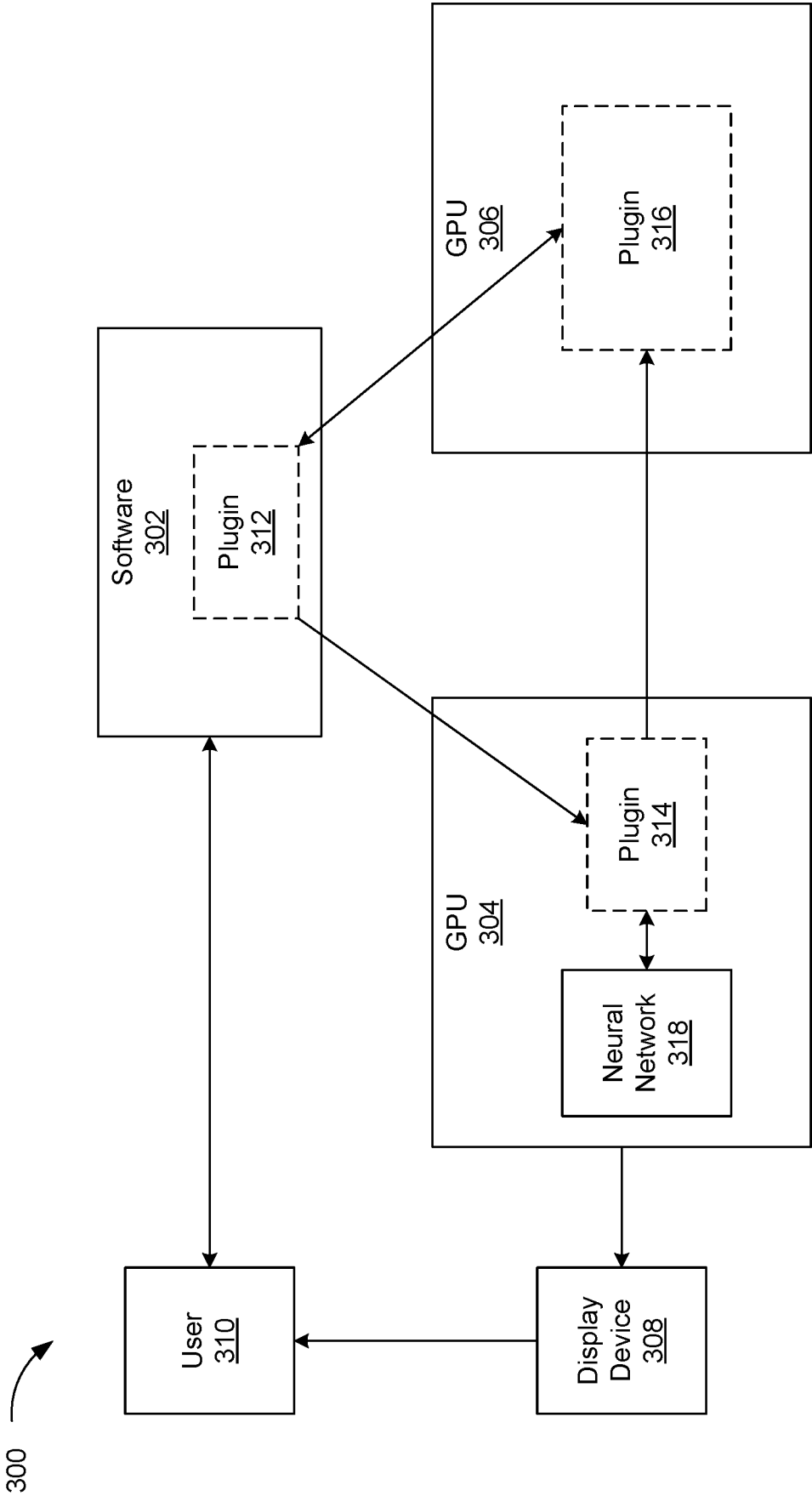


FIG. 3

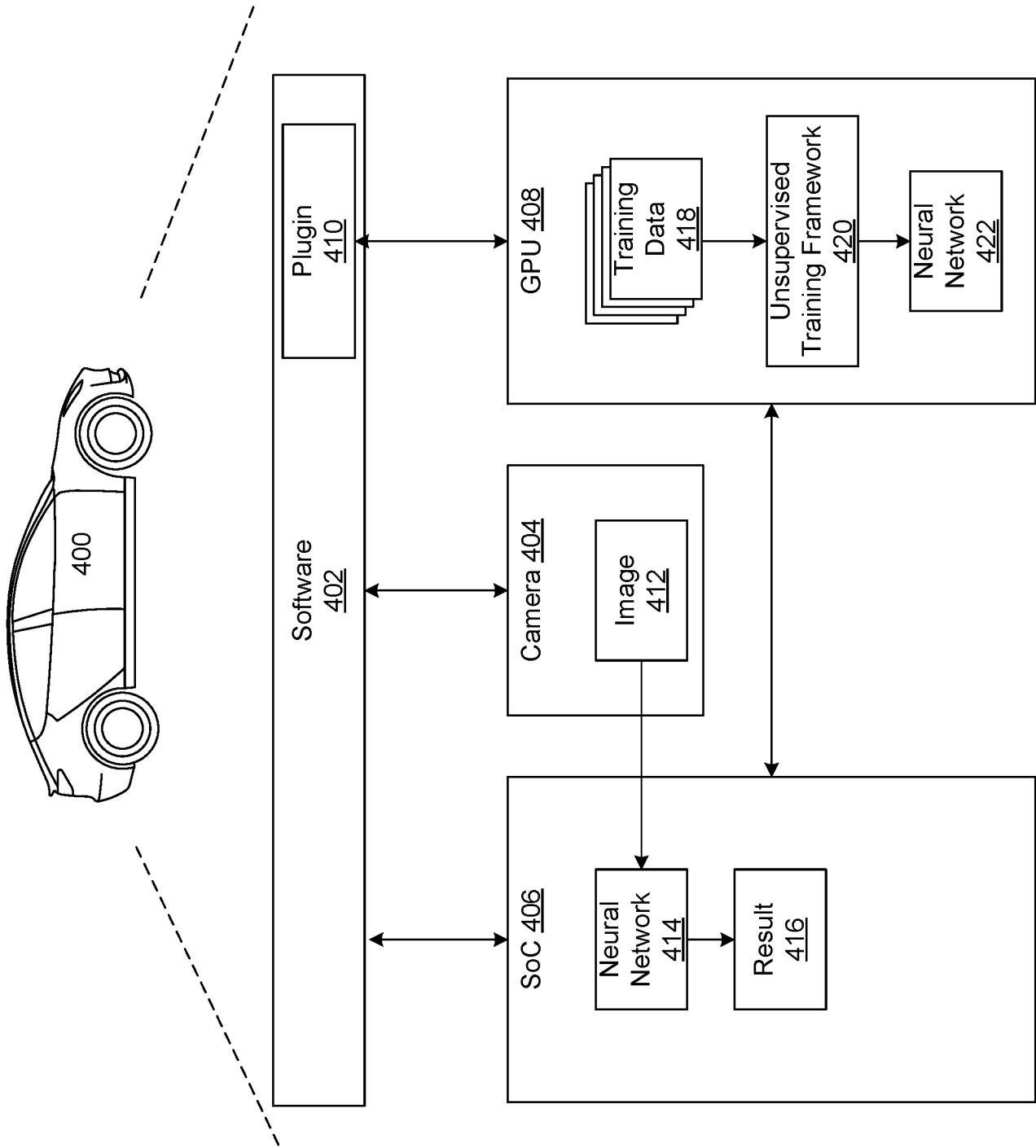


FIG. 4

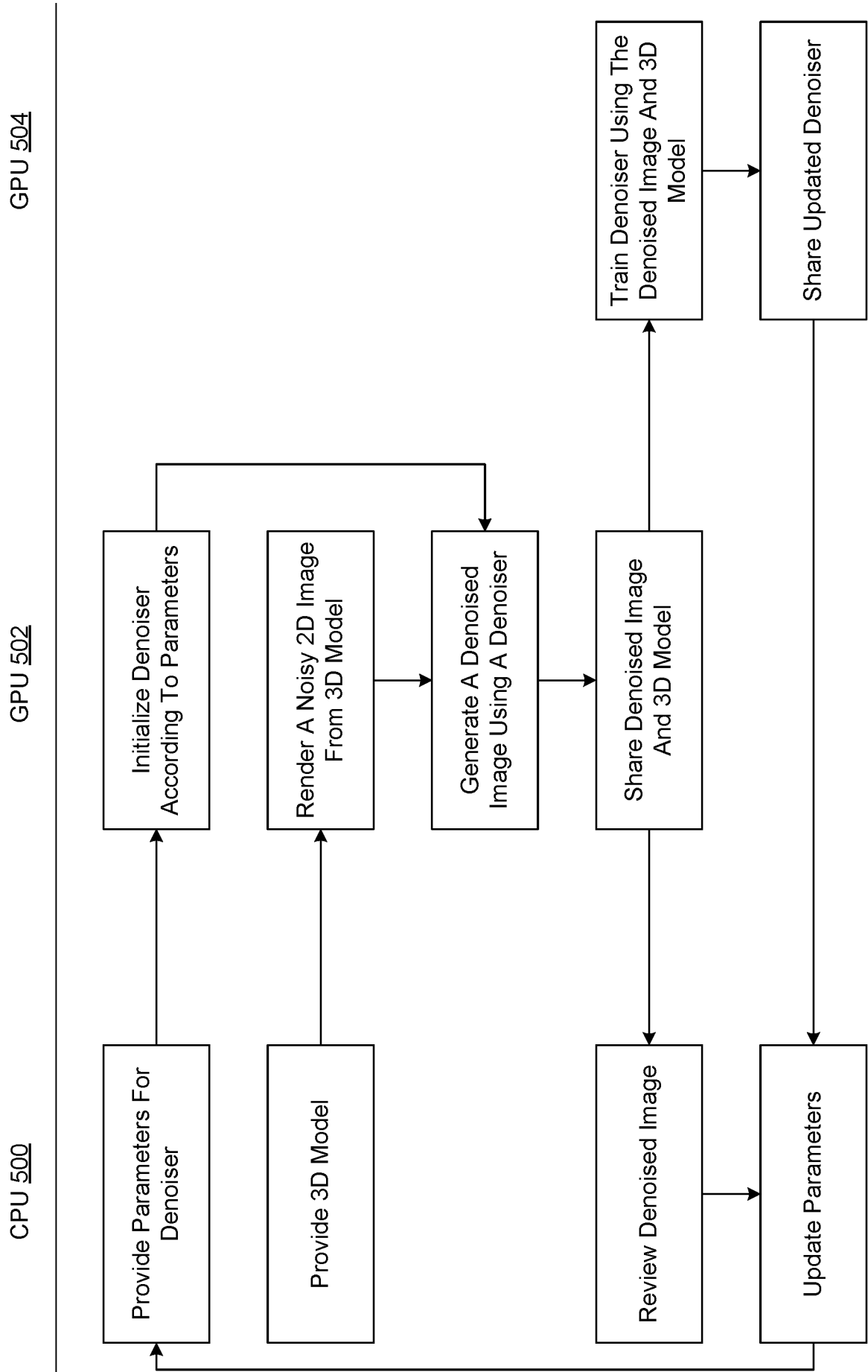
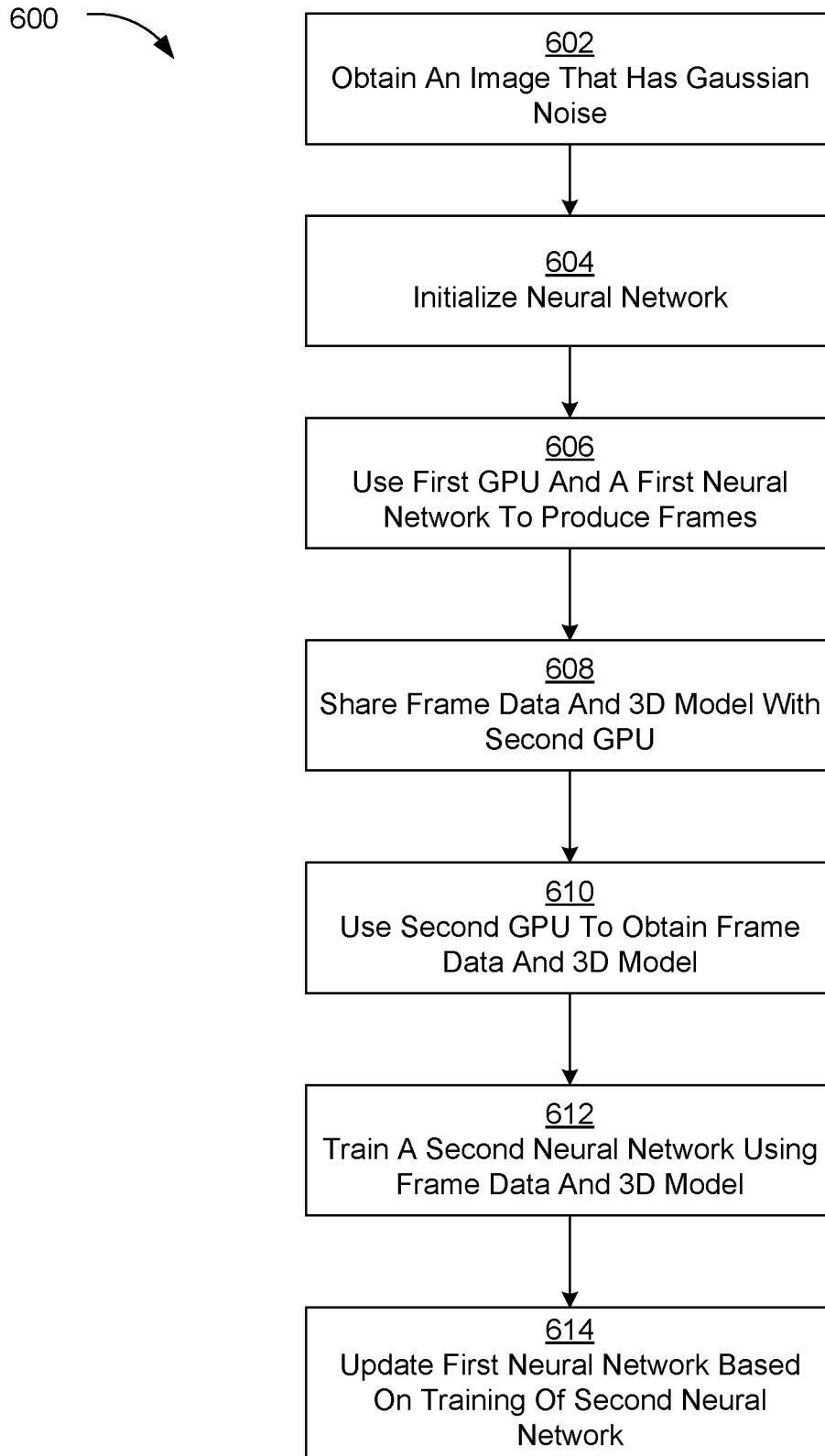


