



US01238889B2

(12) **United States Patent**  
Sundareson et al.

(10) **Patent No.:** US 12,388,889 B2  
(45) **Date of Patent:** Aug. 12, 2025

(54) **AUGMENTING MULTIMEDIA STREAMING BY ANTICIPATING EVENTS USING ARTIFICIAL INTELLIGENCE**(71) Applicant: **NVIDIA Corporation**, Santa Clara, CA (US)(72) Inventors: **Prabindh Sundareson**, Bangalore (IN); **Sachin Dattatray Pandhare**, Bangalore (IN); **Shyam Raikar**, Pune (IN); **Toshant Sharma**, Amritsar (IN); **Prakash Yadav**, Pune (IN)(73) Assignee: **NVIDIA CORPORATION**, Santa Clara, CA (US)

(\* ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **17/512,590**(22) Filed: **Oct. 27, 2021**(65) **Prior Publication Data**

US 2023/0128243 A1 Apr. 27, 2023

(51) **Int. Cl.**

**H04L 65/75** (2022.01)  
**A63F 13/35** (2014.01)  
**A63F 13/85** (2014.01)  
**G06N 20/00** (2019.01)  
**H04L 65/65** (2022.01)  
**H04N 21/2187** (2011.01)

(52) **U.S. Cl.**

CPC ..... **H04L 65/764** (2022.05); **A63F 13/35** (2014.09); **A63F 13/85** (2014.09); **G06N 20/00** (2019.01); **H04L 65/65** (2022.05); **H04N 21/2187** (2013.01)

(58) **Field of Classification Search**

CPC ..... A63F 13/35; A63F 13/85; G06N 20/00; H04L 65/65; H04L 65/764; H04N 21/2187

See application file for complete search history.

(56) **References Cited**

## U.S. PATENT DOCUMENTS

|               |         |                   |
|---------------|---------|-------------------|
| 8,651,961 B2  | 2/2014  | Muller            |
| 9,088,823 B1  | 7/2015  | Price             |
| 9,224,156 B2  | 12/2015 | Moorer            |
| 9,242,176 B2  | 1/2016  | Miura et al.      |
| 9,345,966 B2  | 5/2016  | Miura et al.      |
| 9,364,743 B2  | 6/2016  | Miura et al.      |
| 10,737,183 B1 | 8/2020  | Sharma et al.     |
| 10,741,215 B1 | 8/2020  | Sundareson et al. |

(Continued)

## FOREIGN PATENT DOCUMENTS

|    |               |         |
|----|---------------|---------|
| CN | 113318448 A   | 8/2021  |
| KR | 20200120120 A | 10/2020 |
| WO | 2018140515 A1 | 8/2018  |

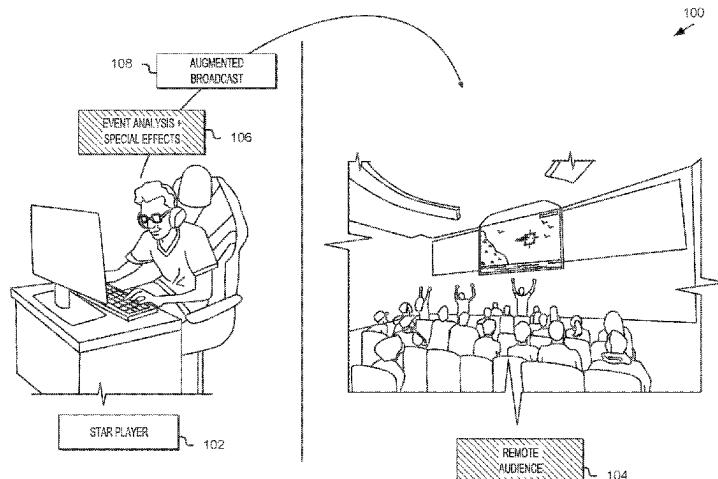
## OTHER PUBLICATIONS

Andy, "GoPro Time Lapse: Beginners Guide (Step by Step) Video Tutorial," retrieved from <https://projectgo.pro/gopro-time-lapse/>, 2019, 40 pages.