FIG. 5

**FIG. 6**

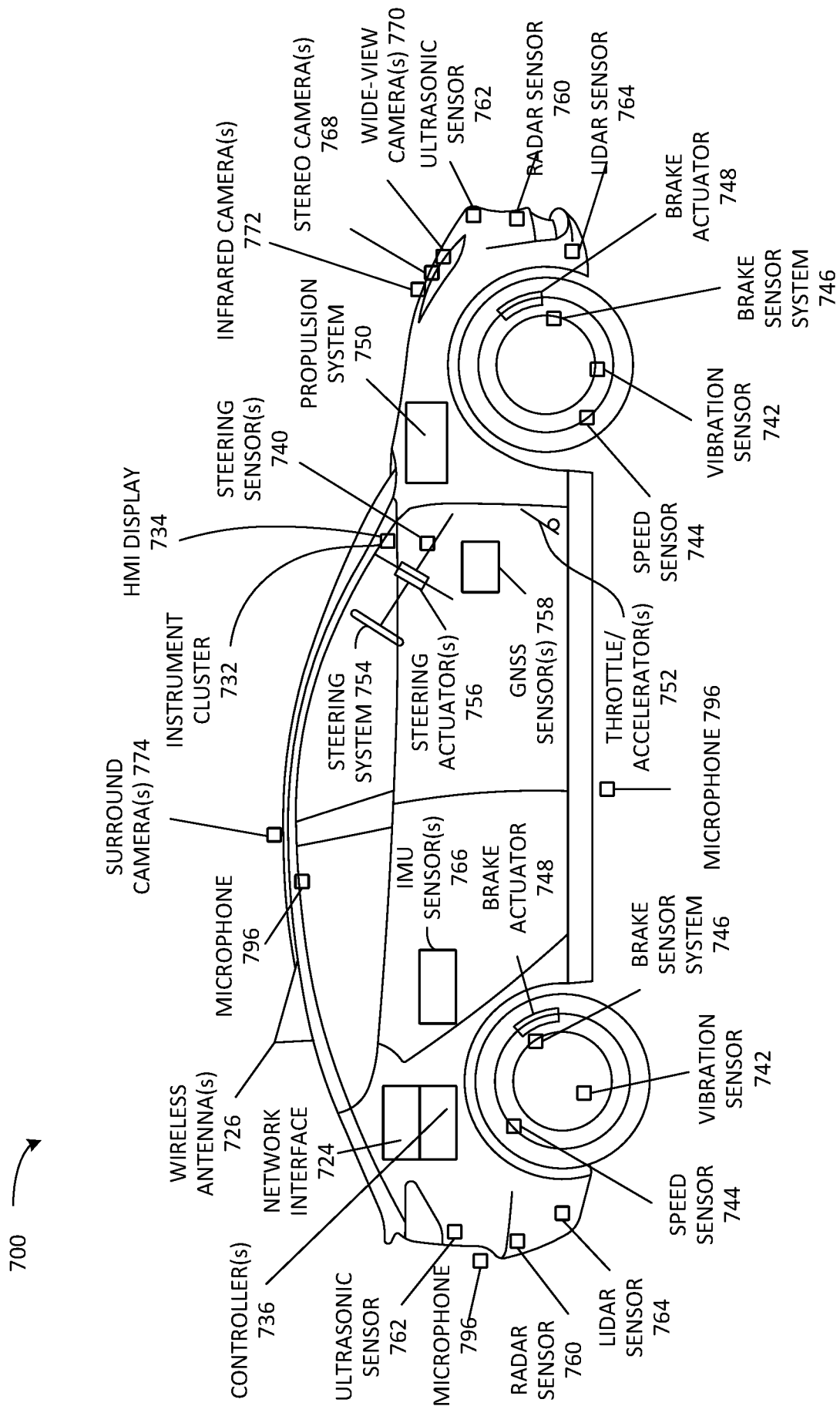


**FIG. 7A**

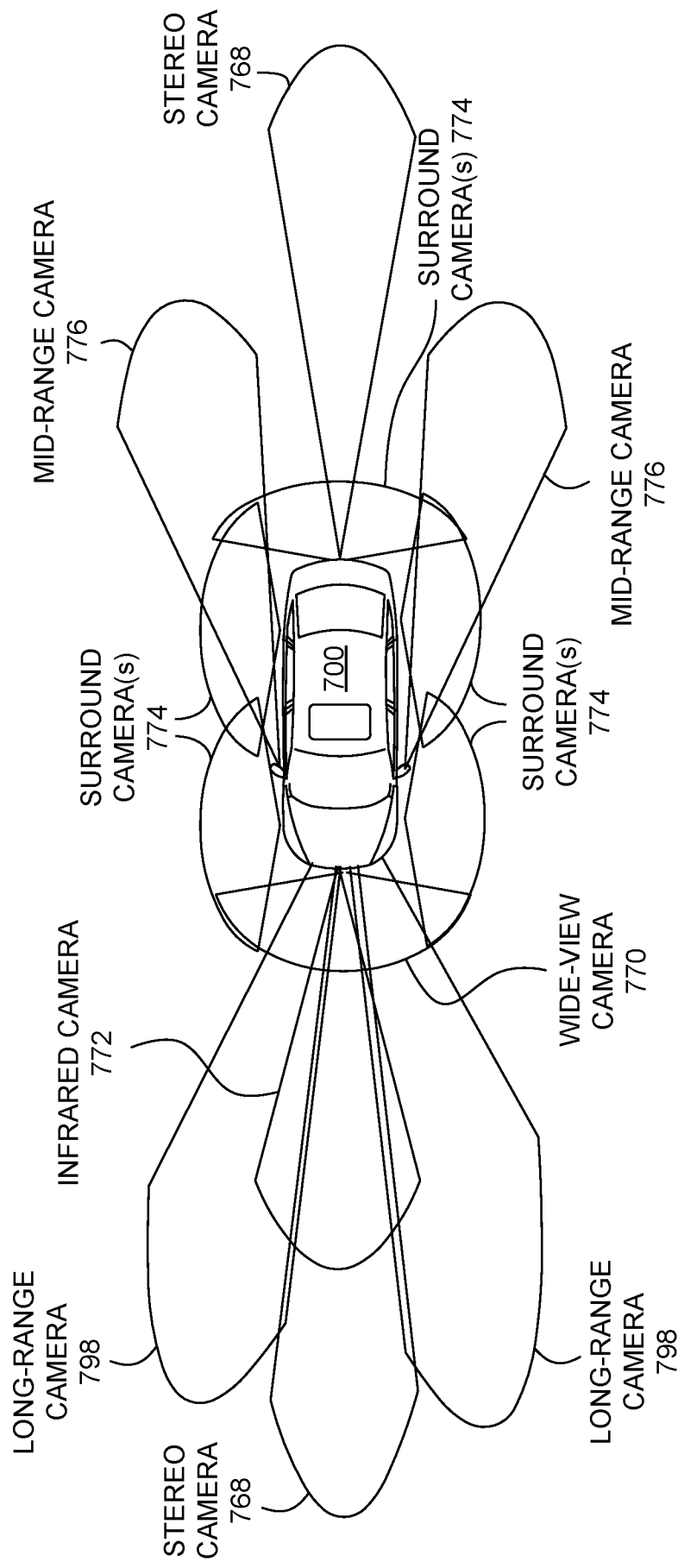


FIG. 7B

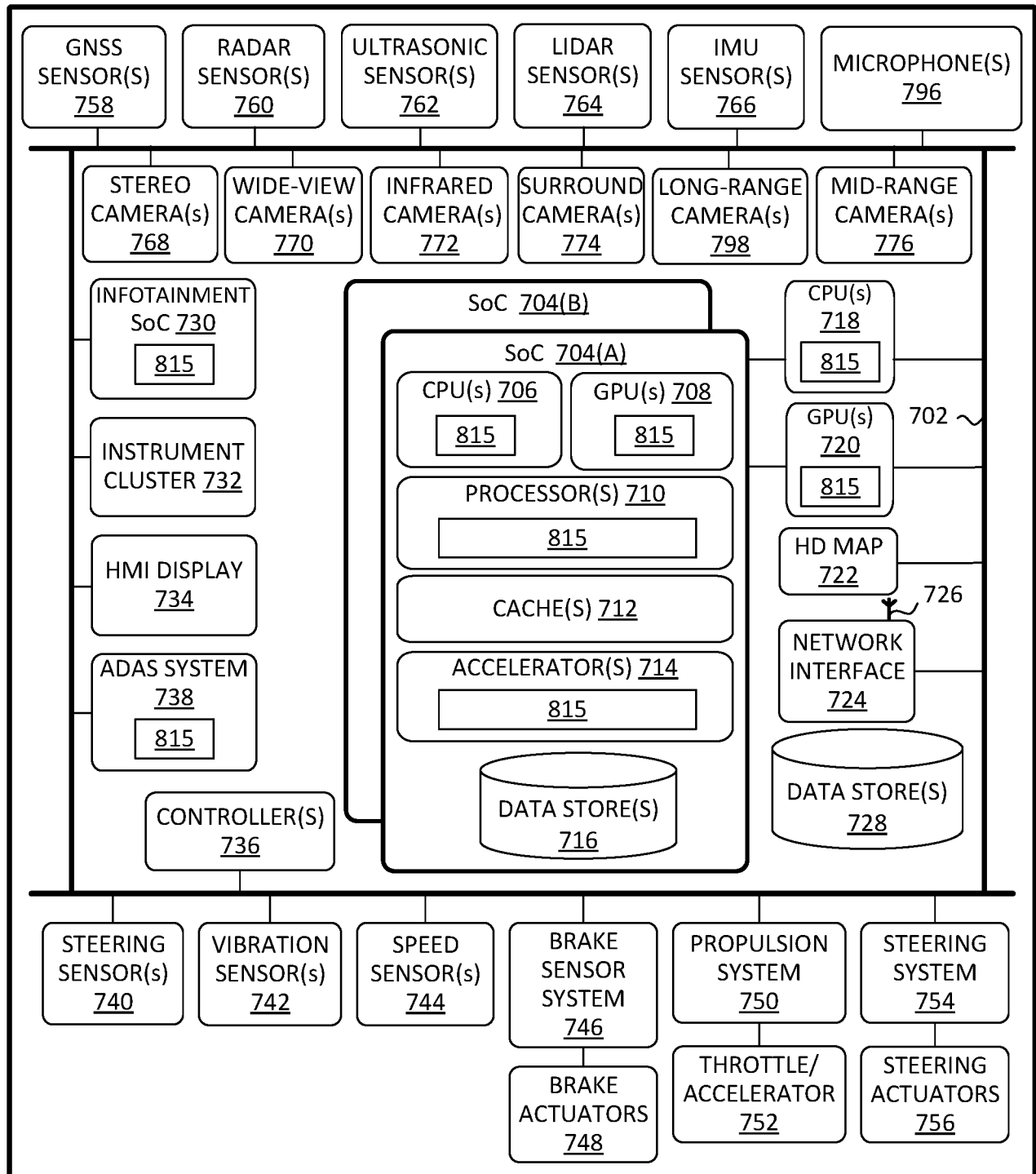
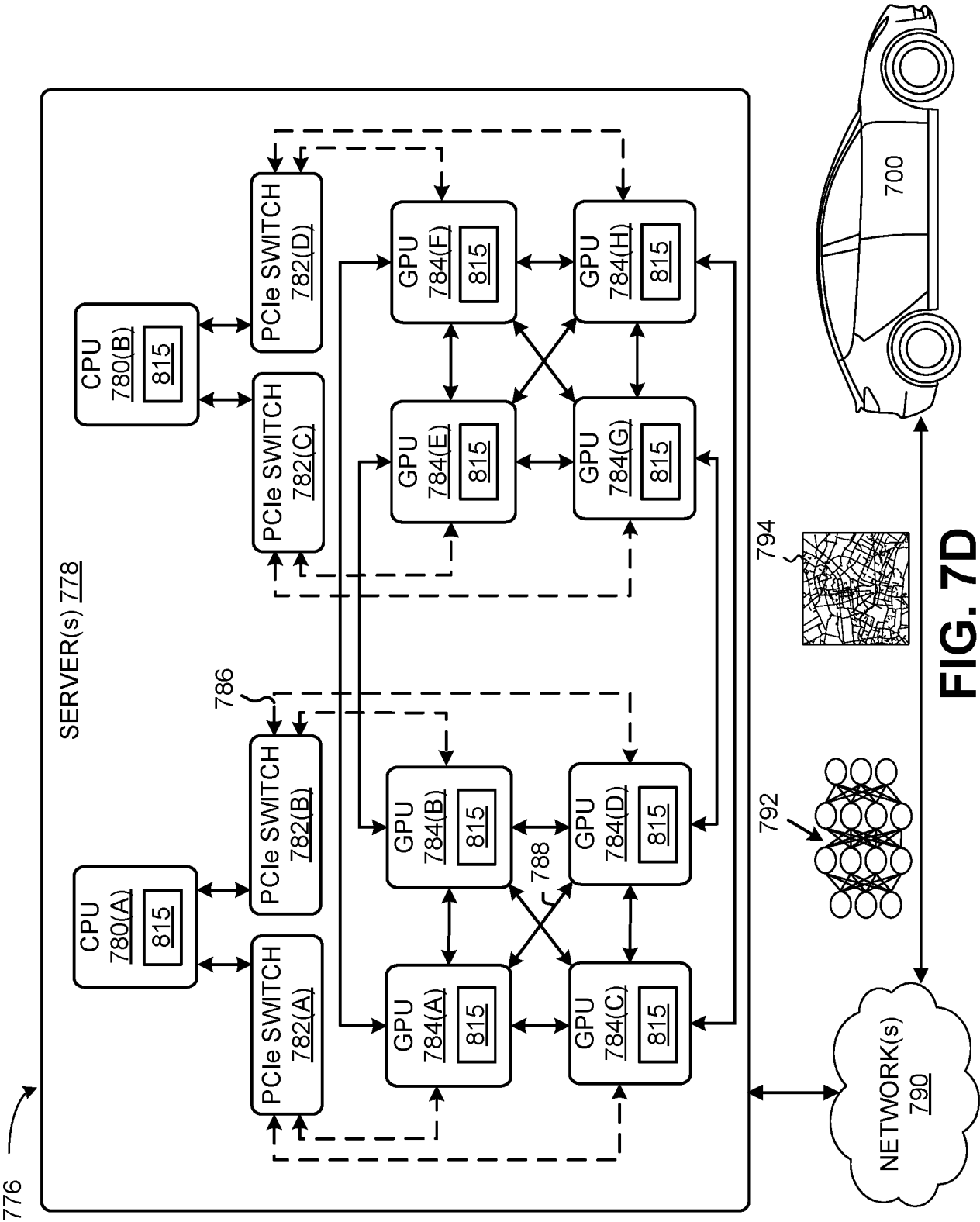
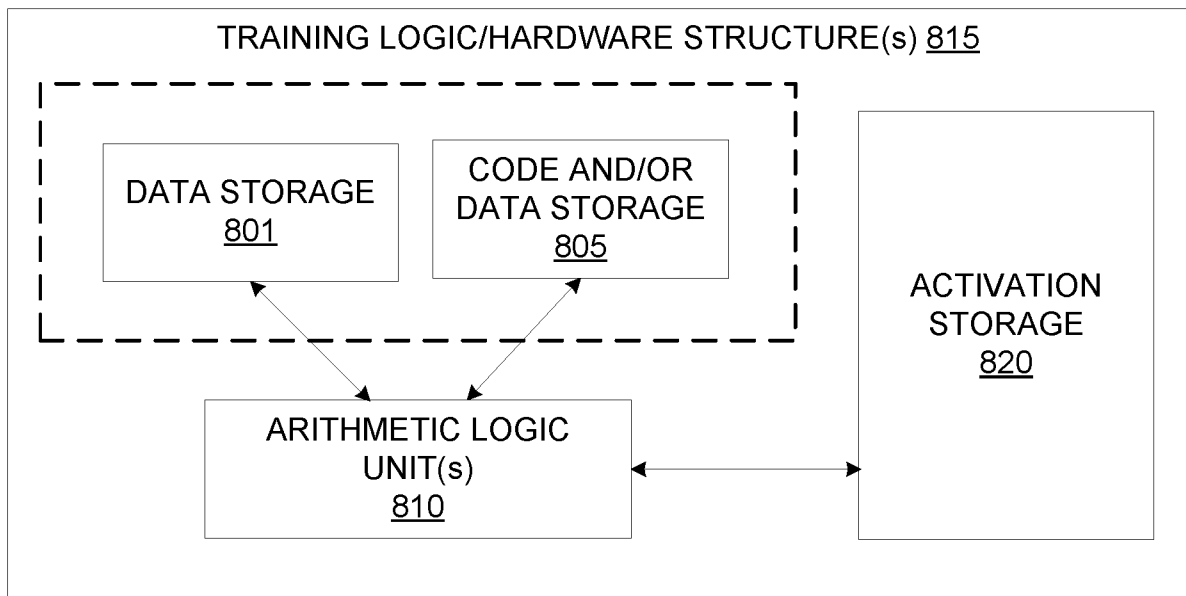
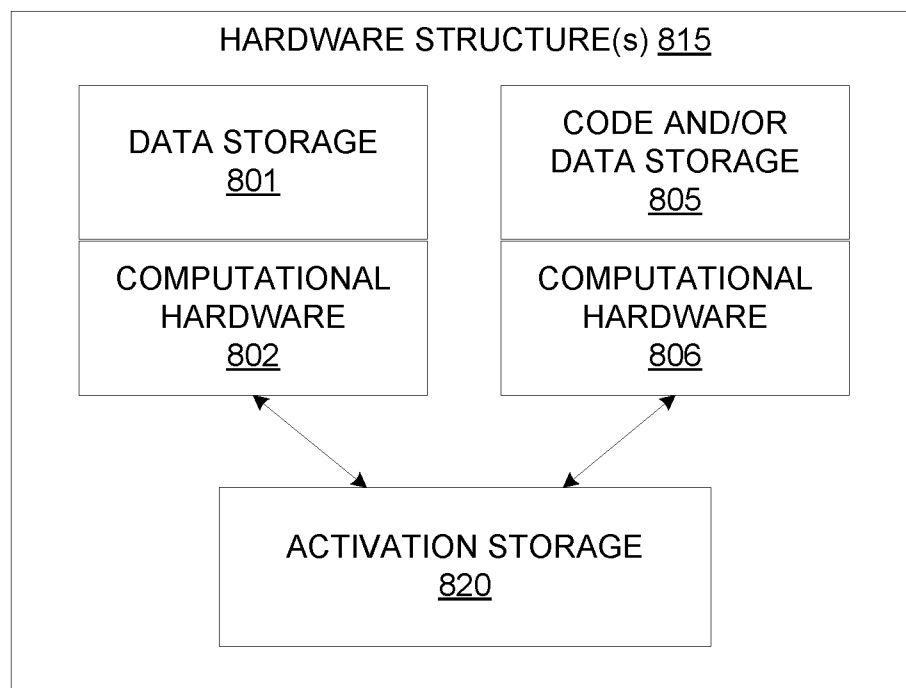


FIG. 7C



**FIG. 8A****FIG. 8B**

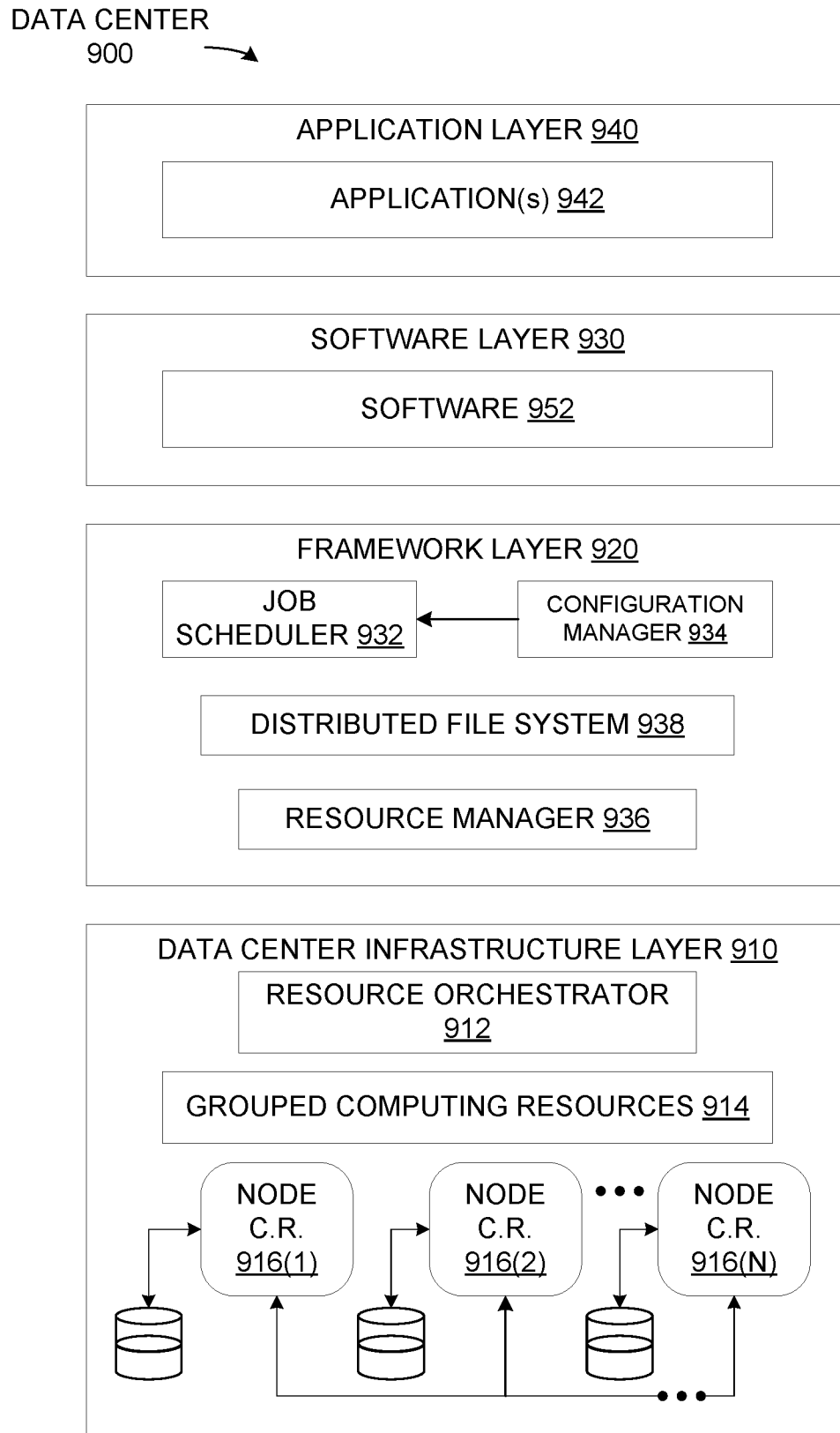


FIG. 9

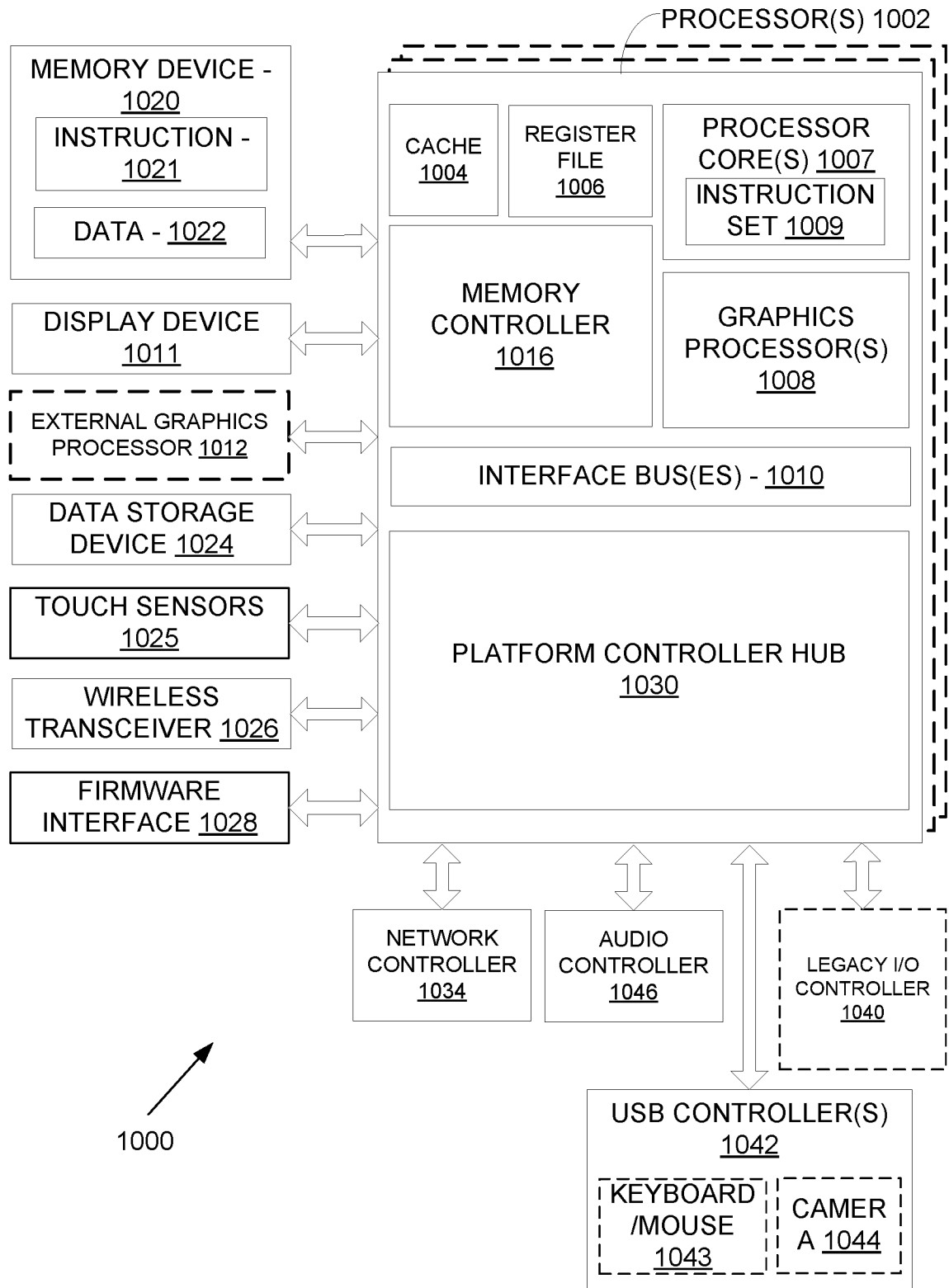
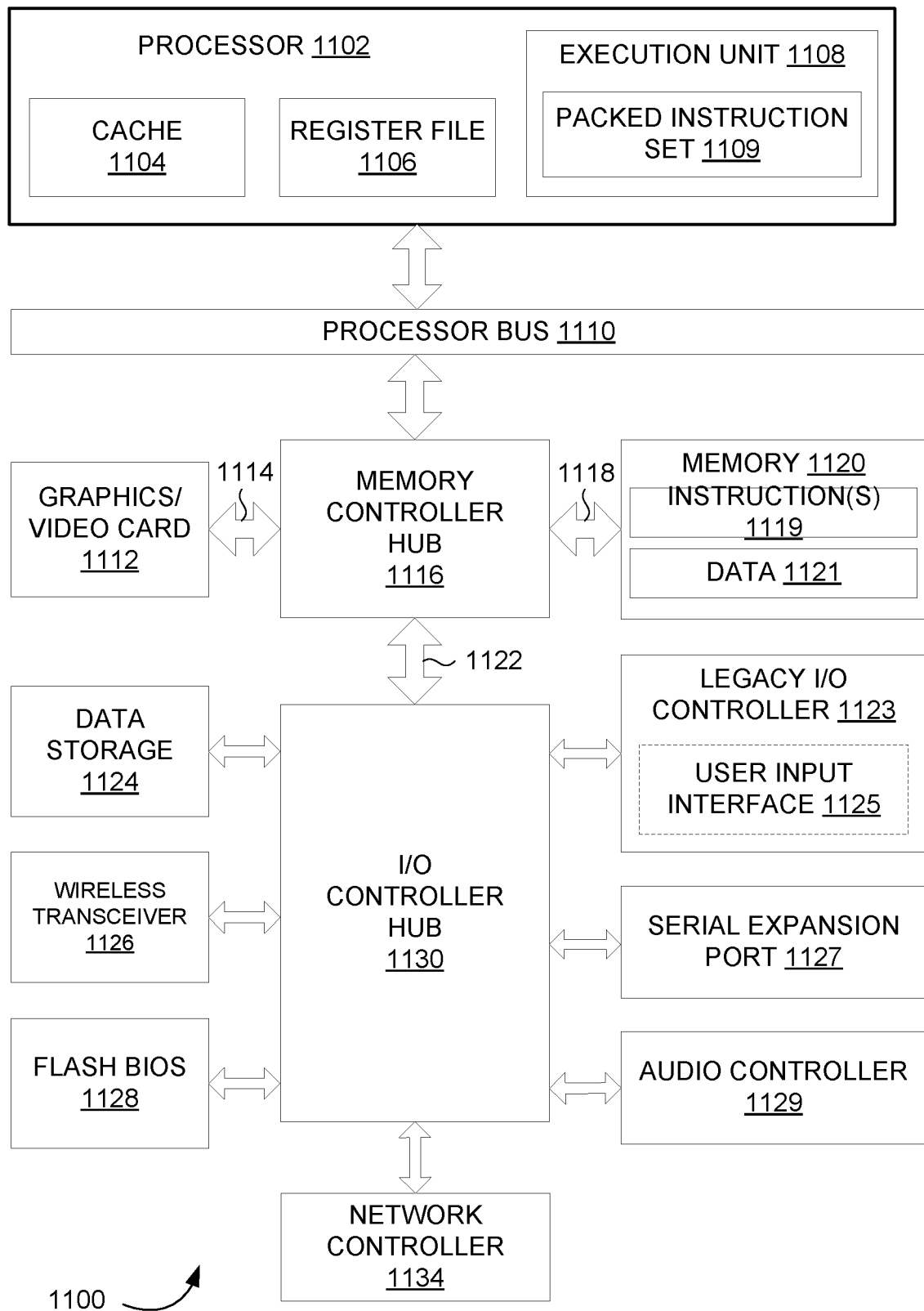