(Continued)

*Primary Examiner* — Thomas J Dailey(74) *Attorney, Agent, or Firm* — Davis Wright Tremaine LLP(57) **ABSTRACT**

Apparatuses, systems, and techniques to predict events within a gameplay session and modify a game broadcast based at least in part on the predicted events. In at least one embodiment, events within the gameplay session are predicted, which once predicted, cause assets to be generated and once the event is detected the assets are included in the game broadcast.

**28 Claims, 58 Drawing Sheets**

(56)

**References Cited**

## U.S. PATENT DOCUMENTS

|                   |         |                          |                            |
|-------------------|---------|--------------------------|----------------------------|
| 11,058,948 B1 *   | 7/2021  | Hardiman .....           | A63F 13/798                |
| 2005/0066279 A1 * | 3/2005  | LeBarton .....           | G11B 27/10<br>715/725      |
| 2006/0059225 A1   | 3/2006  | Stonehocker et al.       |                            |
| 2008/0138029 A1 * | 6/2008  | Xu .....                 | G06F 16/7854<br>348/E5.022 |
| 2010/0035689 A1   | 2/2010  | Altshuler et al.         |                            |
| 2012/0100910 A1   | 4/2012  | Eichorn et al.           |                            |
| 2015/0024839 A1   | 1/2015  | Zahn et al.              |                            |
| 2015/0202529 A1   | 7/2015  | Paradise et al.          |                            |
| 2015/0217196 A1   | 8/2015  | McCarthy et al.          |                            |
| 2015/0222239 A1   | 8/2015  | Zhang et al.             |                            |
| 2015/0347918 A1 * | 12/2015 | Lucey .....              | G06N 20/00<br>706/12       |
| 2016/0158656 A1   | 6/2016  | Condrey                  |                            |
| 2017/0065888 A1   | 3/2017  | Cheng                    |                            |
| 2017/0065889 A1   | 3/2017  | Cheng                    |                            |
| 2017/0113143 A1   | 4/2017  | Marr et al.              |                            |
| 2017/0136367 A1   | 5/2017  | Watari et al.            |                            |
| 2017/0140570 A1   | 5/2017  | Leibel et al.            |                            |
| 2017/0228600 A1   | 8/2017  | Syed et al.              |                            |
| 2017/0282077 A1   | 10/2017 | De La Cruz               |                            |
| 2017/0294081 A1   | 10/2017 | Washington et al.        |                            |
| 2018/0078861 A1   | 3/2018  | Borggren                 |                            |
| 2018/0078862 A1   | 3/2018  | Schleicher               |                            |
| 2018/0140515 A1   | 5/2018  | Perez Nowak et al.       |                            |
| 2018/0161675 A1   | 6/2018  | Miron et al.             |                            |
| 2019/0099668 A1   | 4/2019  | Aliakseyeu et al.        |                            |
| 2019/0111347 A1   | 4/2019  | Rimon                    |                            |
| 2019/0313024 A1   | 10/2019 | Selinger et al.          |                            |
| 2019/0373315 A1 * | 12/2019 | Song .....               | H04N 21/44008              |
| 2020/0078679 A1   | 3/2020  | Dantas de Castro         |                            |
| 2020/0175303 A1   | 6/2020  | Bhat et al.              |                            |
| 2020/0304885 A1   | 9/2020  | Ickin                    |                            |
| 2020/0336668 A1 * | 10/2020 | Hutsler .....            | H04N 23/695                |
| 2020/0346121 A1   | 11/2020 | Beaumont et al.          |                            |
| 2021/0146241 A1   | 5/2021  | Blesdale-Shepherd et al. |                            |
| 2021/0299575 A1   | 9/2021  | Stafford et al.          |                            |
| 2021/0329352 A1   | 10/2021 | Thomas et al.            |                            |

## OTHER PUBLICATIONS

- Bernardi et al., "A Time Series Classification Approach to Game Bot Detection," Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, ACM, 2017, 11 pages.
- Cheong et al., "Automatically Generating Summary Visualizations from Game Logs," Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Oct. 24, 2008, 6 pages.
- Guo, "Localization and Recognition of the Scoreboard in Sports Video Based on SIFT Point Matching," Springer-Verlag Berlin Heidelberg, 2011, 11 pages.
- Guérin et al., "CNN Features are Also Great at Unsupervised Classification," Jul. 6, 2017, 10 pages.
- Harvey, "Five Video Classification Methods," retrieved from <https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5>, Mar. 22, 2017, 15 pages.