FIG. 10

**FIG. 11**

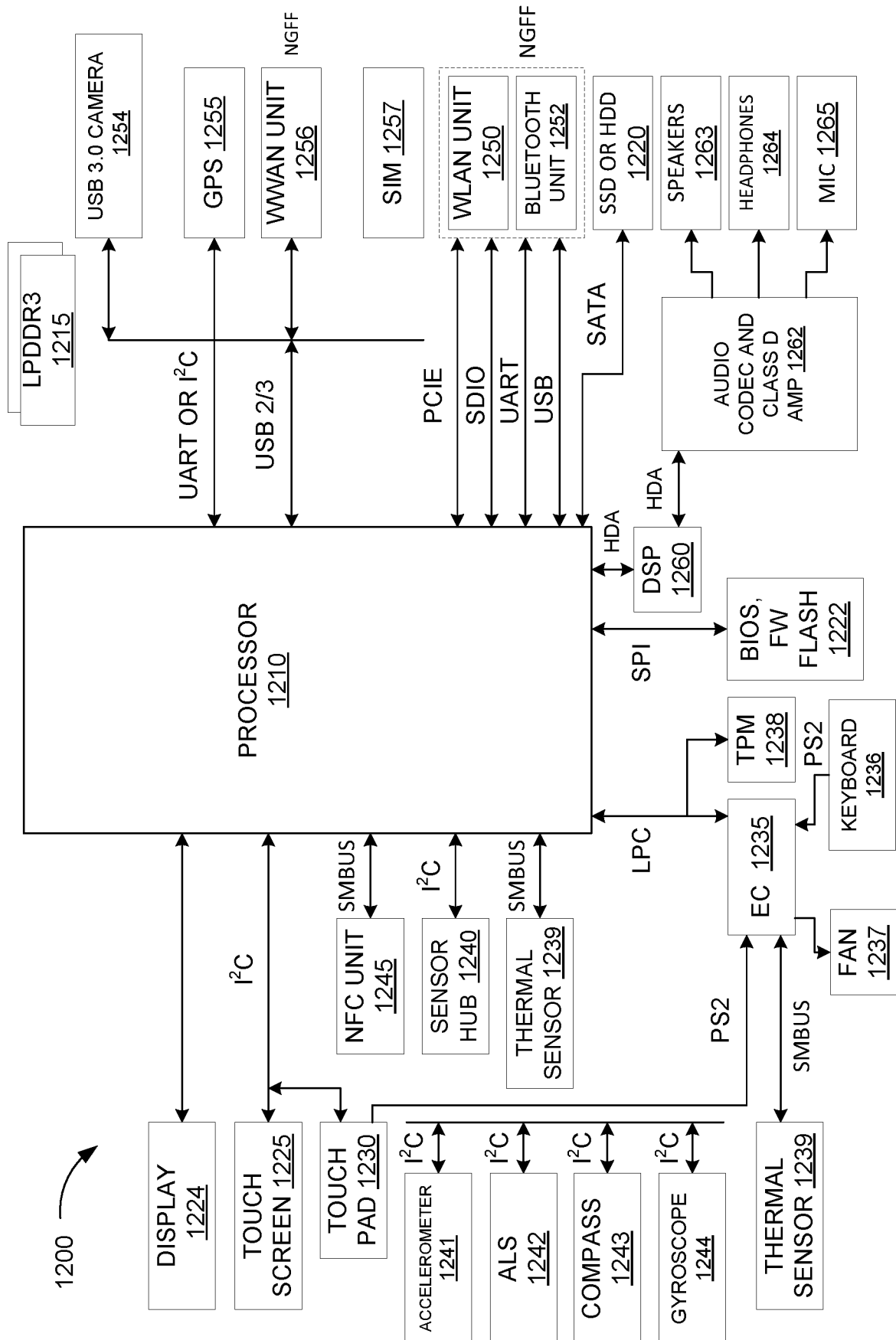


FIG. 12

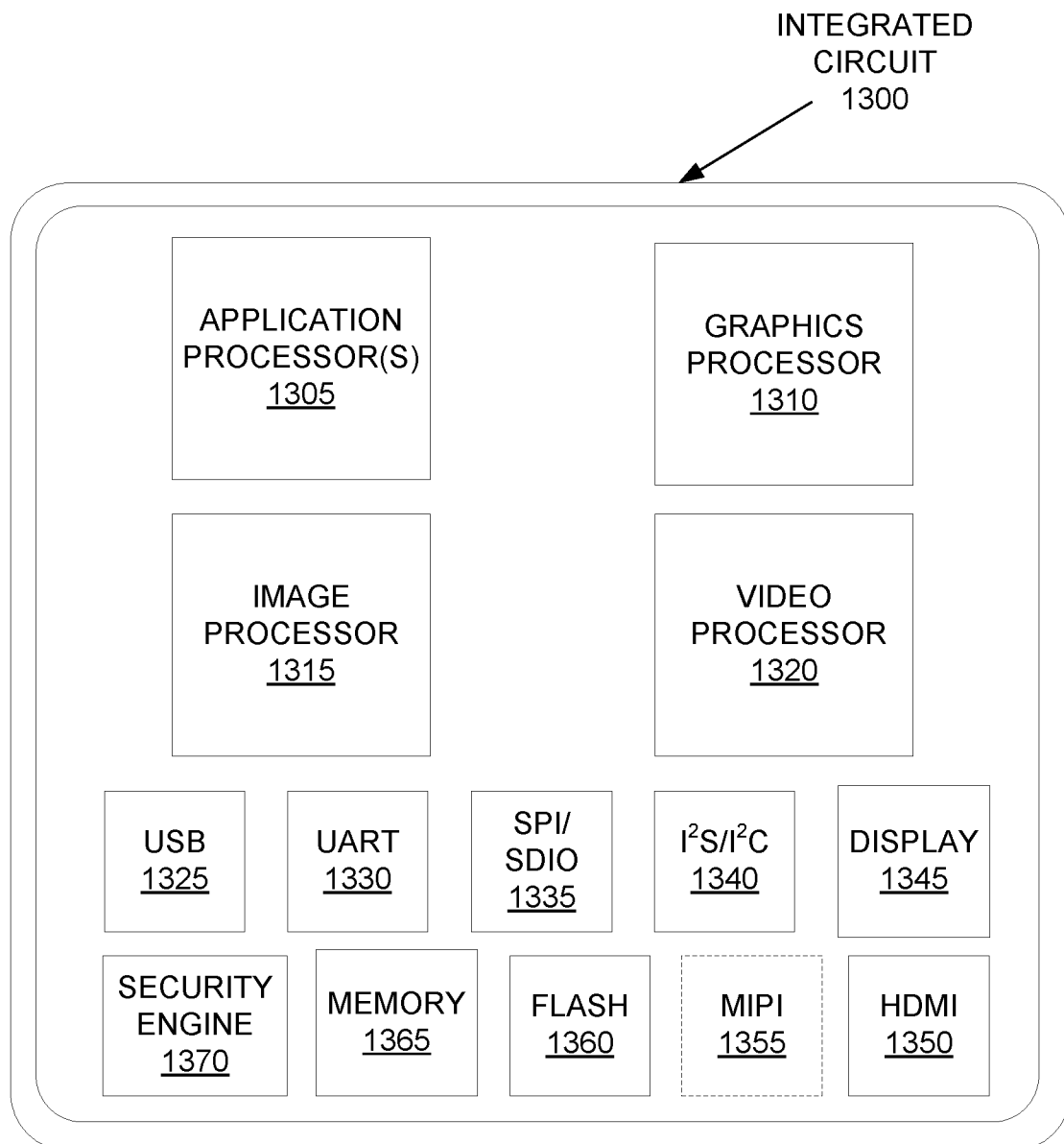


FIG. 13

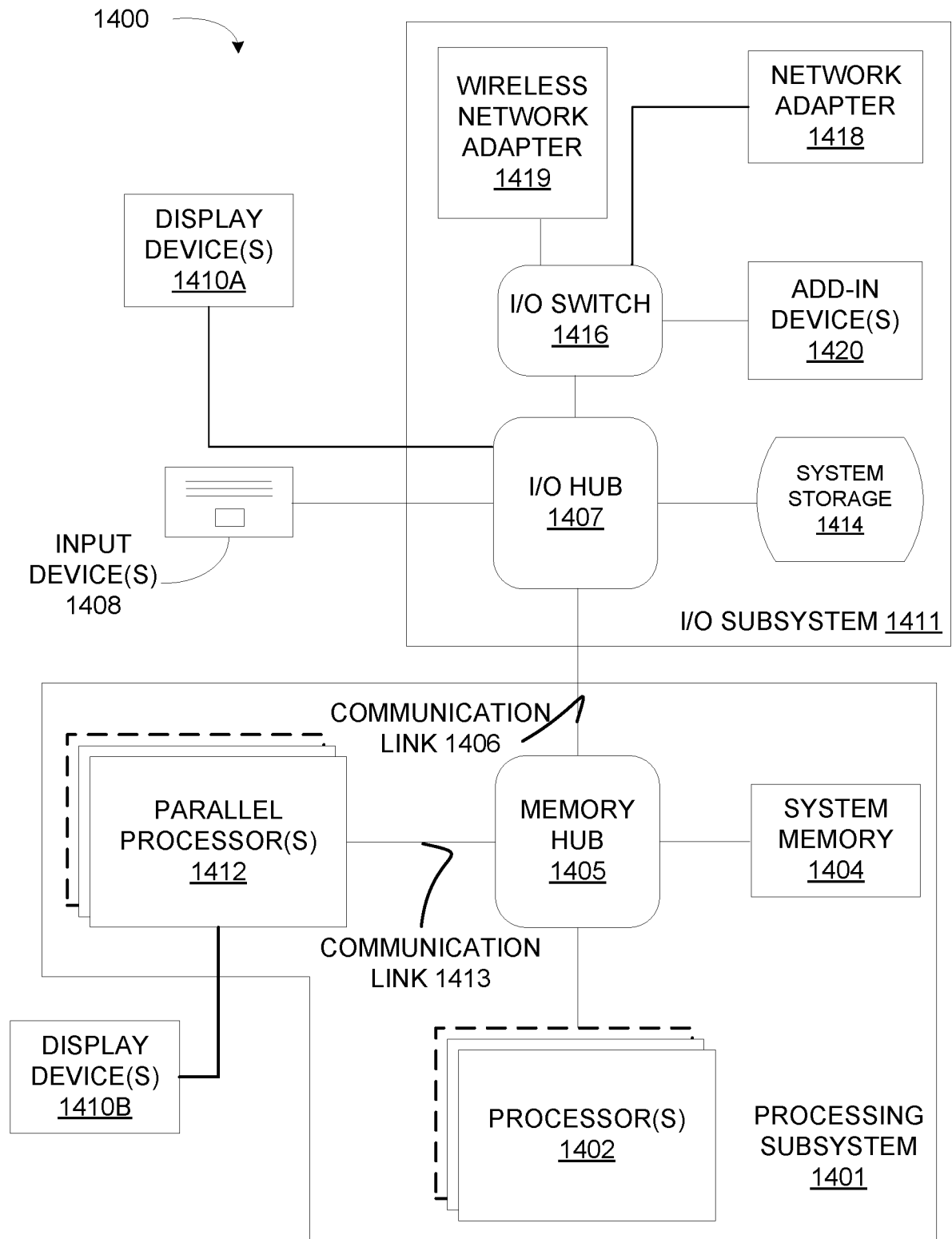


FIG. 14

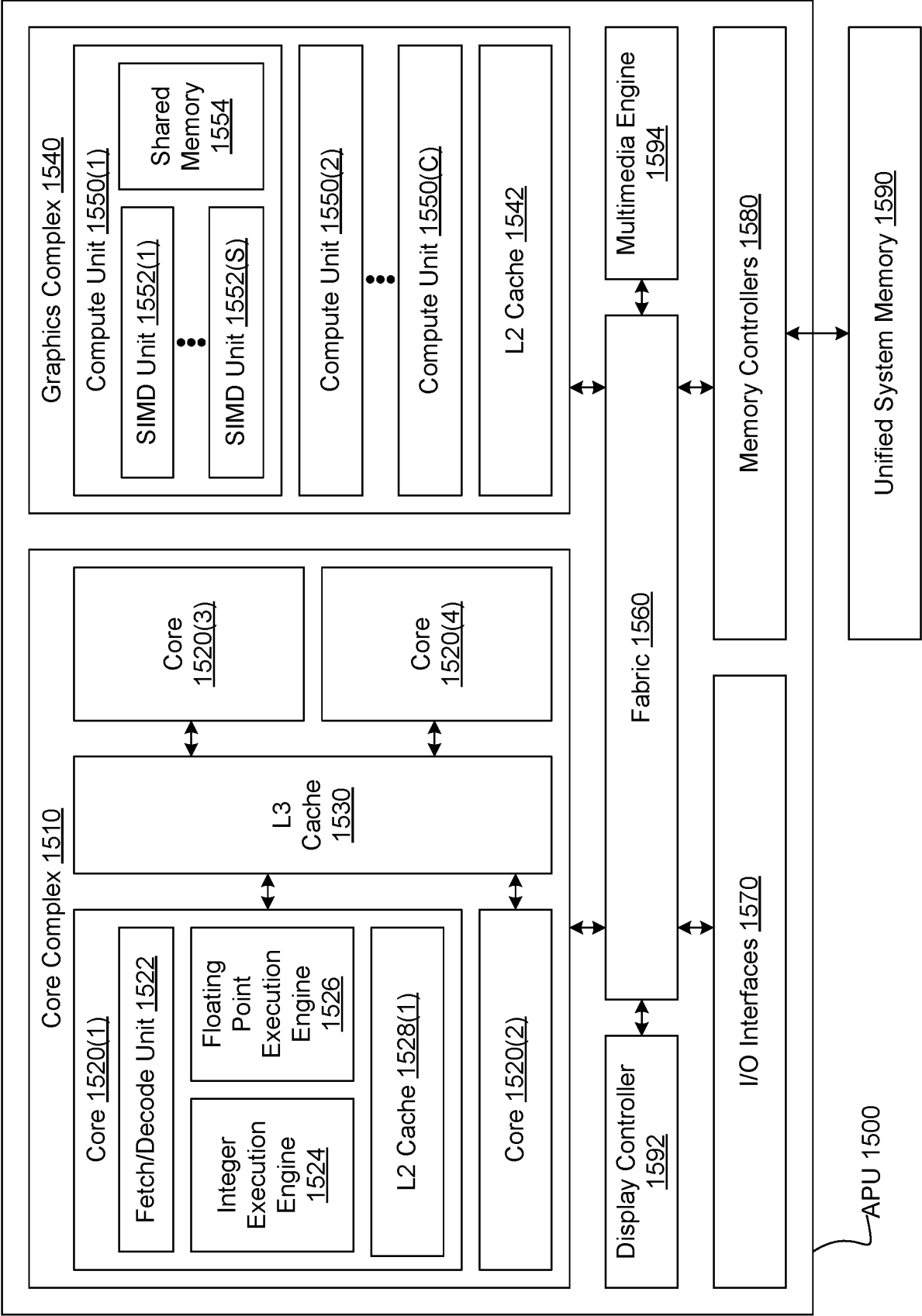


FIG. 15

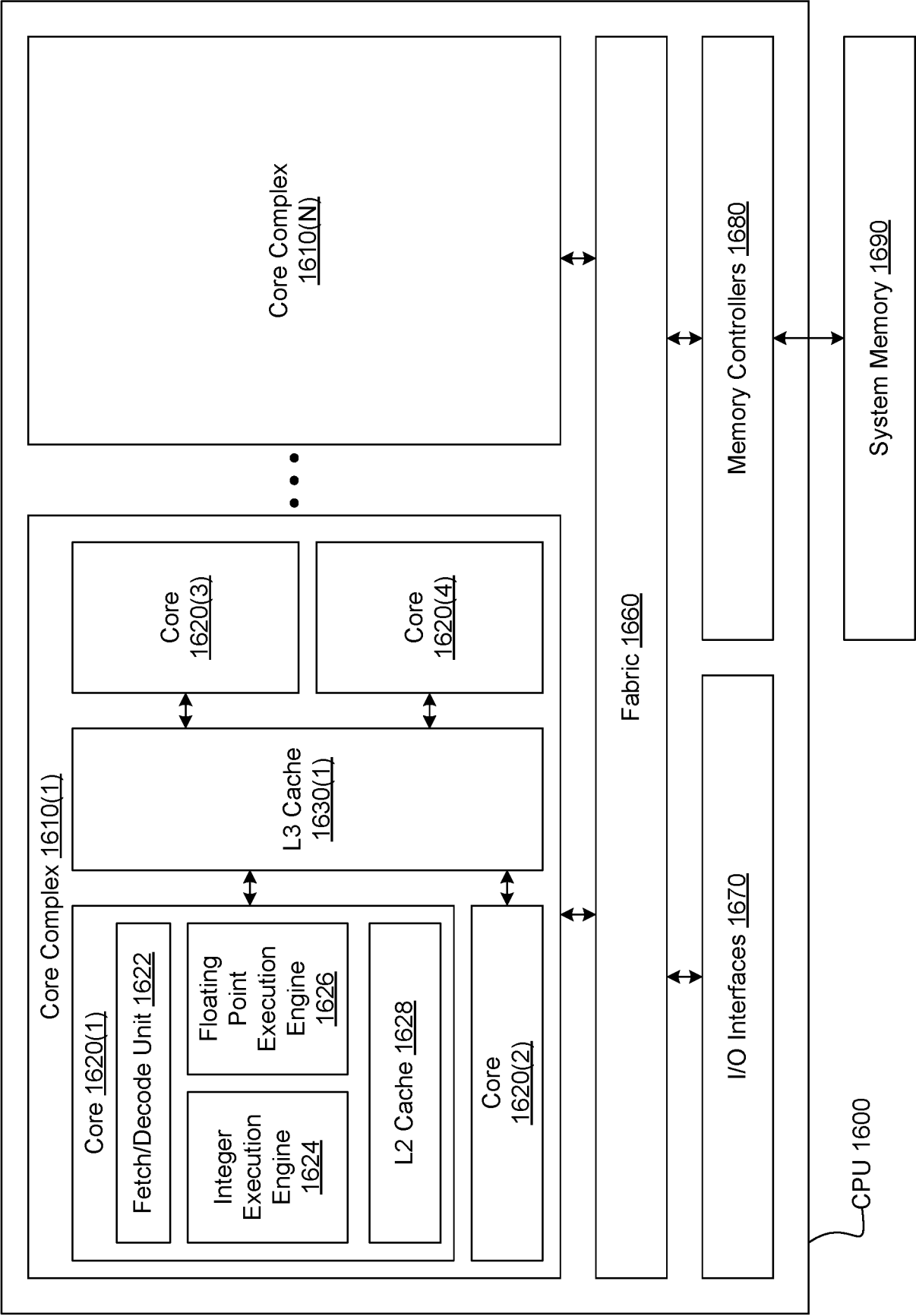


FIG. 16

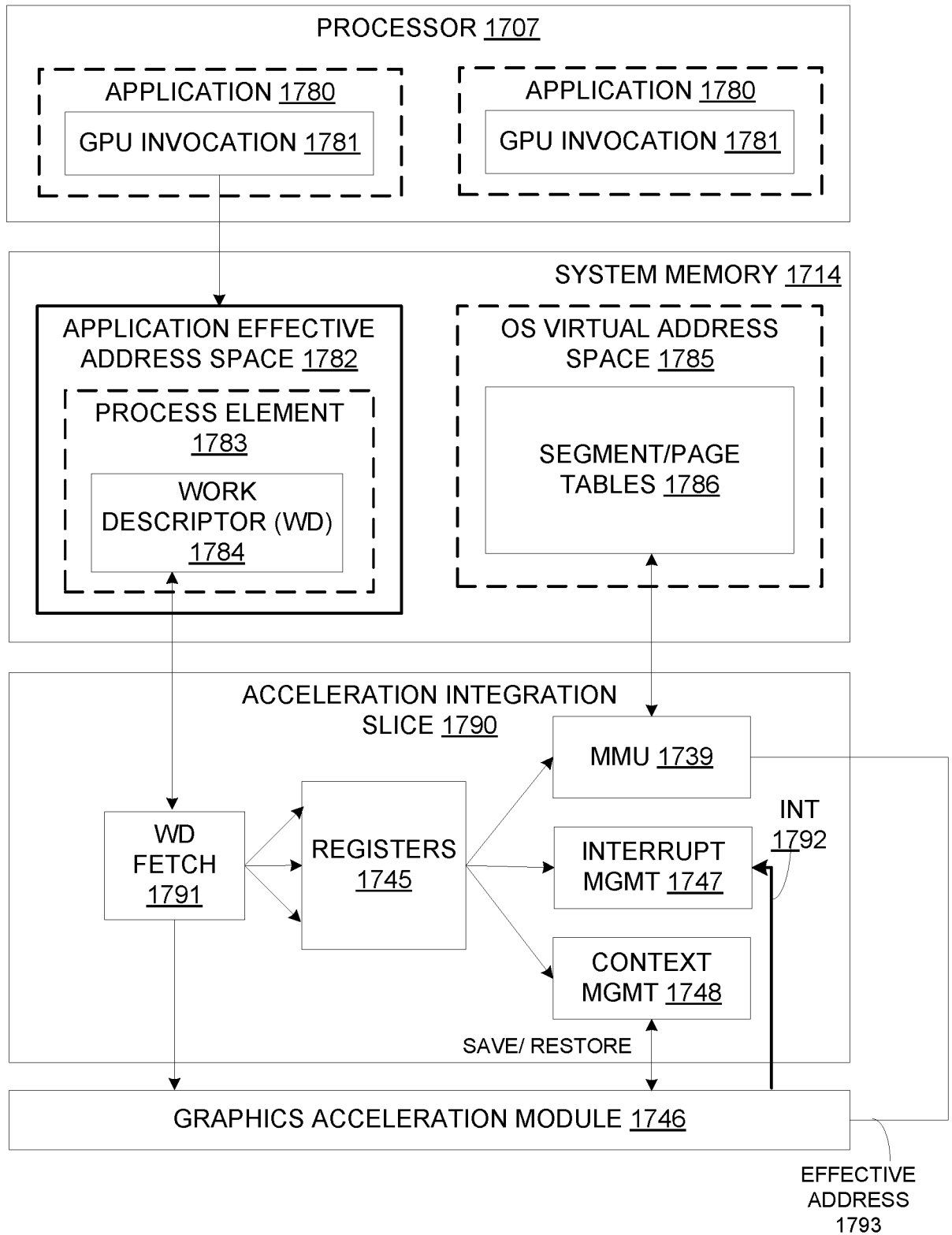


FIG. 17

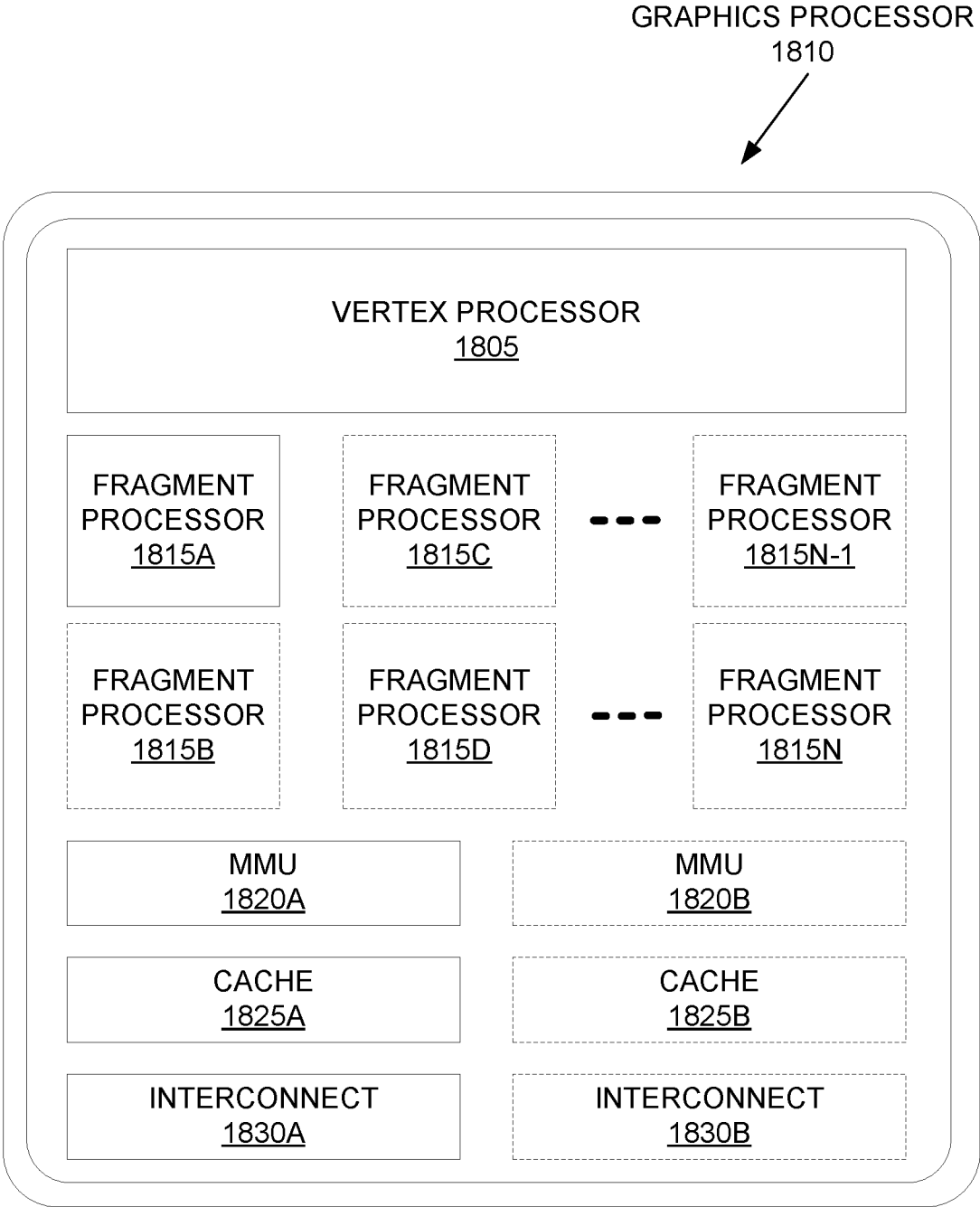
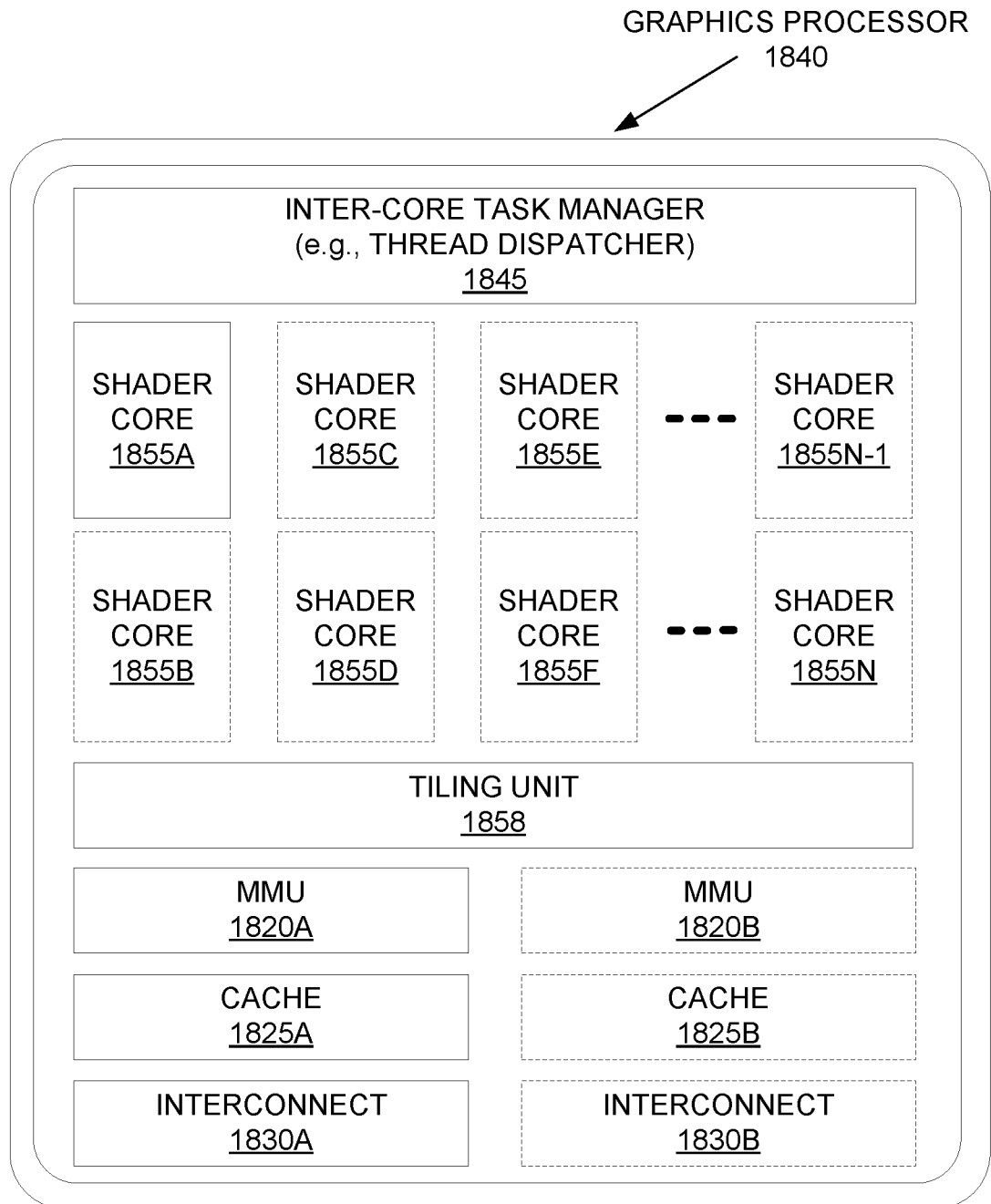


FIG. 18A

**FIG. 18B**

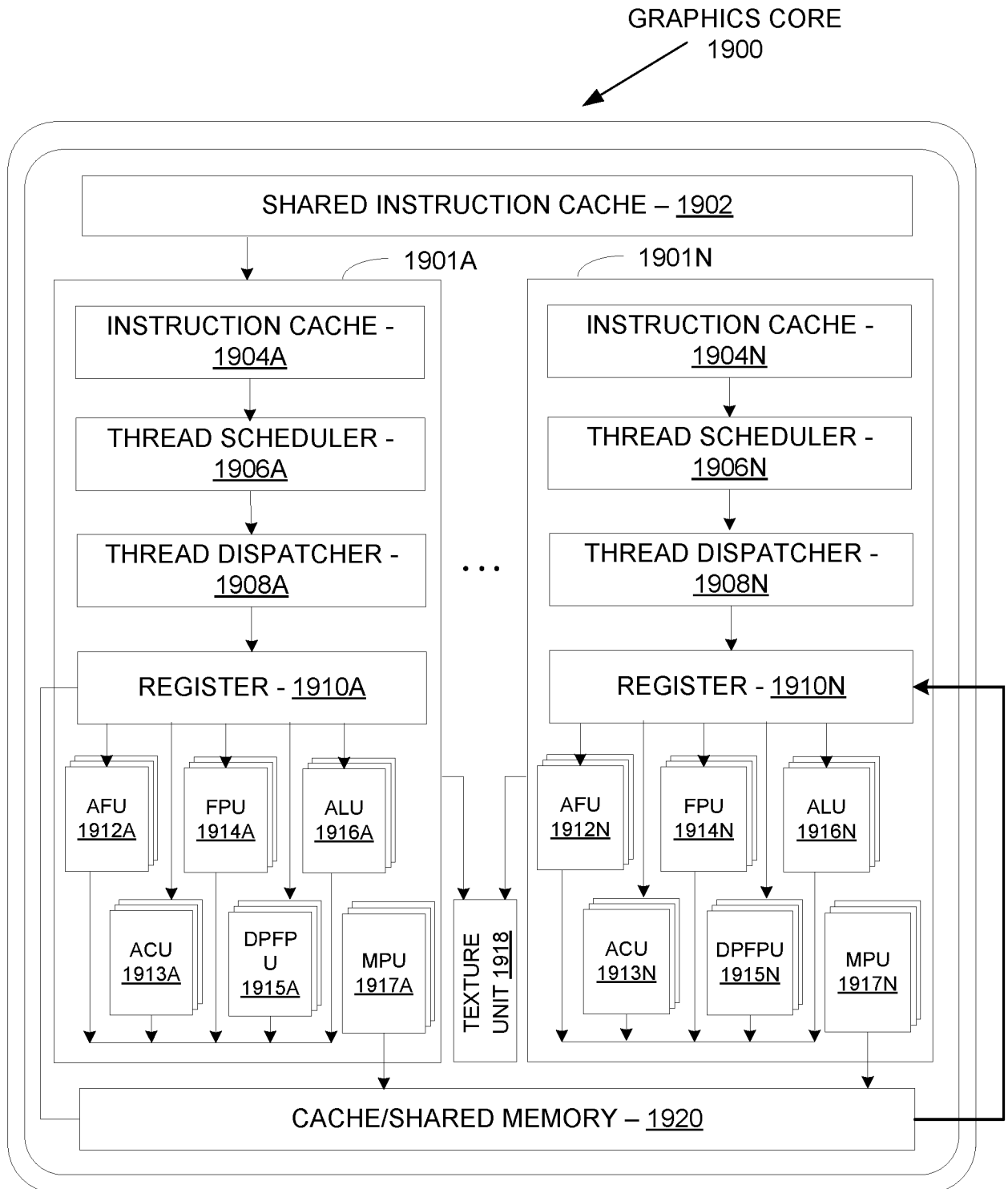


FIG. 19A



FIG. 19B

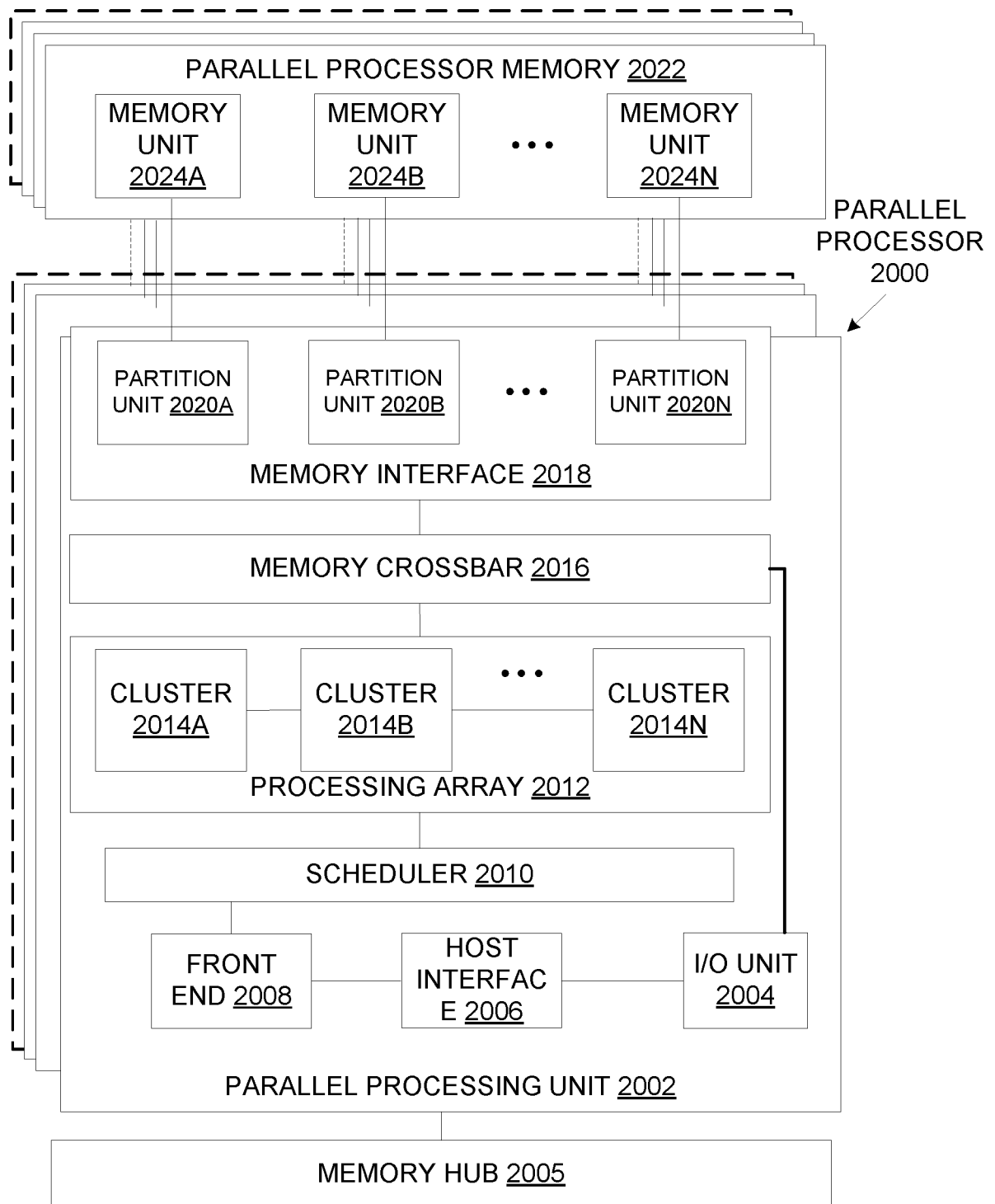


FIG. 20A

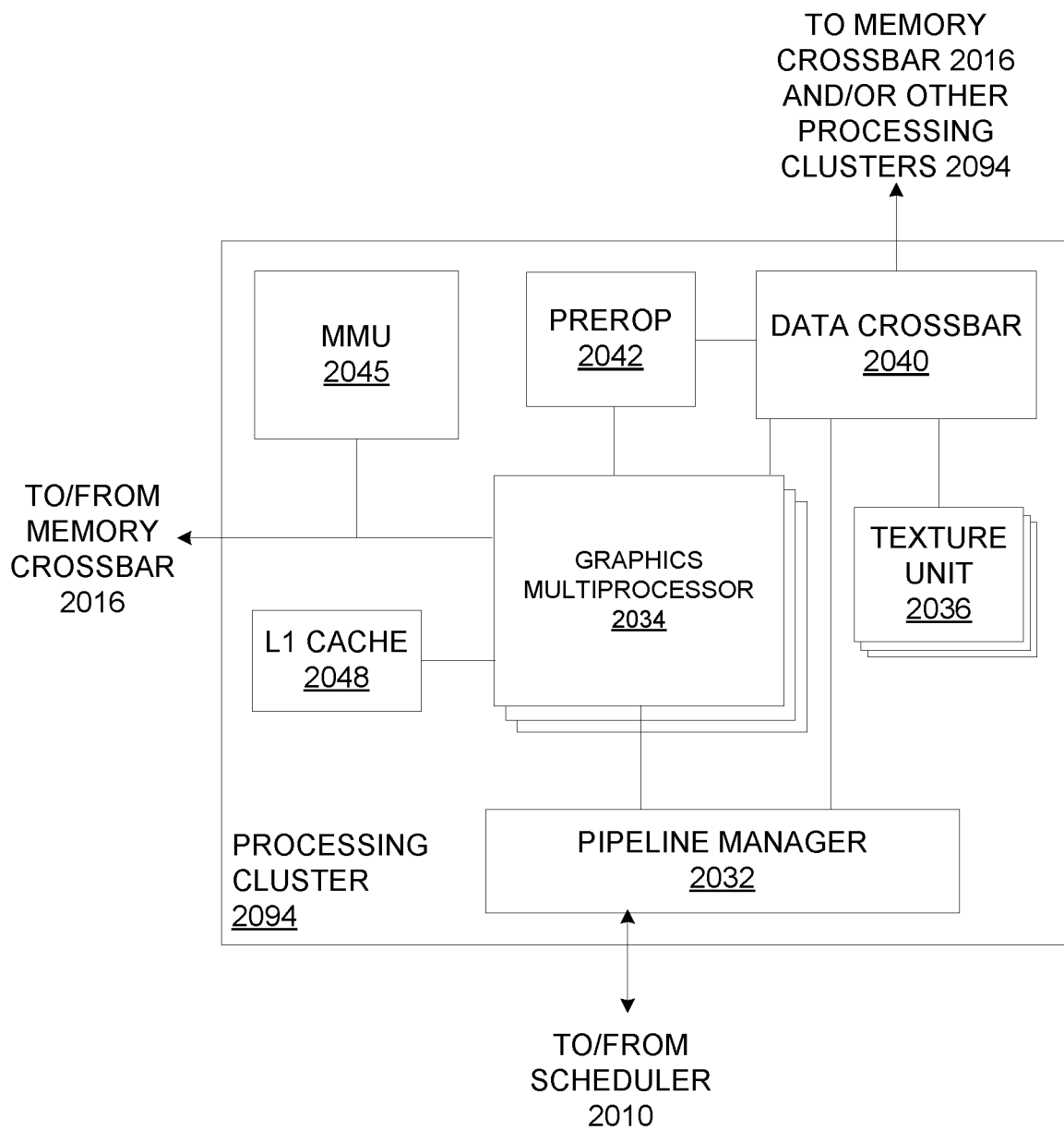


FIG. 20B

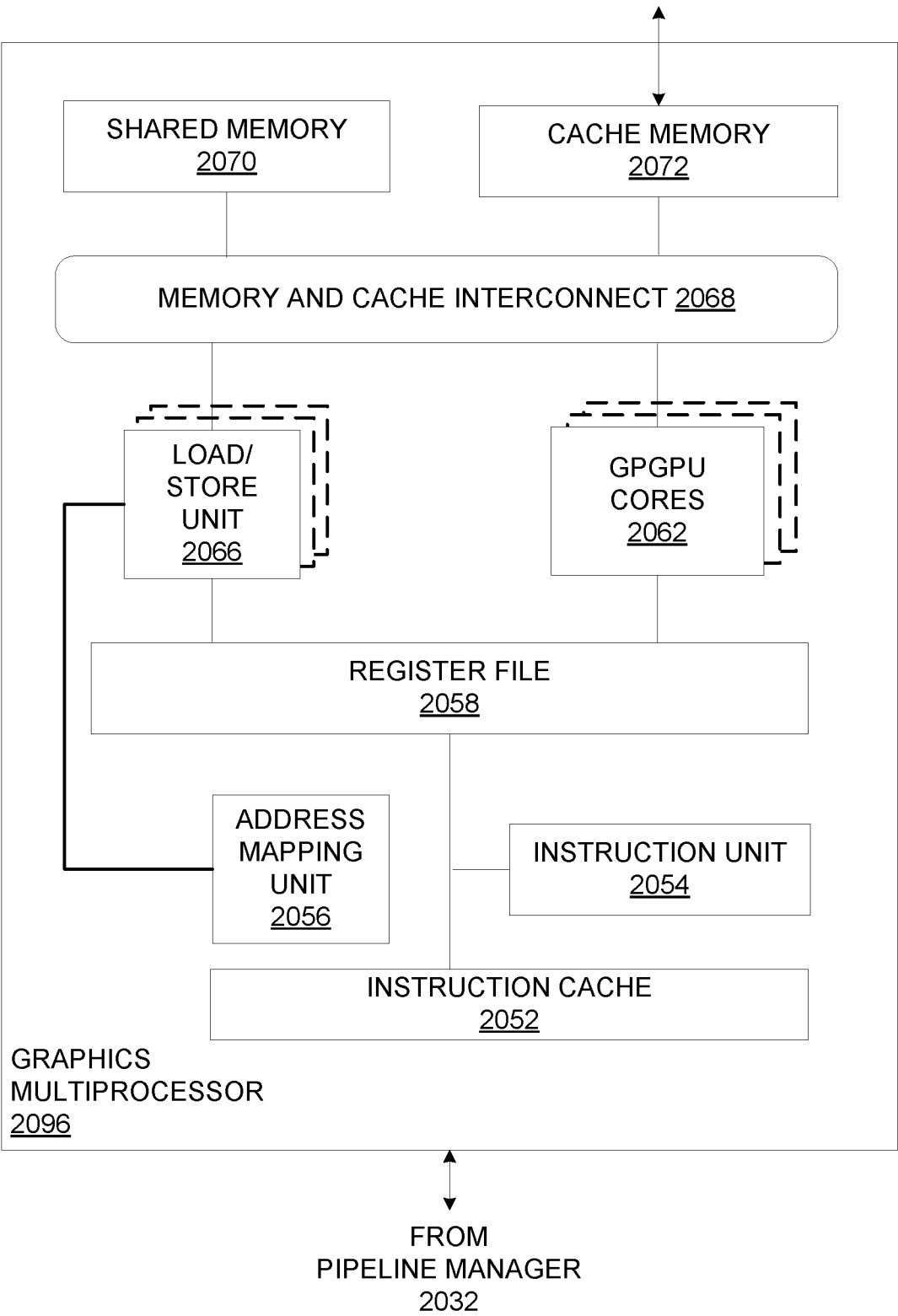


FIG. 20C

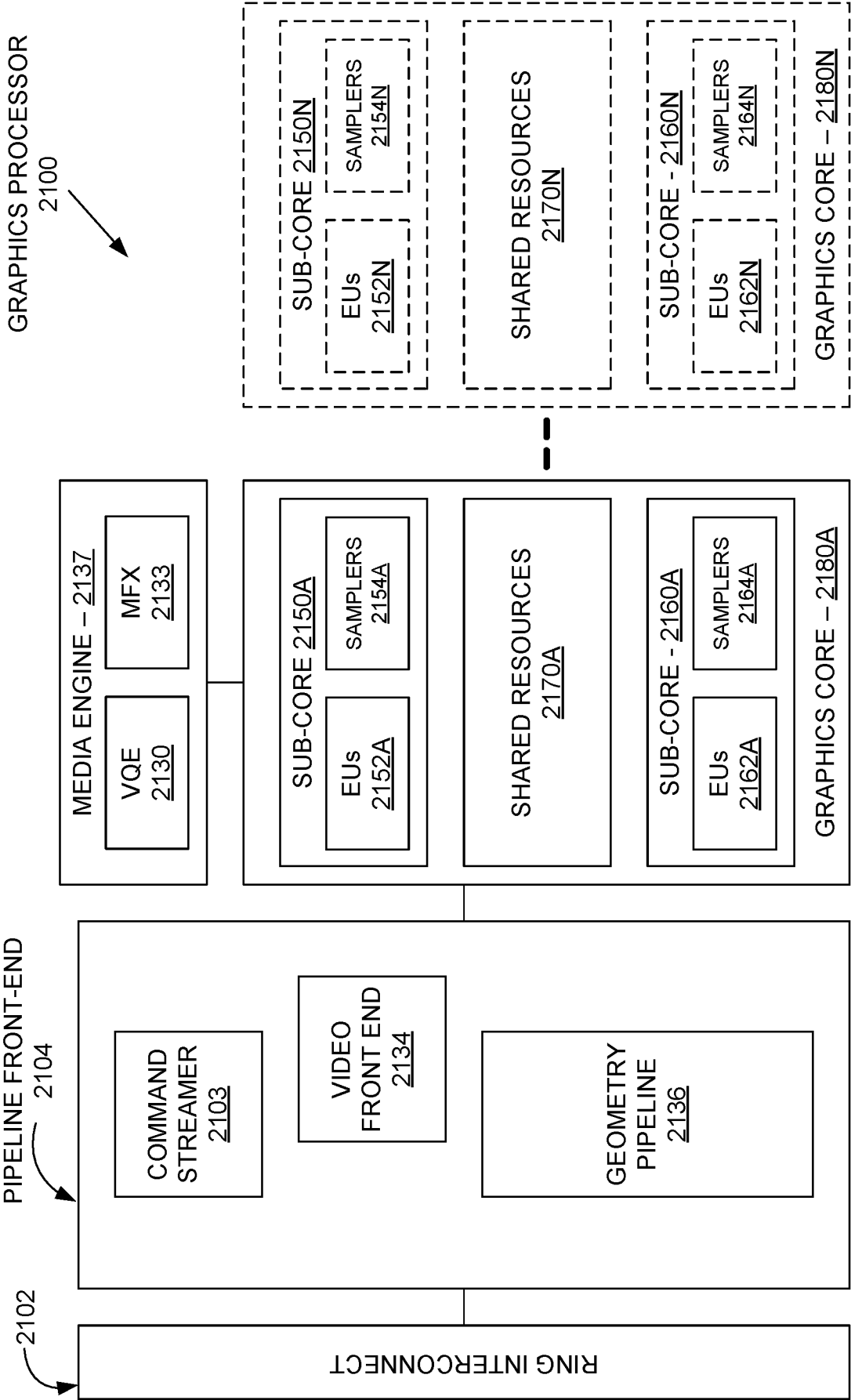