IEEE "IEEE Standard for Floating-Point Arithmetic", Microprocessor Standards Committee of the IEEE Computer Society, IEEE Std 754-2008, dated Jun. 12, 2008, 70 pages.

International Search Report and Written Opinion for Application No. PCT/US2019/053647, mailed Feb. 27, 2020, 11 pages.

International Search Report and Written Opinion for Application No. PCT/US2021/013632, mailed Mar. 30, 2021, 12 pages.

Kaminsky et al., "Identifying Game Players with Mouse Biometrics," University of Washington, Technical Report, 2008, 13 pages.

Limmen "What Influences Keyboard Input Speed" retrieved from <https://blog.wooting.nl/whatinfluences-keyboard-speed/>, May 31, 2018, 24 pages.

Lutz et al., "Efficient k-means on GPU" DaMon, Jun. 11, 2018, 3 pages.

Non-Final Office Action for U.S. Appl. No. 16/747,143, dated Mar. 11, 2021.

Notice of Allowance for U.S. Appl. No. 16/459,957 dated Jan. 27, 2021, 5 pages.

Notice of Allowance for U.S. Appl. No. 16/527,963 dated Jul. 16, 2020, 8 pages.

Platzer, "Sequence-based Bot Detection in Massive Multiplayer Online Games," 2011 8th International Conference on Information, Communications & Signal Processing, 2011, 5 pages.

Ringer et al., "Deep Unsupervised Multi-View Detection of Video Game Stream Highlights," Jul. 25, 2018, 6 pages.

Society of Automotive Engineers On-Road Automated Vehicle Standards Committee, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," Standard No. J3016-201609, issued Jan. 2014, revised Sep. 2016, 30 pages.

Society of Automotive Engineers On-Road Automated Vehicle Standards Committee, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," Standard No. J3016-201806, issued Jan. 2014, revised Jun. 2018, 35 pages.

Vailaya et al., "Video Clustering." Michigan State University, Technical Report, 1996, 8 pages.

XMP, "XMP Specification Part 3, Storage in Files," Adobe Systems Incorporated, <https://wwwimages2.adobe.com/content/dam/acom/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2016-08/XMPSpecificationPart3.pdf>, 2008, 86 pages.

YouTube "YouTube Player API Reference for iframe Embeds," [https://developers.google.com/youtube/frame\\_api\\_reference](https://developers.google.com/youtube/frame_api_reference), Mar. 26, 2012, 43 pages.

Gardiner, "How to Add Graphics and Overlays to Live Video With Open Broadcaster (OBS)," Retrieved from, <http://web.archive.org/web/20200928092901/https://photography.tutsplus.com/tutorials/how-to-add-custom-graphics-obs-studio-cms-35066>, Sep. 8, 2020, 12 Pages.

Rosenzveig, "Auto Generate Subtitles Based on Video Transcript," Retrieved from, [https://cloudinary.com/blog/with\\_automatic\\_video\\_subtitles\\_silence\\_speaks\\_volumes](https://cloudinary.com/blog/with_automatic_video_subtitles_silence_speaks_volumes), Jan. 17, 2018, 6 Pages.

Office Action for Chinese Application No. 202211303962.5, mailed Apr. 24, 2024, 31 pages.

Office Action for Chinese Application No. 202211303962.5, mailed Sep. 12, 2024, 33 pages.

Office Action for Chinese Application No. 202211303962.5, mailed Oct. 30, 2024, 33 pages.