FIG. 21

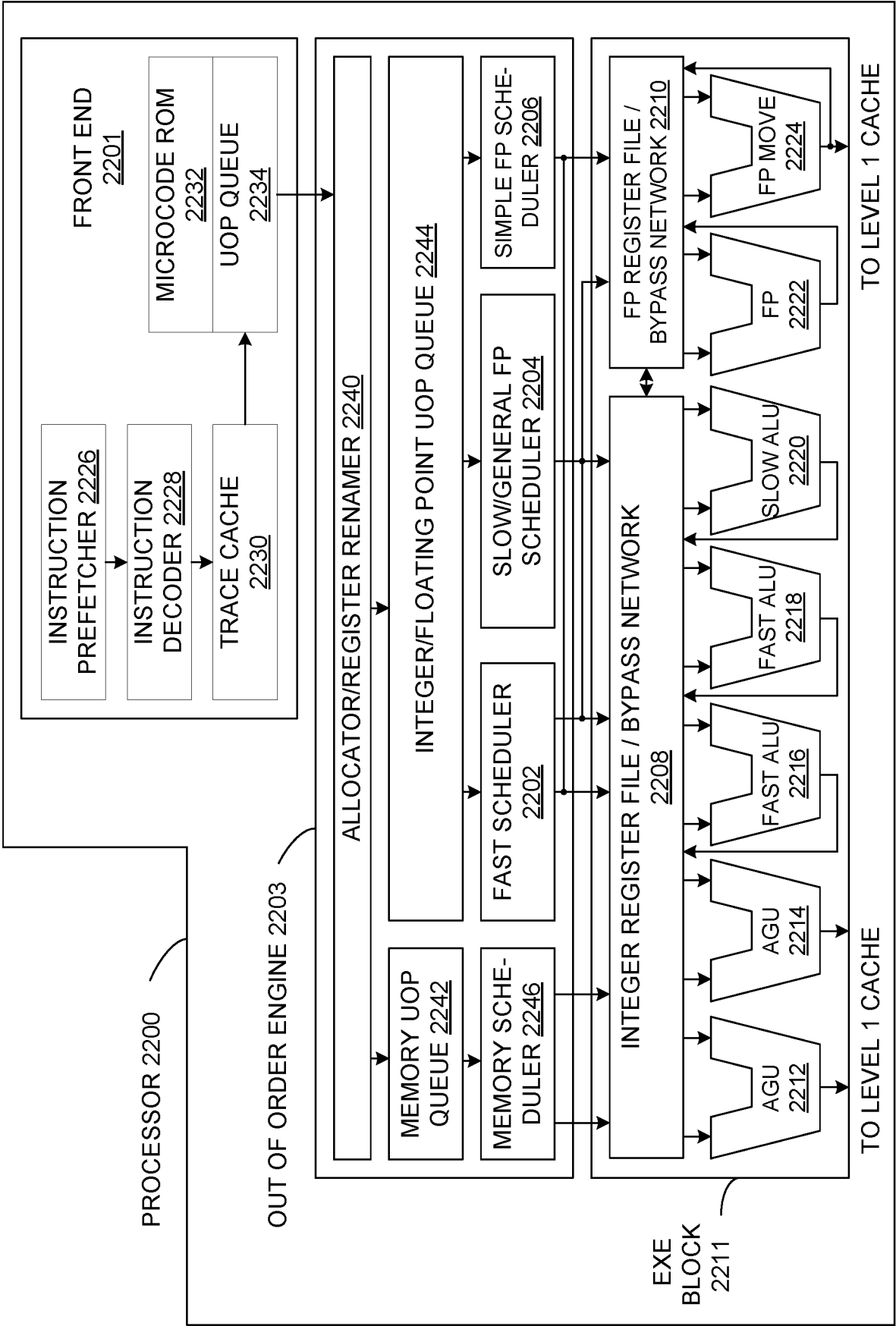


FIG. 22

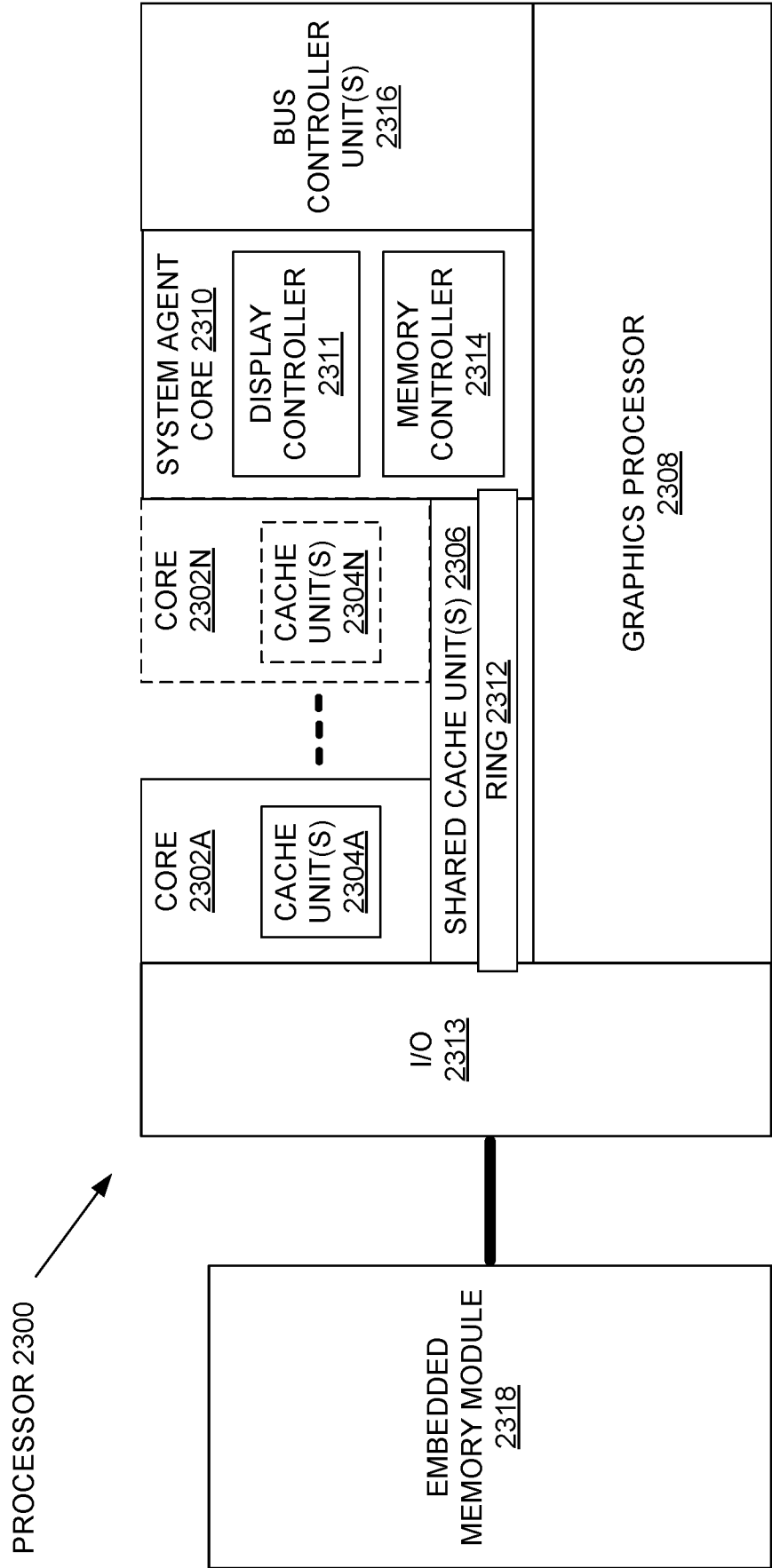


FIG. 23

2400

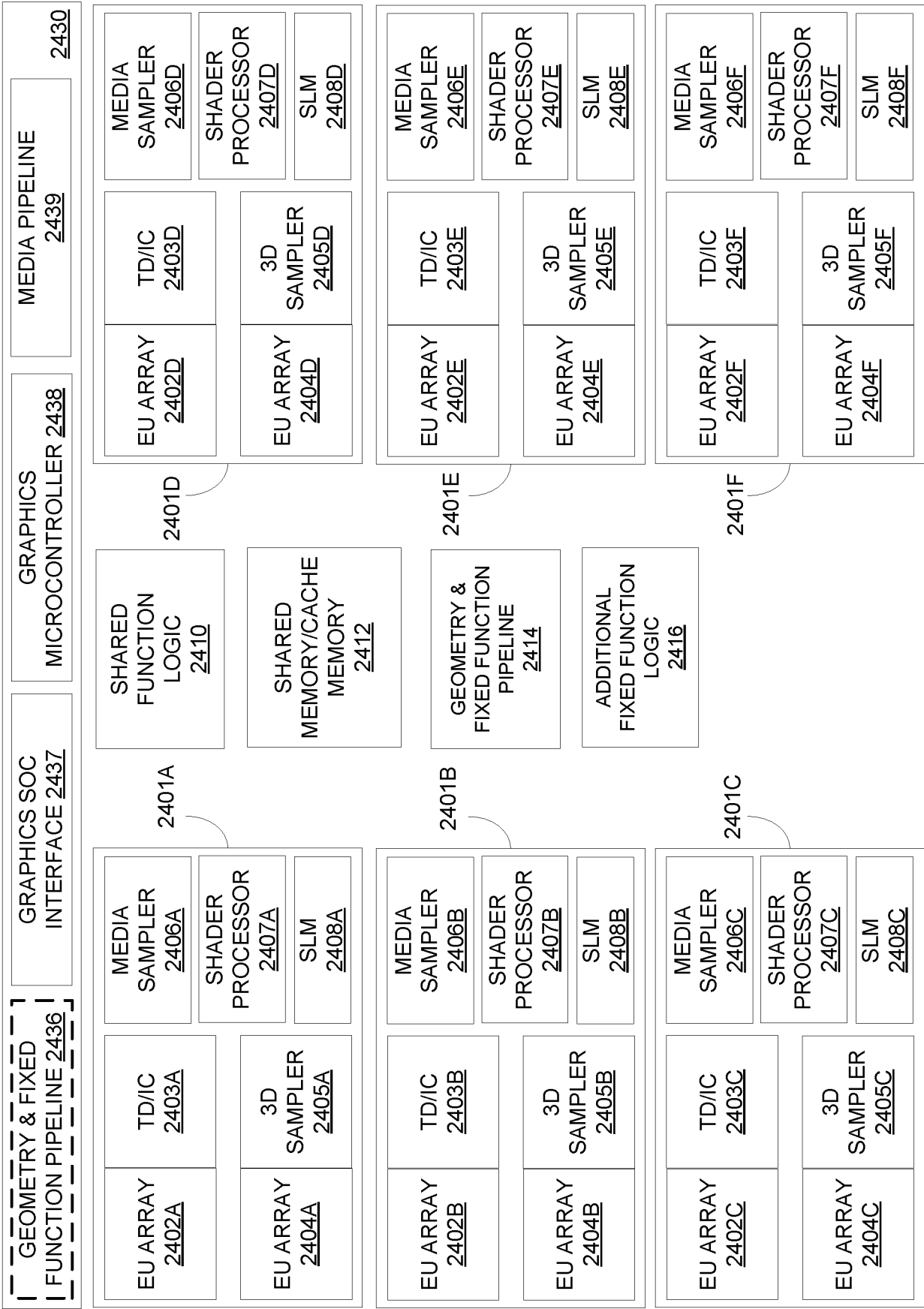


FIG. 24

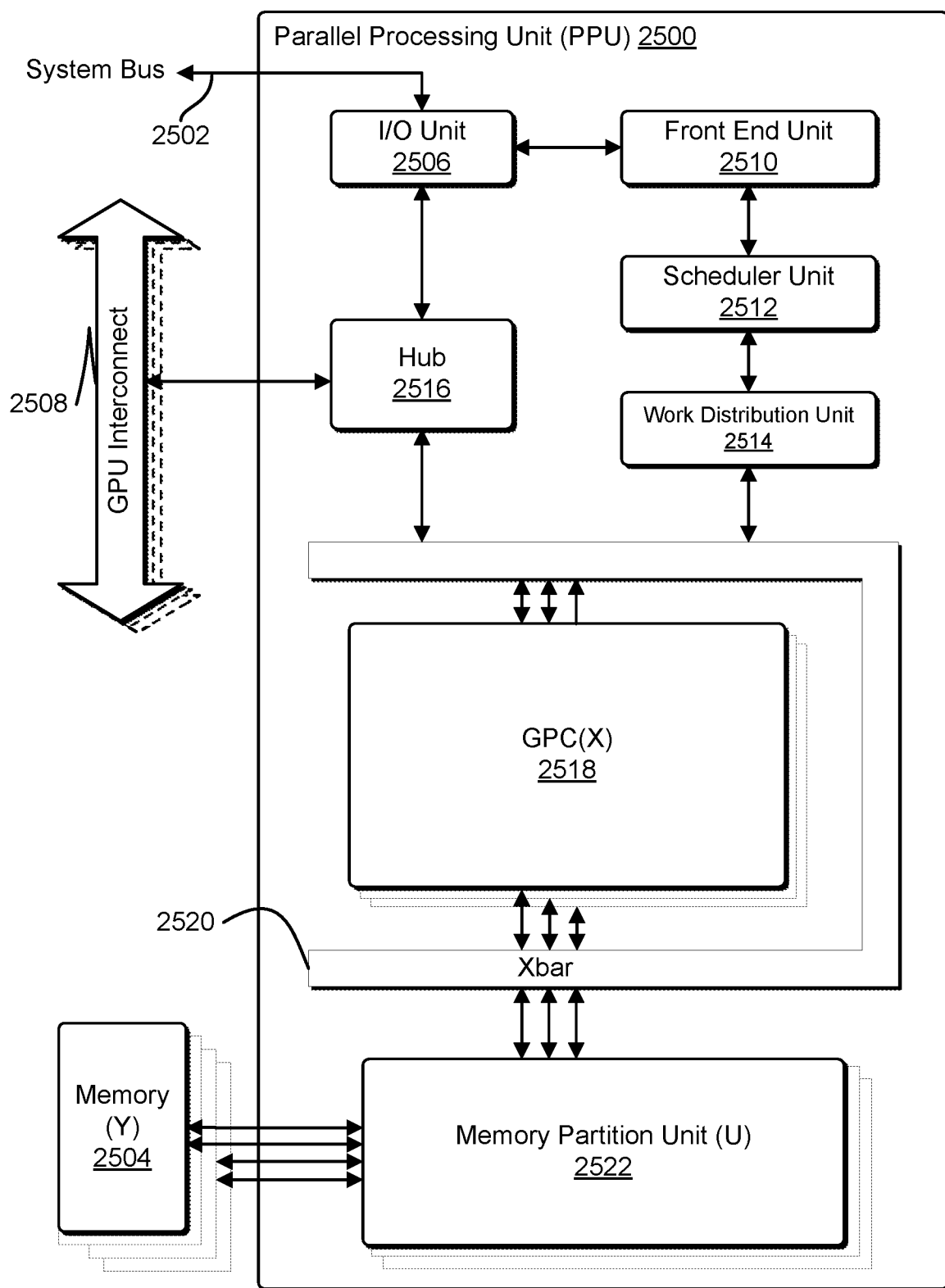


FIG. 25

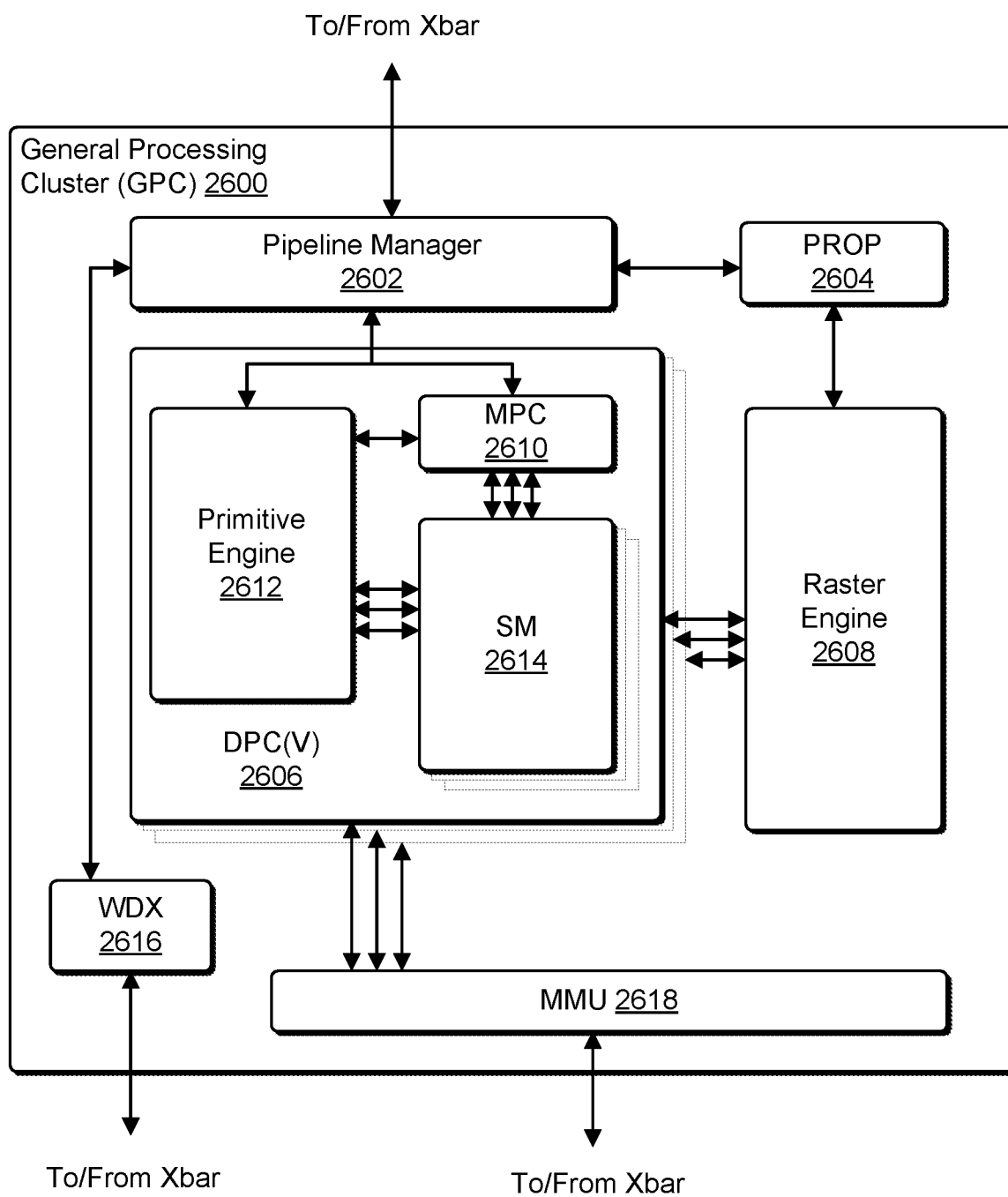