\* cited by examiner

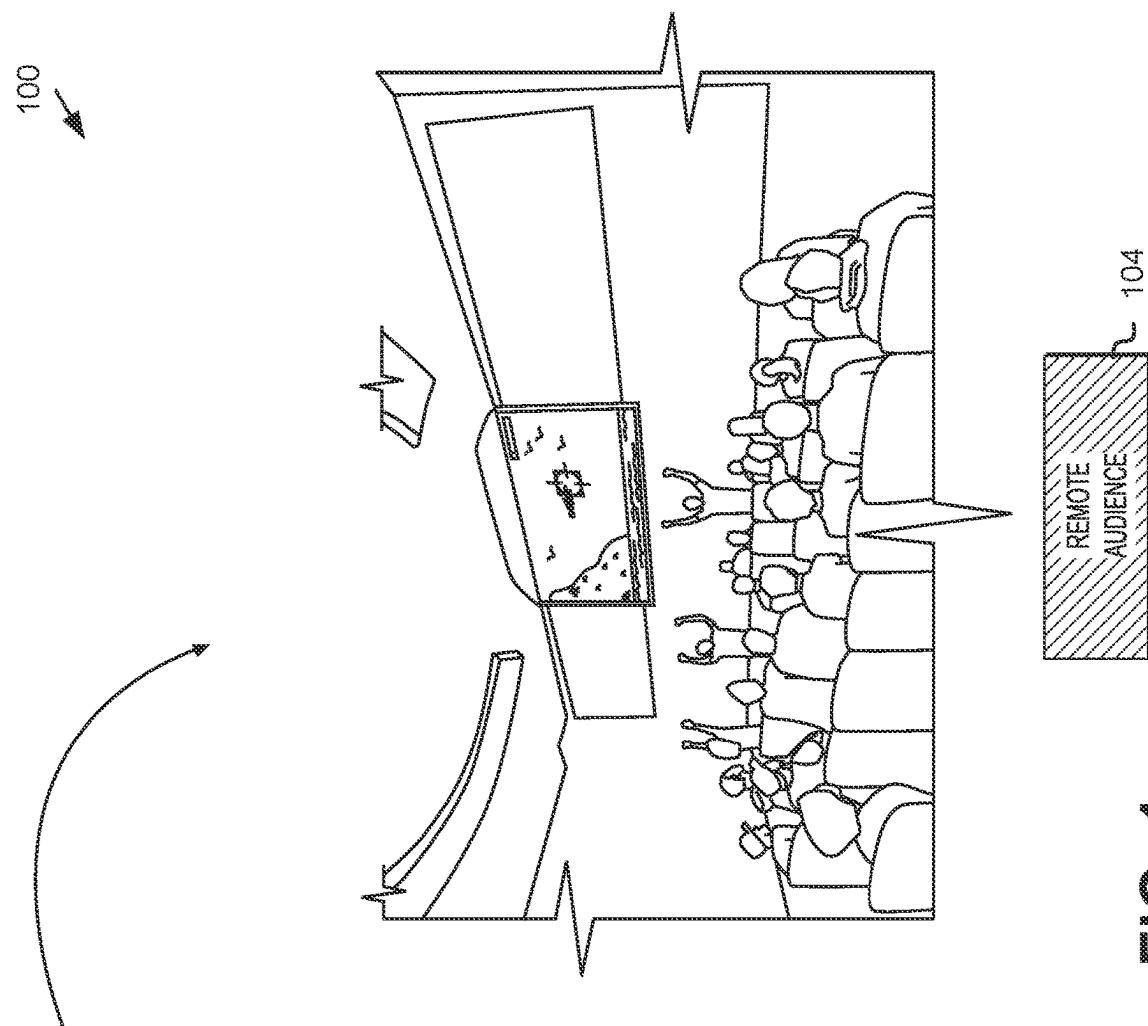
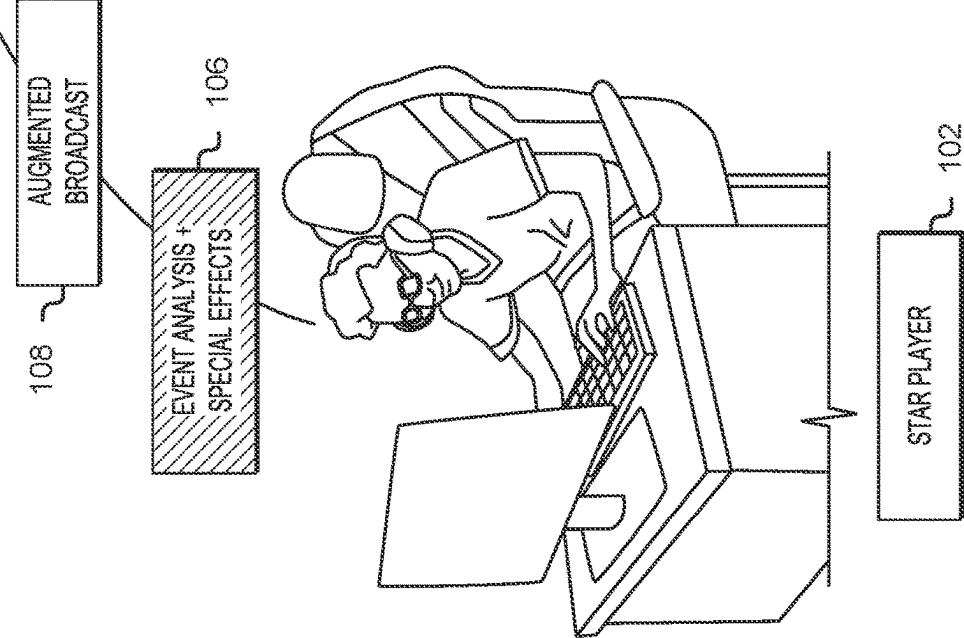


FIG. 1



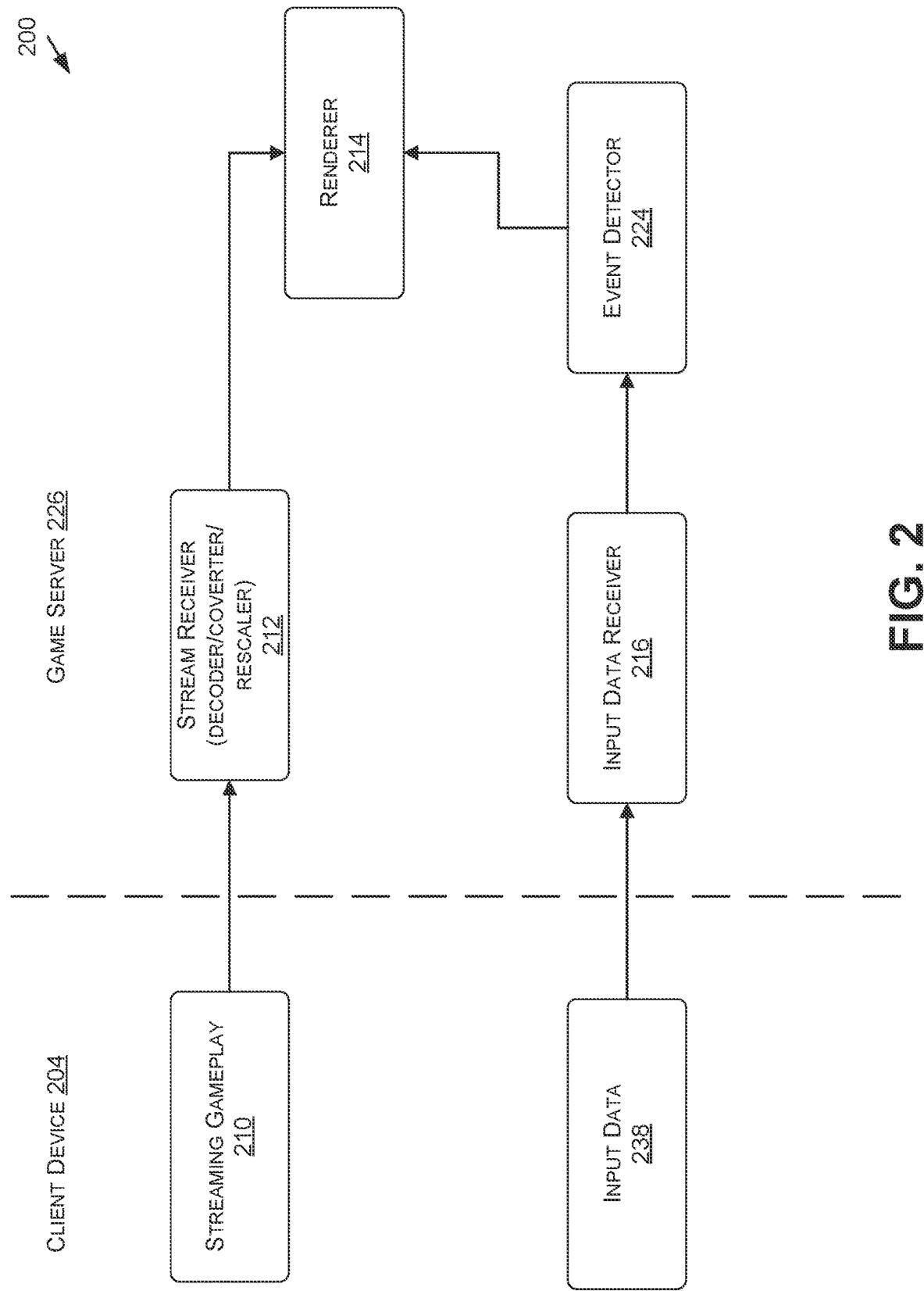