FIG. 26

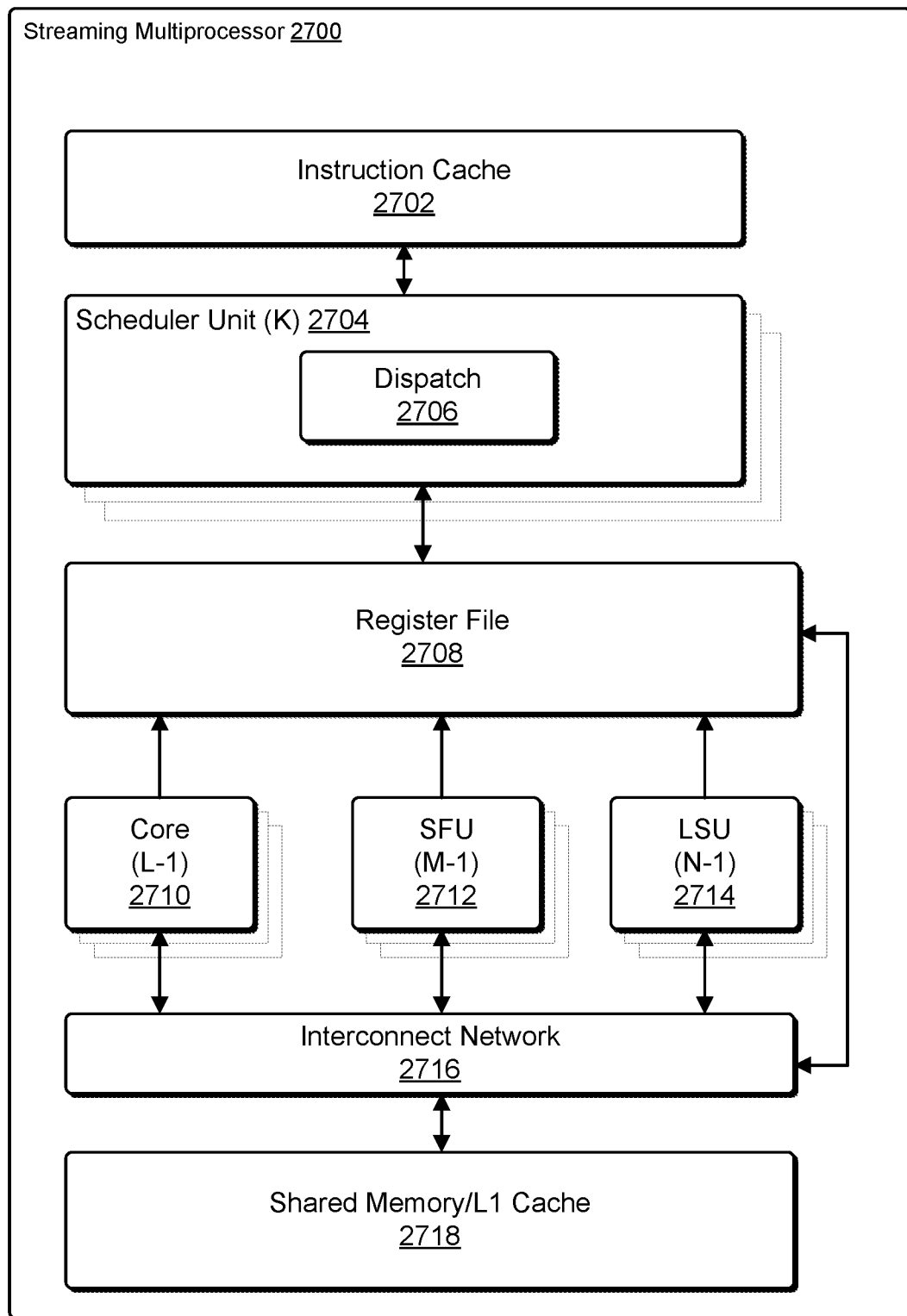
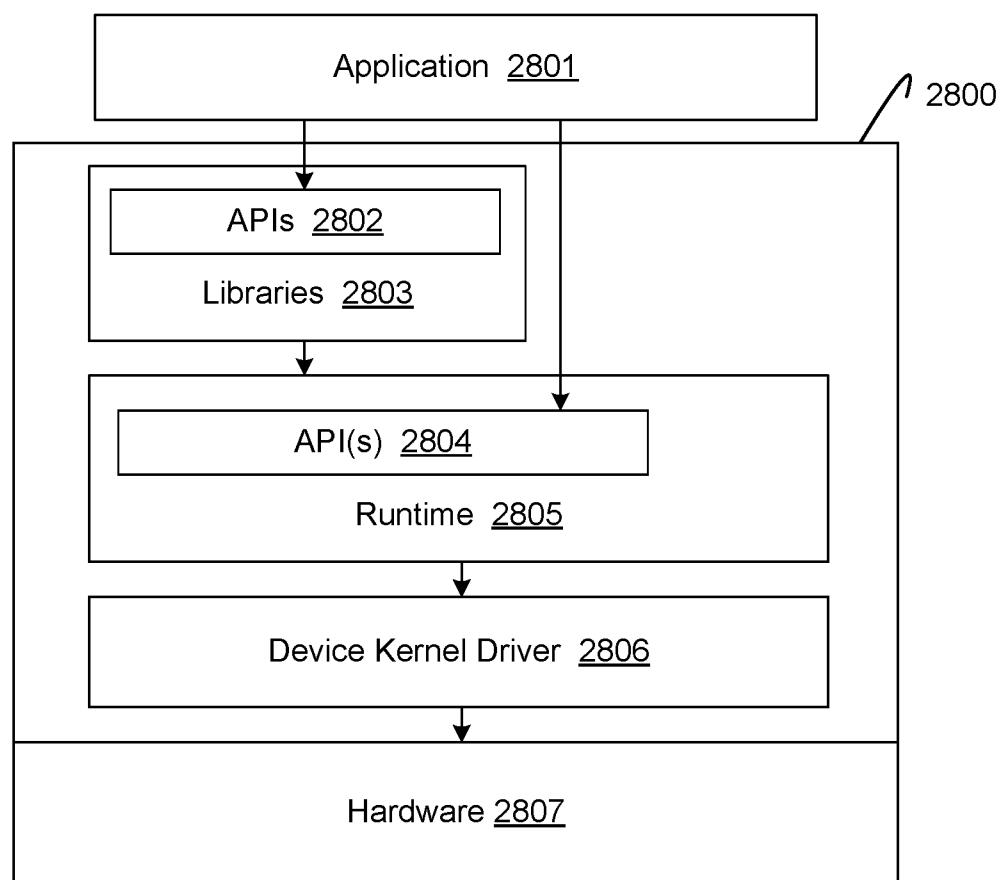
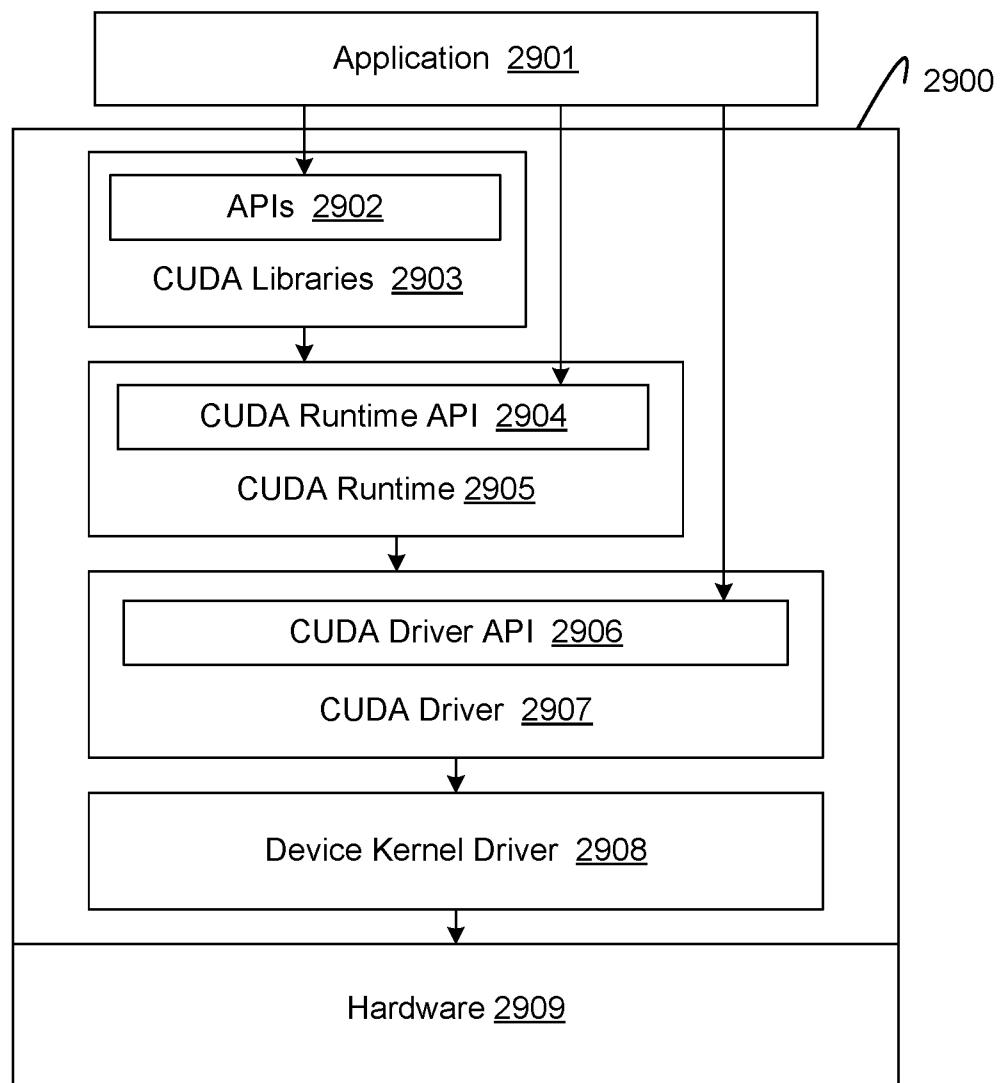


FIG. 27

**FIG. 28**

**FIG. 29**

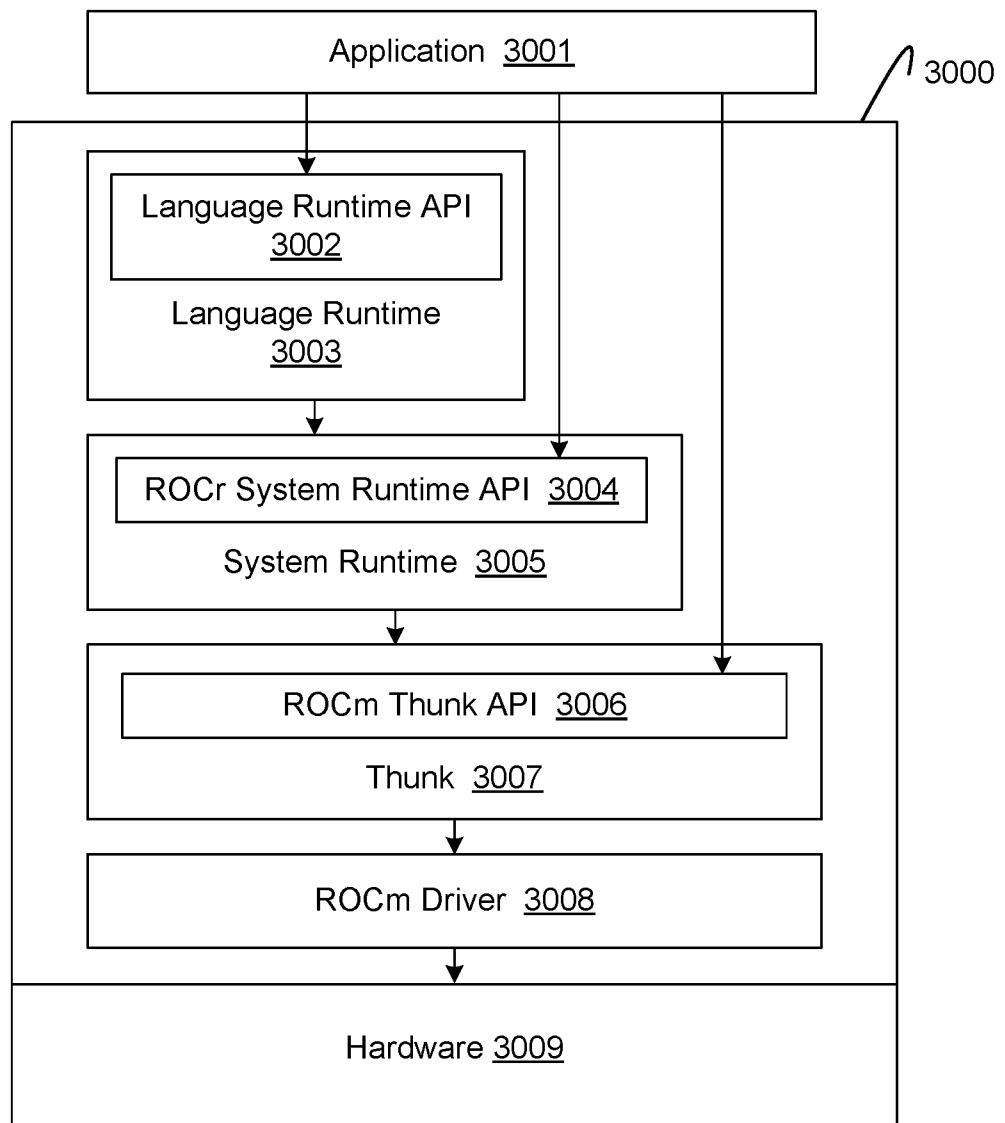
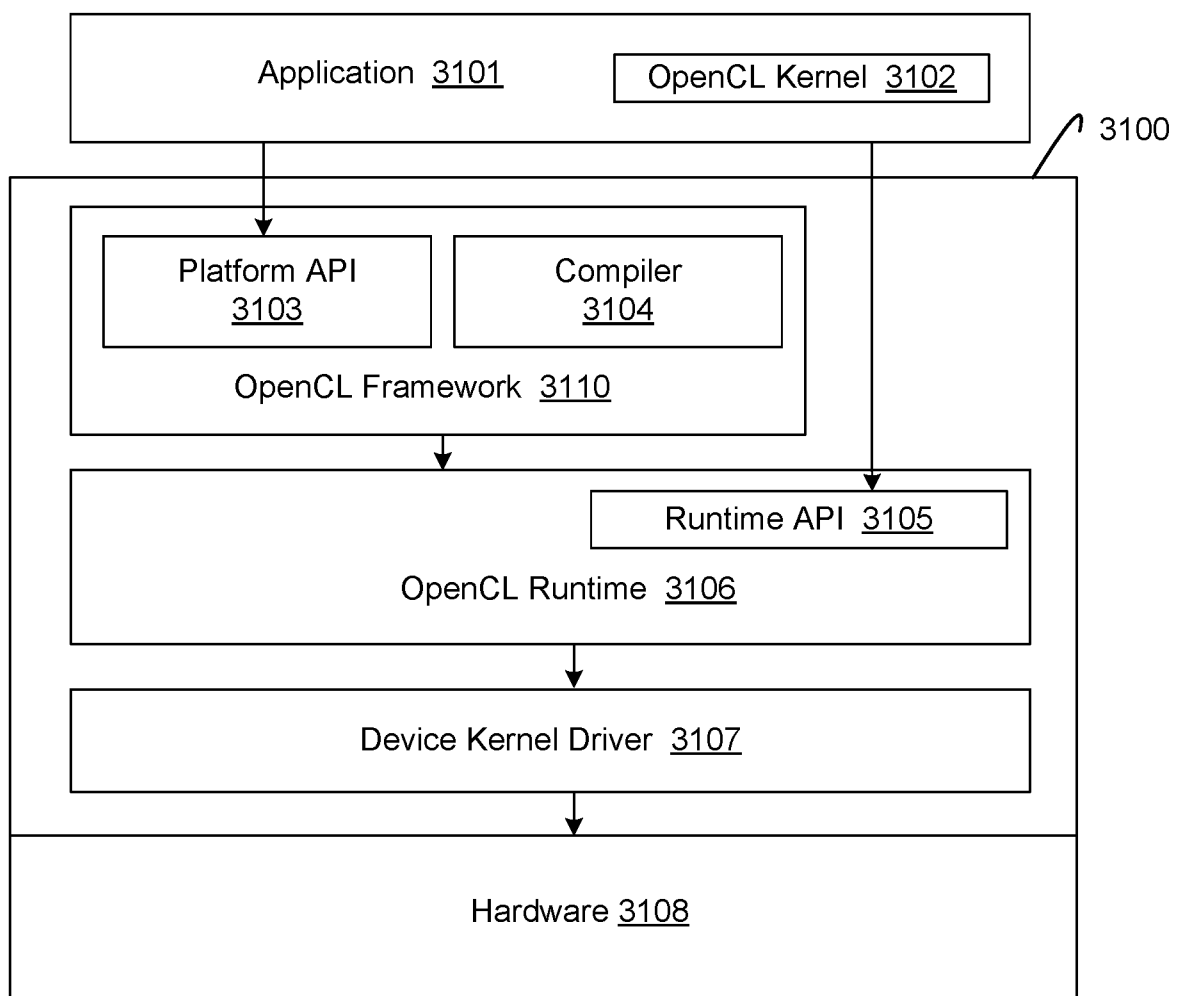