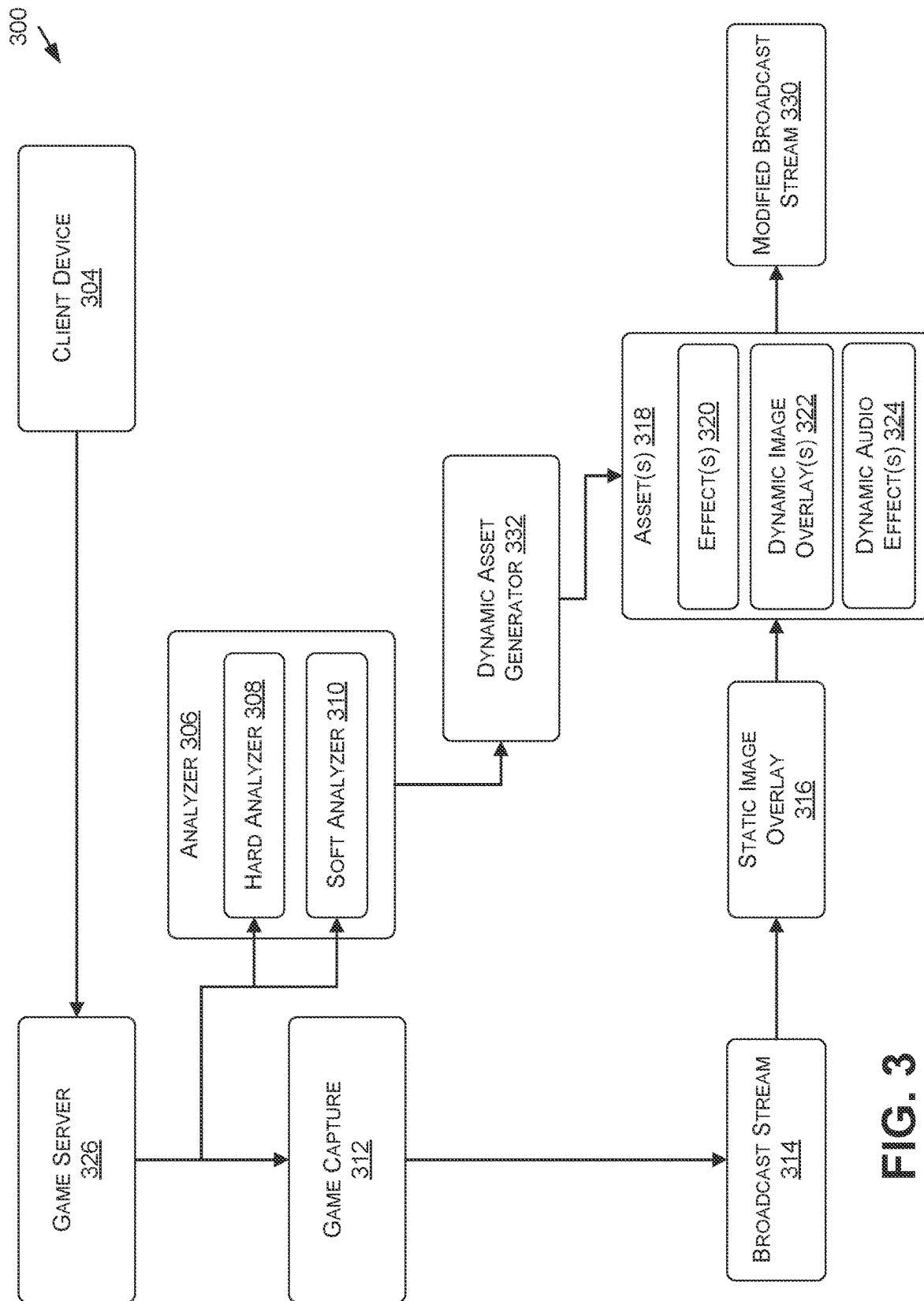