FIG. 30

**FIG. 31**

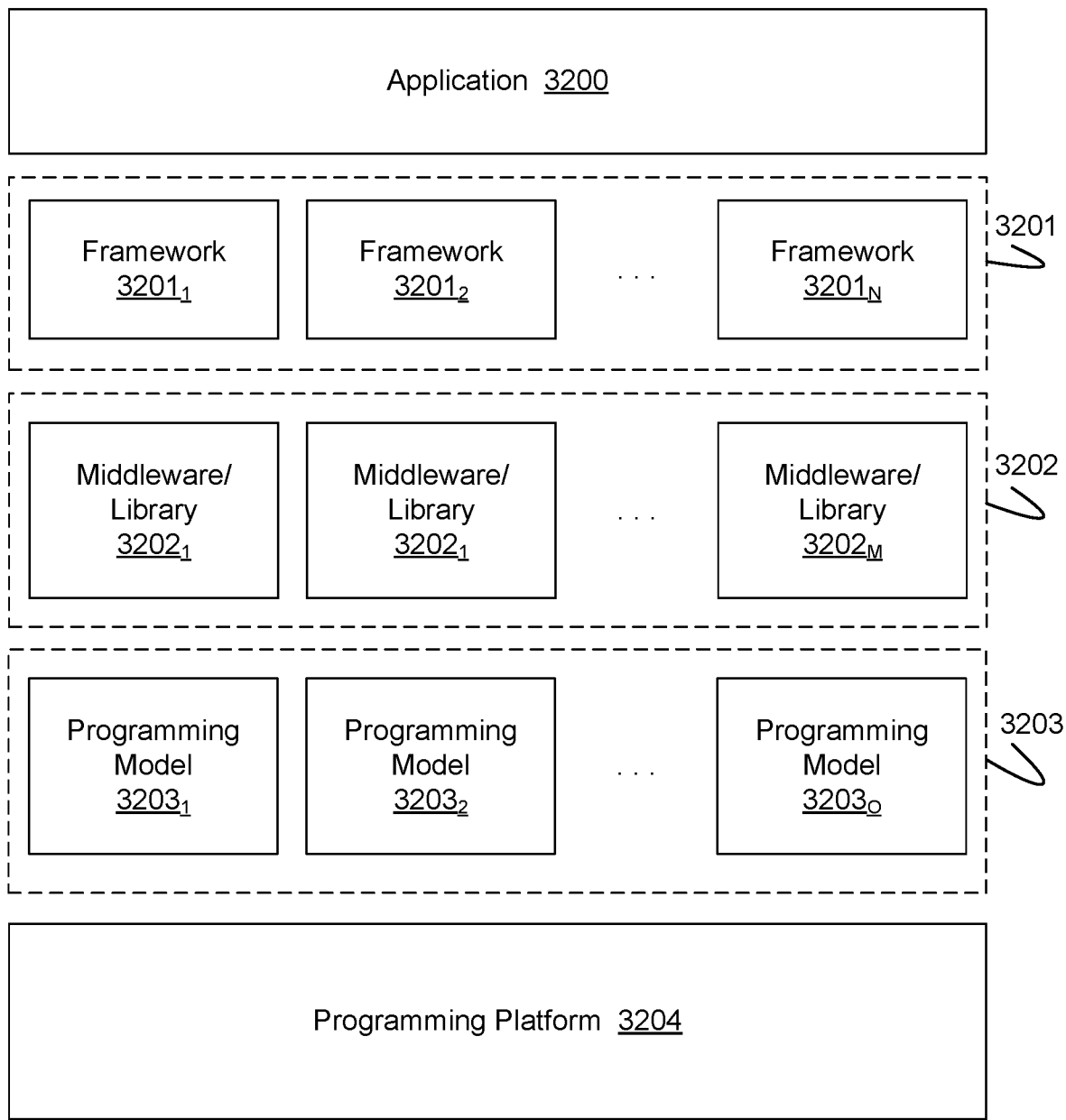
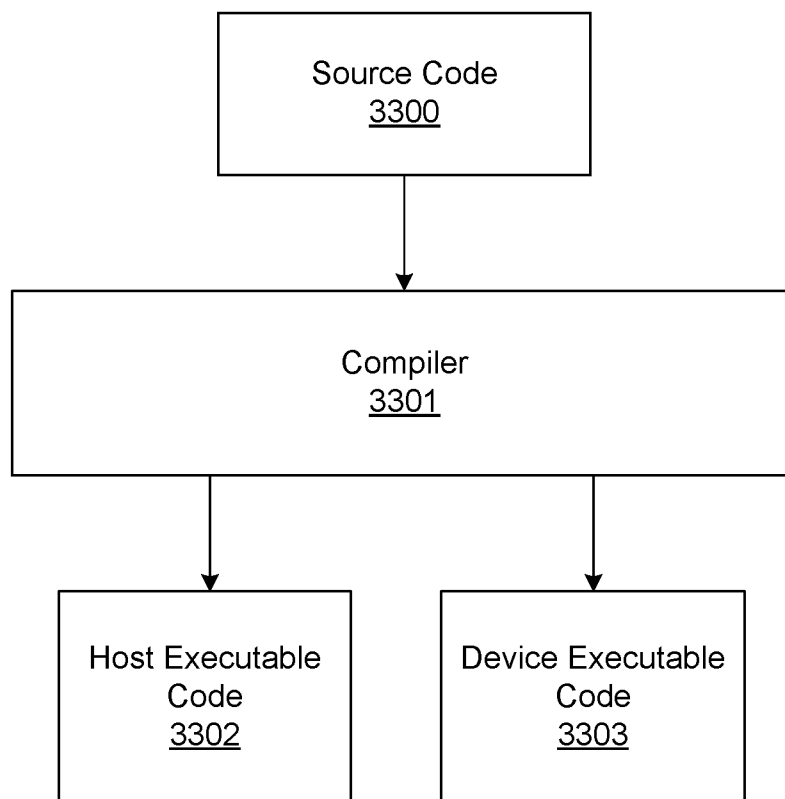


FIG. 32

**FIG. 33**

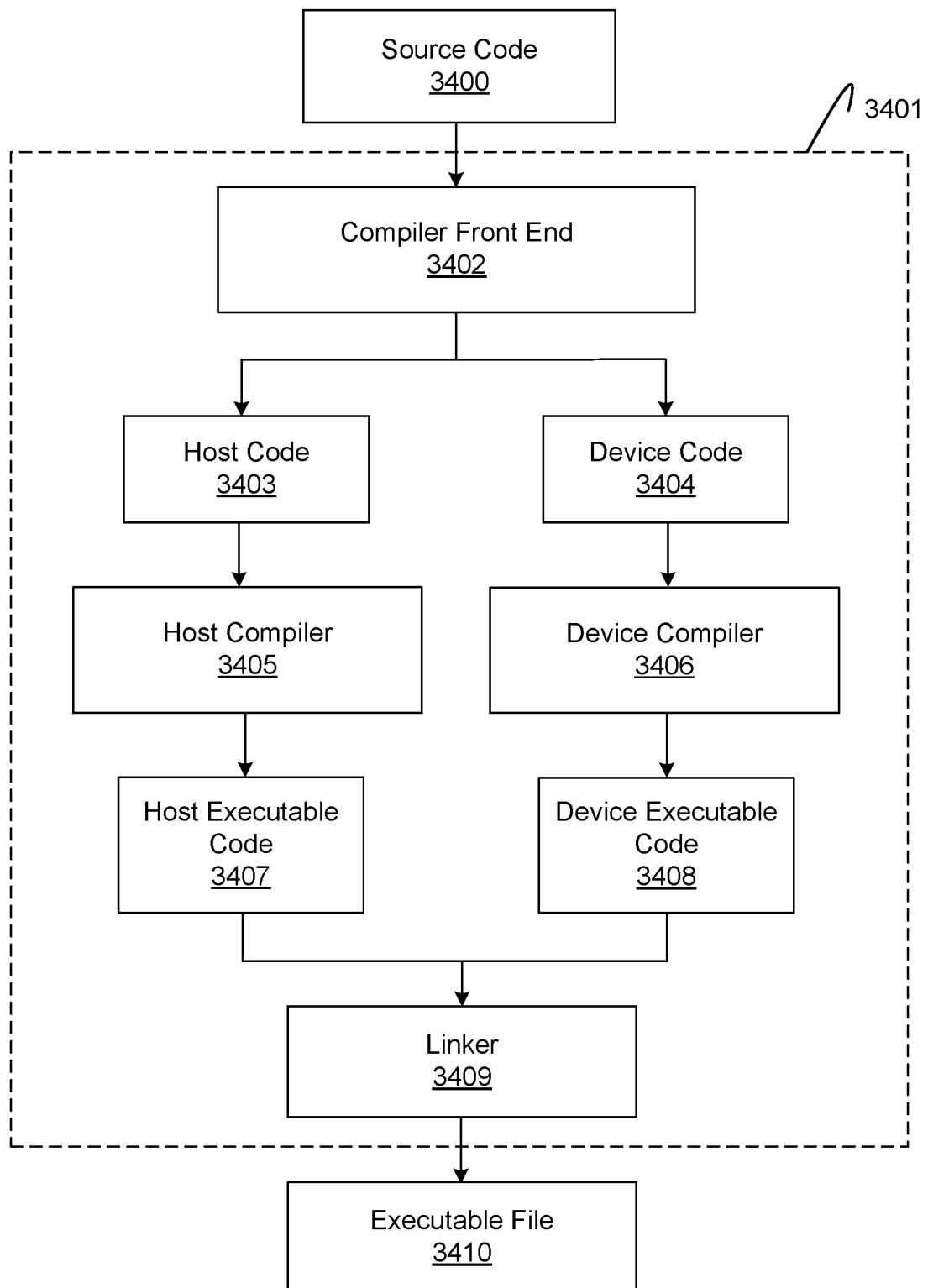
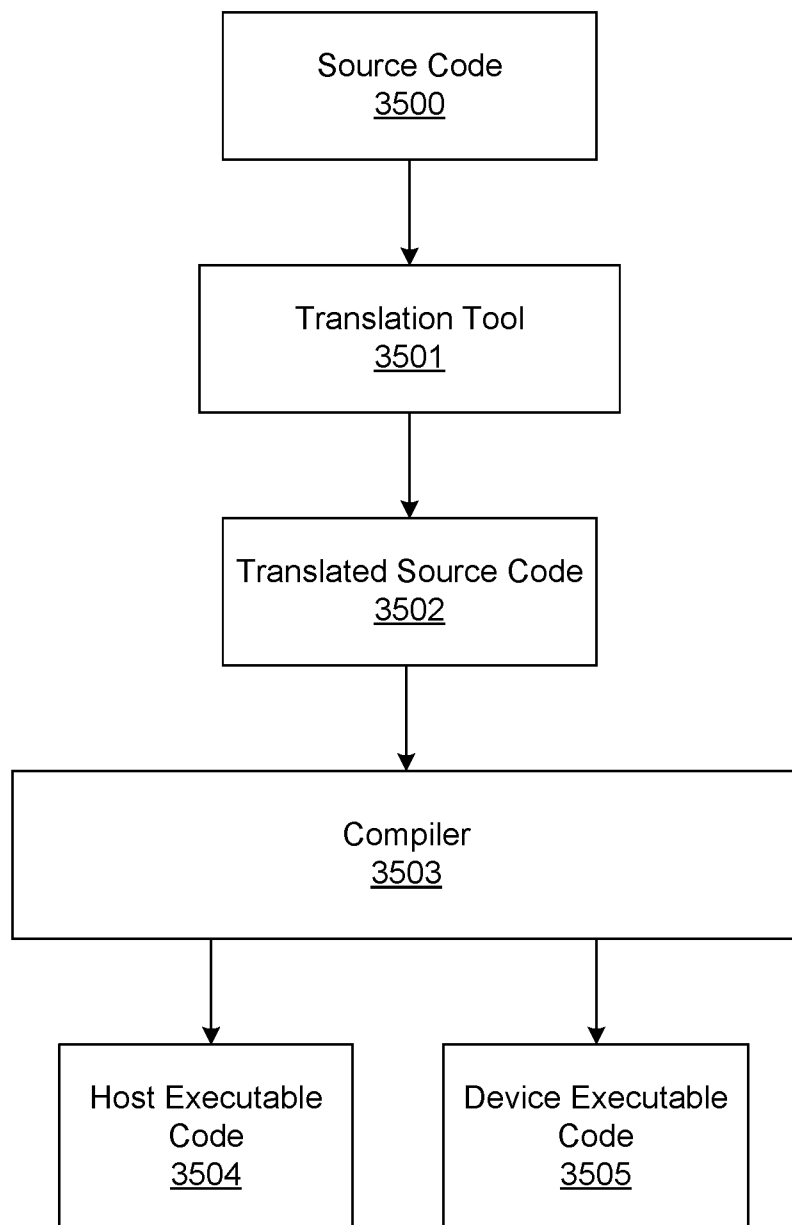


FIG. 34

**FIG. 35**

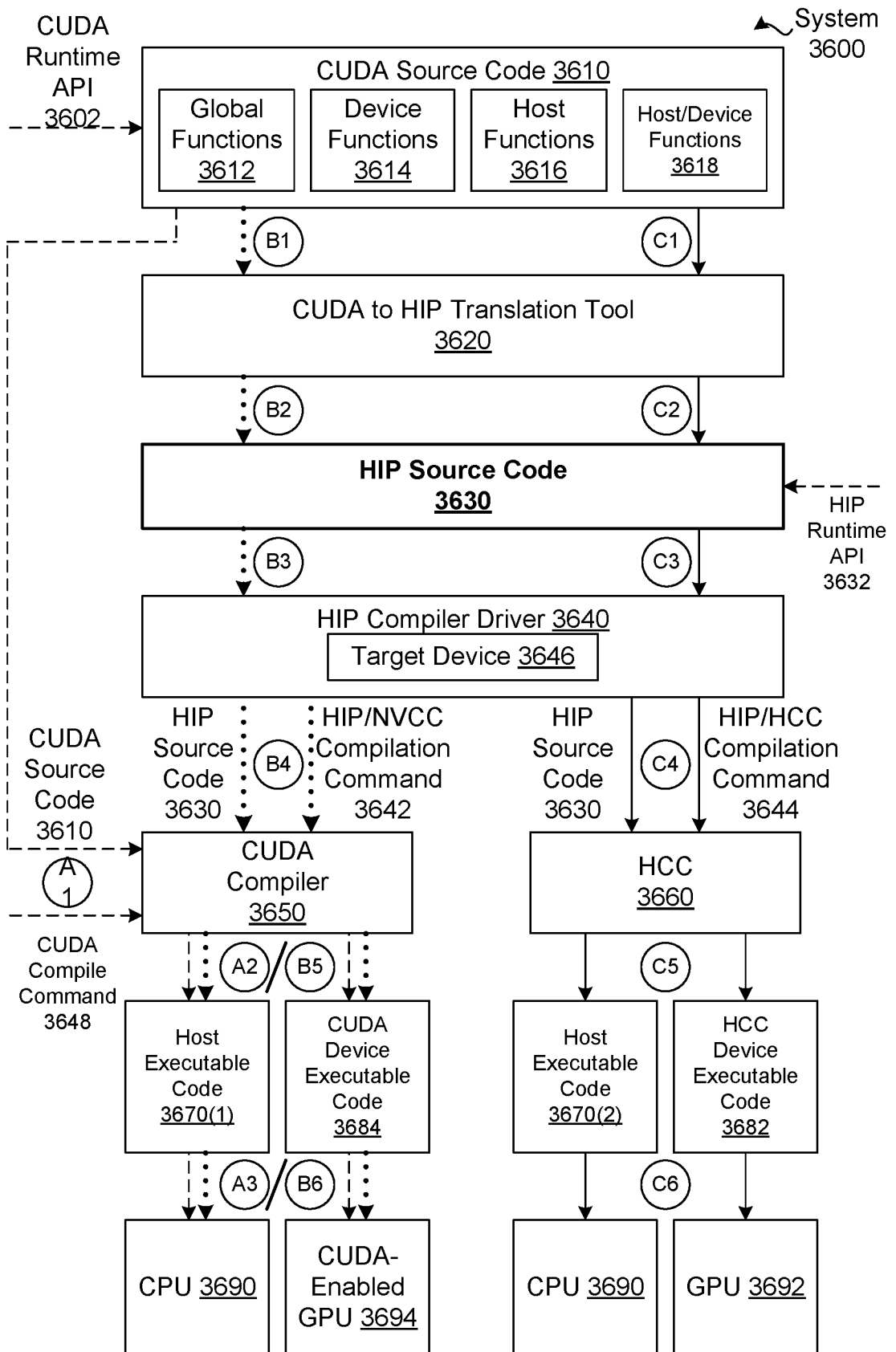
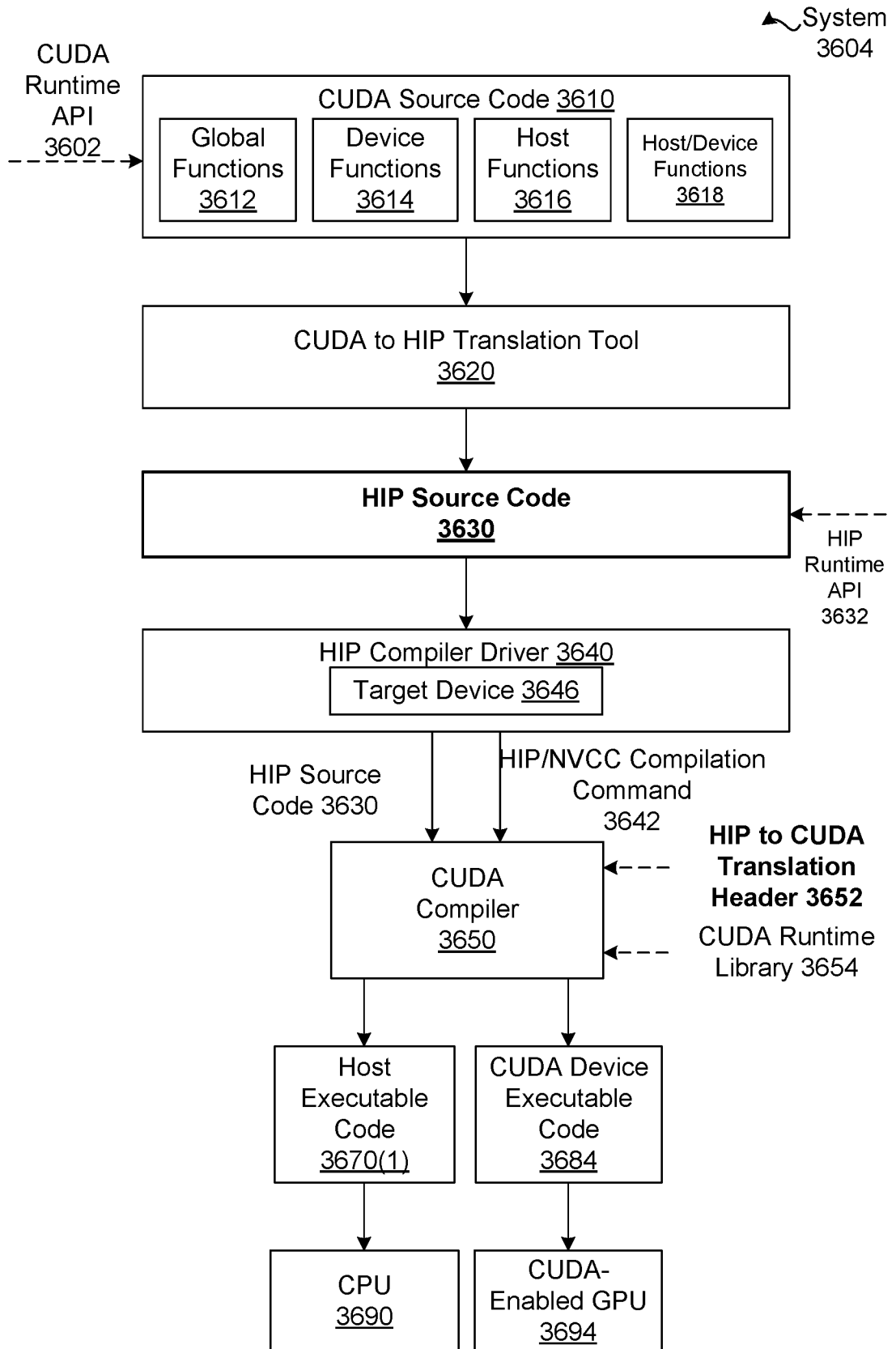


**FIG. 36A**



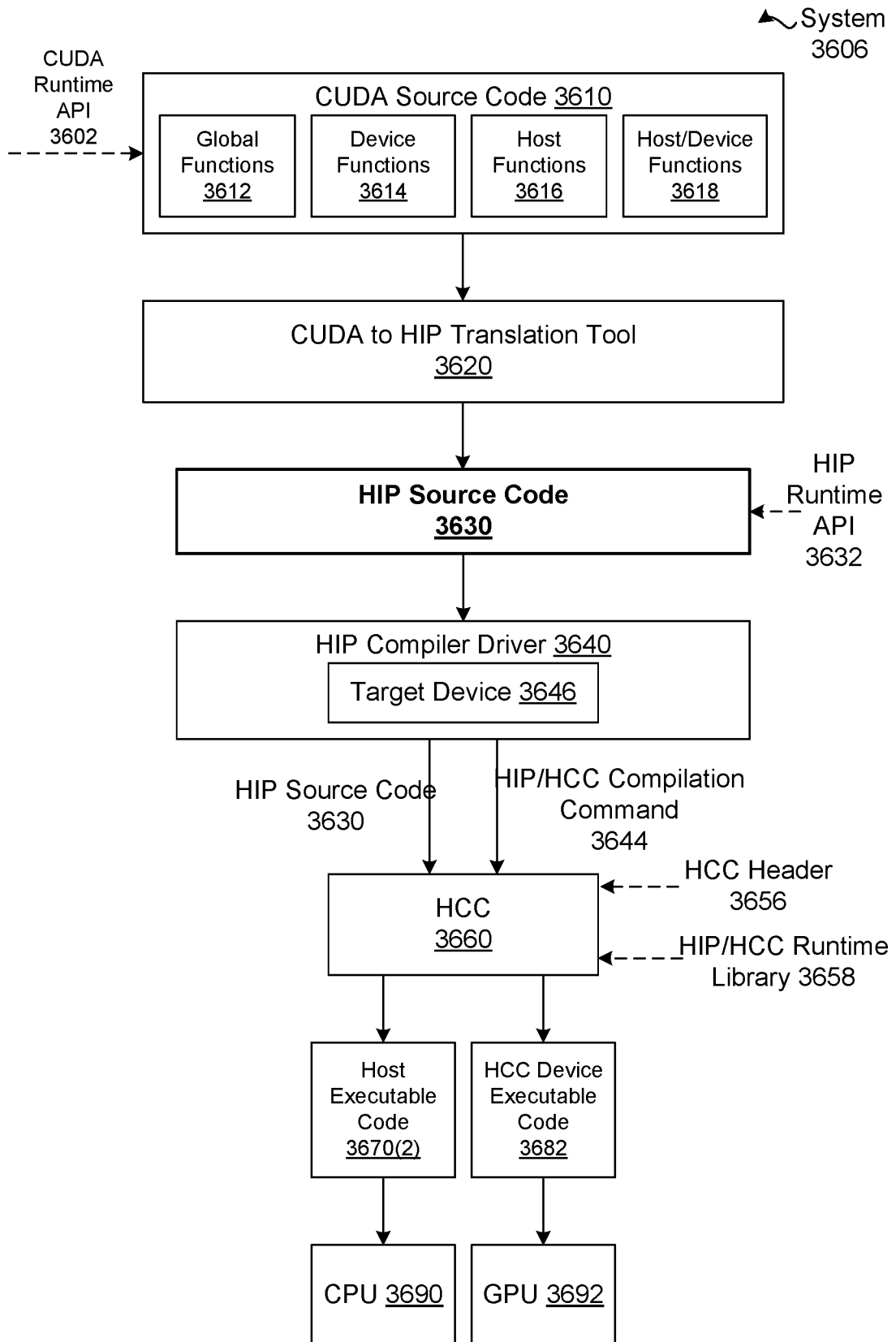


FIG. 36C

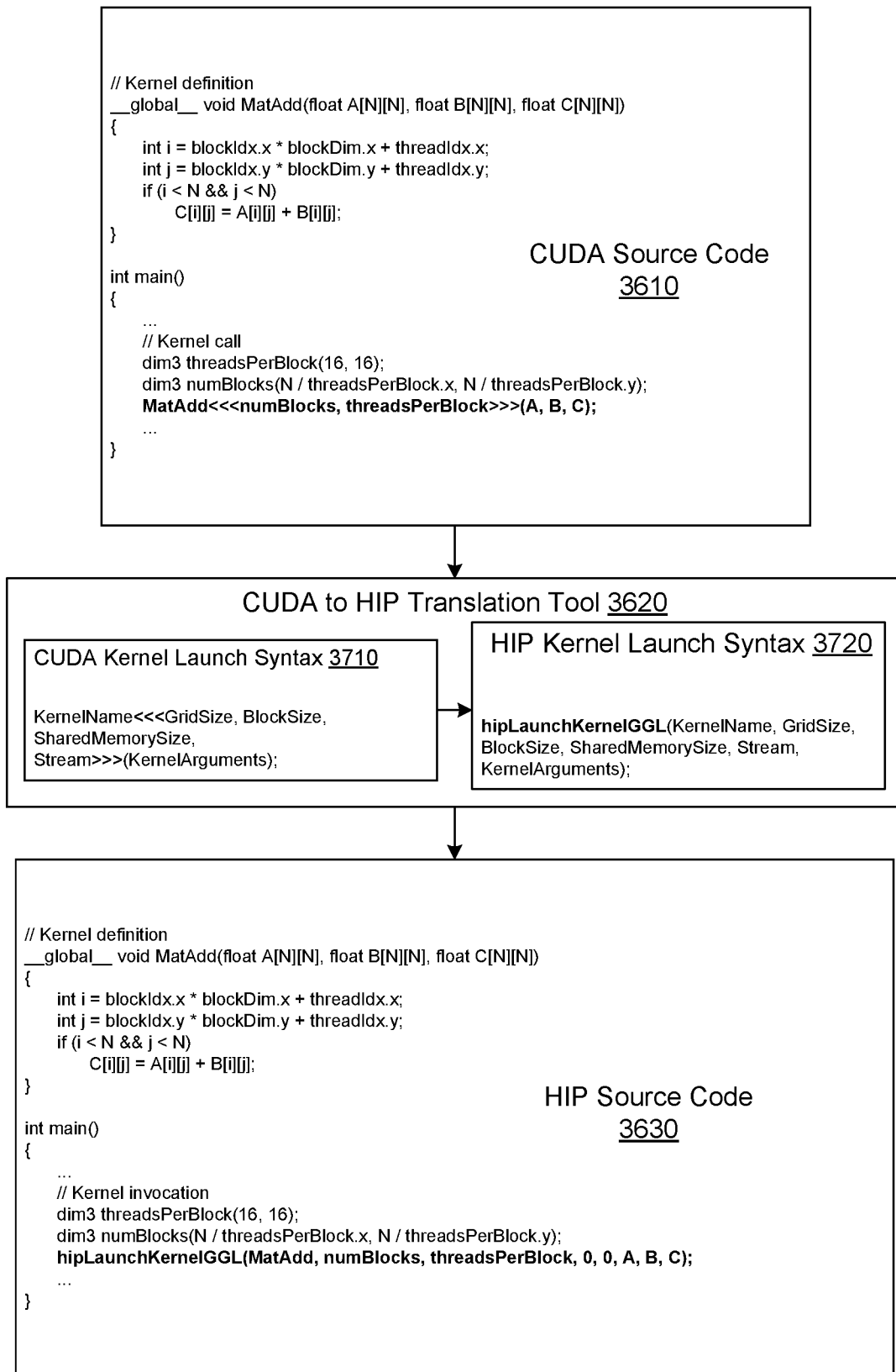


FIG. 37

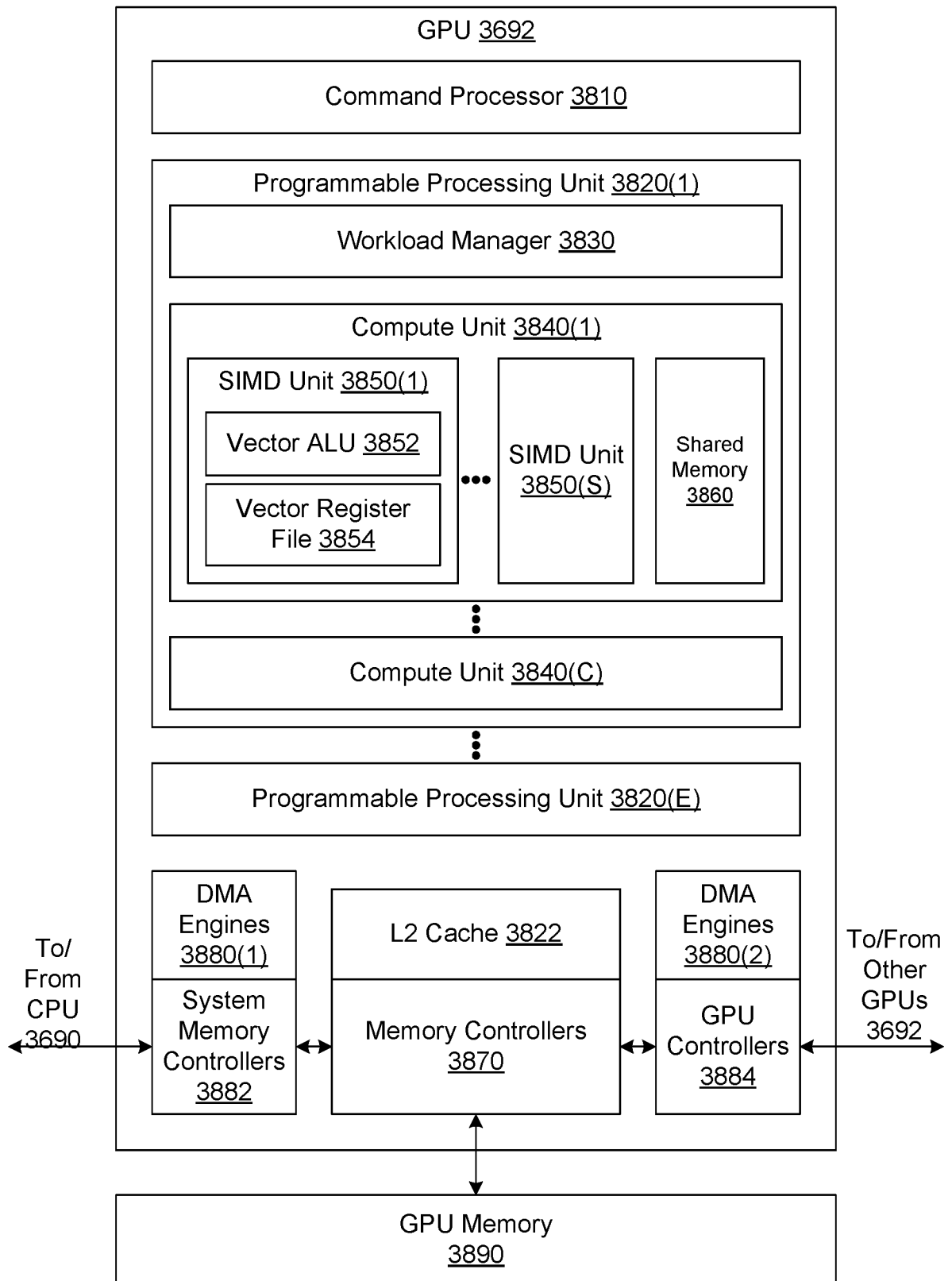


FIG. 38

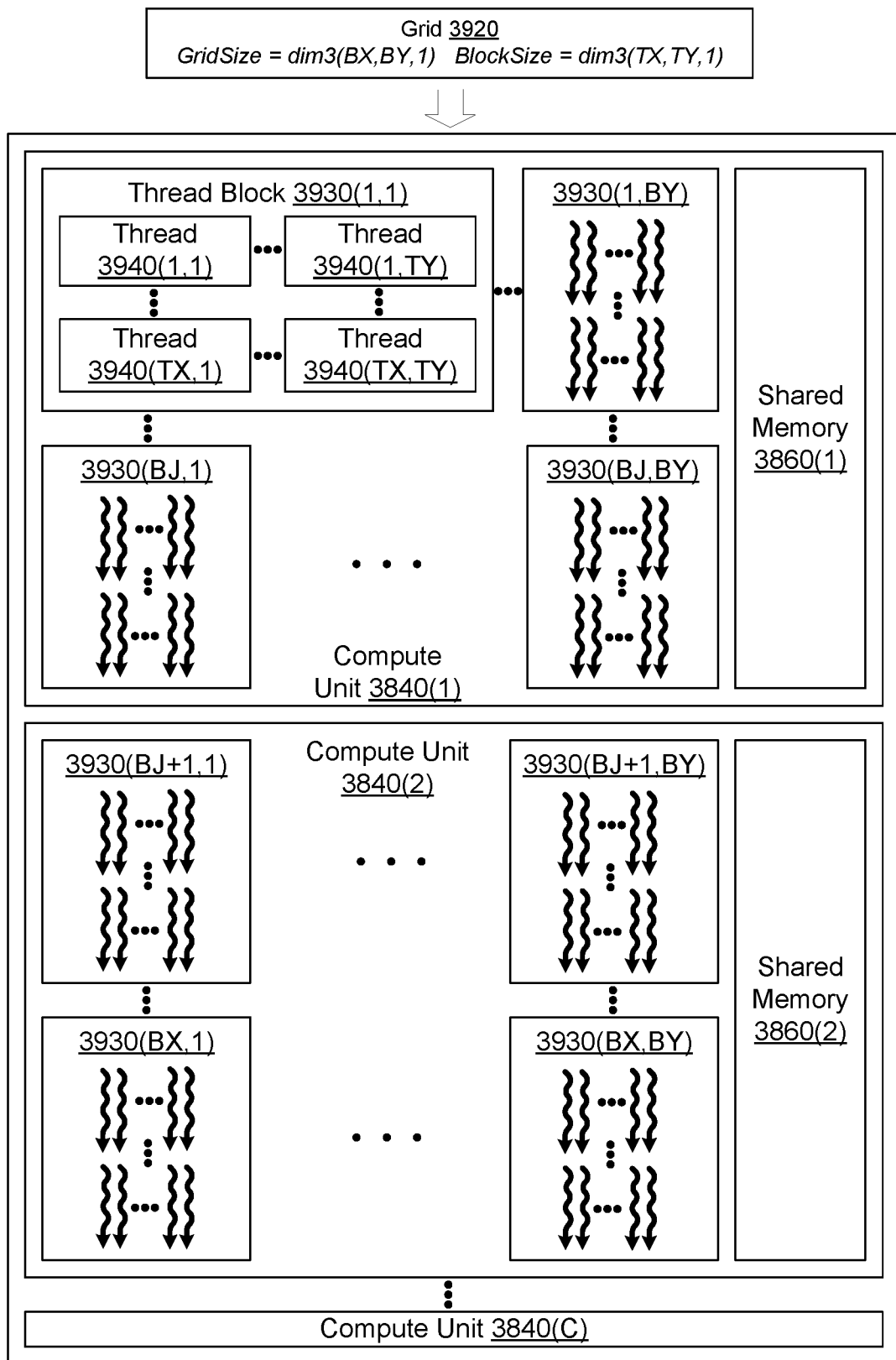


FIG. 39

Programmable Processing  
Unit 3820(1)

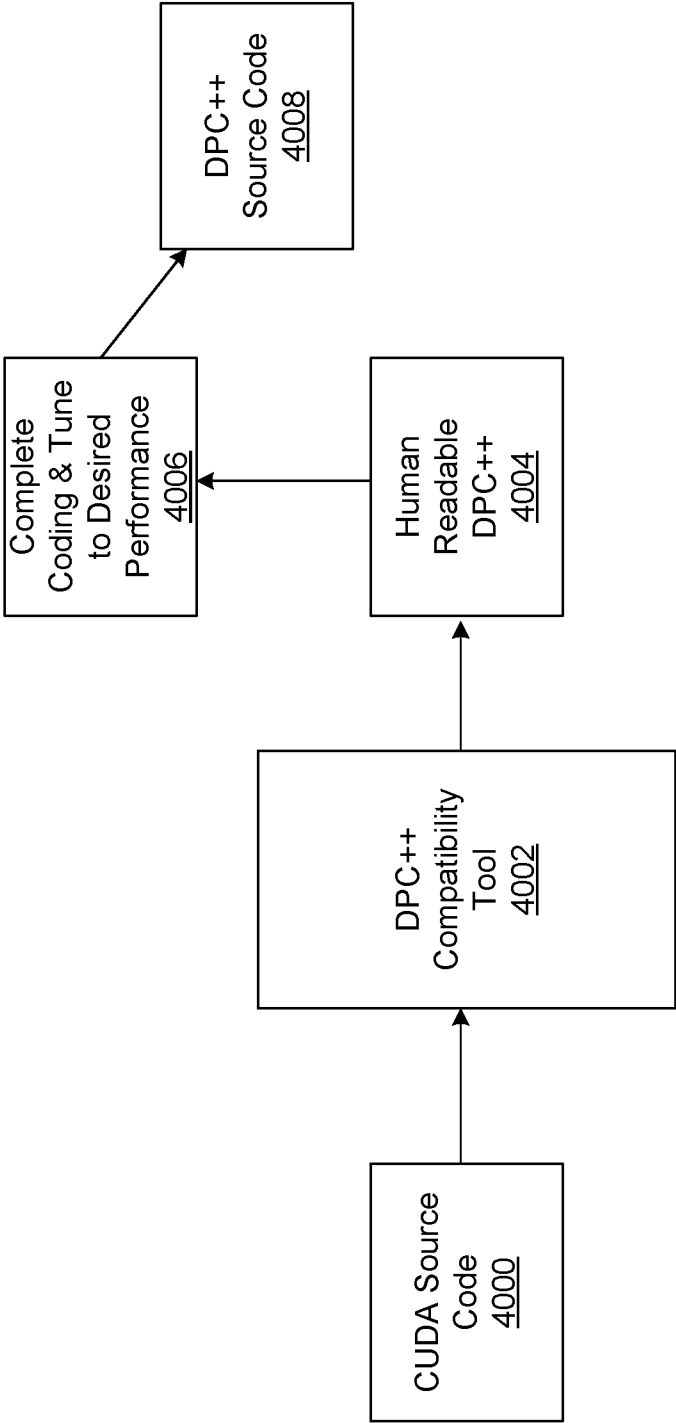


FIG. 40

# INTERNATIONAL SEARCH REPORT

International application No  
**PCT/US2021/045622**

## A. CLASSIFICATION OF SUBJECT MATTER

**INV.** G06N3/06 G06T1/20 G06T1/60  
**ADD.**

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

**G06N G06T**

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

**EPO-Internal, WPI Data**

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages                                 | Relevant to claim No.             |
|-----------|--------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| <b>X</b>  | <b>US 2020/184315 A1 (KIM SUNGHO [KR] ET AL)</b><br><b>11 June 2020 (2020-06-11)</b>                               | <b>1-15,</b><br><b>19-21</b>      |
| <b>Y</b>  | <b>paragraphs [0035], [0057] - [0060],</b><br><b>[0082], [0104] - [0106]; figures 3, 4, 7,</b><br><b>8, 10, 11</b> | <b>16-18</b>                      |
|           | -----                                                                                                              |                                   |
| <b>X</b>  | <b>WO 2018/184222 A1 (INTEL CORP [US]; GUO</b><br><b>YIWEN [CN] ET AL.)</b><br><b>11 October 2018 (2018-10-11)</b> | <b>1-8,14,</b><br><b>15,19-21</b> |
| <b>Y</b>  | <b>figures 4A, 9A</b>                                                                                              | <b>16-18</b>                      |
|           | -----                                                                                                              |                                   |
| <b>X</b>  | <b>US 2020/126192 A1 (MUNKBERG CARL JACOB</b><br><b>[SE] ET AL) 23 April 2020 (2020-04-23)</b>                     | <b>1,8,14</b>                     |
| <b>Y</b>  | <b>paragraphs [0123], [0142]; claim 1;</b><br><b>figures 1B, 2A, 3</b>                                             | <b>16-18</b>                      |
|           | -----                                                                                                              |                                   |
|           | -/--                                                                                                               |                                   |



Further documents are listed in the continuation of Box C.



See patent family annex.

\* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

**21 October 2021**

Date of mailing of the international search report

**07/01/2022**

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040,  
Fax: (+31-70) 340-3016

Authorized officer

**Trimeche, Mejdi**

# INTERNATIONAL SEARCH REPORT

International application No  
PCT/US2021/045622

## C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|-----------|------------------------------------------------------------------------------------|-----------------------|
| X         | US 2018/293713 A1 (VOGELS THIJIS [CH] ET AL) 11 October 2018 (2018-10-11)          | 1, 8, 14              |
| Y         | paragraphs [0064], [0238]; claim 1;<br>figures 3, 4A, 4B, 5, 13, 17<br>-----       | 16-18                 |

## INTERNATIONAL SEARCH REPORT

International application No.  
**PCT/US2021/045622**

### Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:  
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:  
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
3. ☐ Claims Nos.:  
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

### Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

**see additional sheet**

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims;; it is covered by claims Nos.:

**1-21**

#### Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- ☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- ☐ No protest accompanied the payment of additional search fees.

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2021/045622

| Patent document<br>cited in search report | Publication<br>date | Patent family<br>member(s)                                                                       | Publication<br>date                                                |
|-------------------------------------------|---------------------|--------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| US 2020184315 A1                          | 11-06-2020          | KR 20200069901 A<br>US 2020184315 A1                                                             | 17-06-2020<br>11-06-2020                                           |
| WO 2018184222 A1                          | 11-10-2018          | CN 110352432 A<br>EP 3607495 A1<br>US 2020026988 A1<br>WO 2018184222 A1                          | 18-10-2019<br>12-02-2020<br>23-01-2020<br>11-10-2018               |
| US 2020126192 A1                          | 23-04-2020          | NONE                                                                                             |                                                                    |
| US 2018293713 A1                          | 11-10-2018          | US 2018293496 A1<br>US 2018293712 A1<br>US 2018293713 A1<br>US 2020143522 A1<br>US 2020184605 A1 | 11-10-2018<br>11-10-2018<br>11-10-2018<br>07-05-2020<br>11-06-2020 |

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-21

A processor and method for implementing a neural network to generate a version of an image based on three-dimensional representation of image

---

2. claims: 22-42

A multi-processor core for training a neural network and inferring the result of the neural network

---