FIG. 2

**FIG. 3**

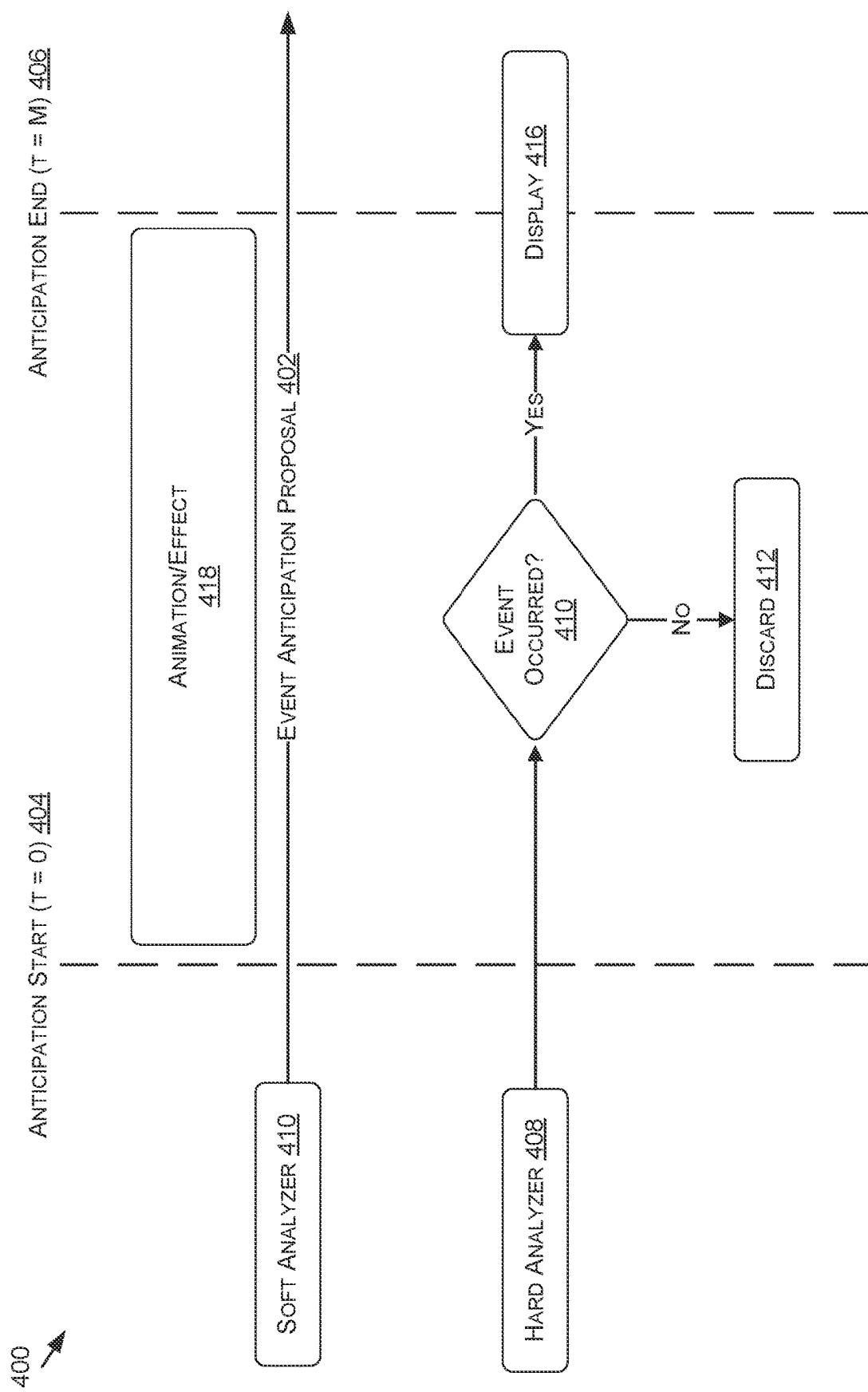


FIG. 4

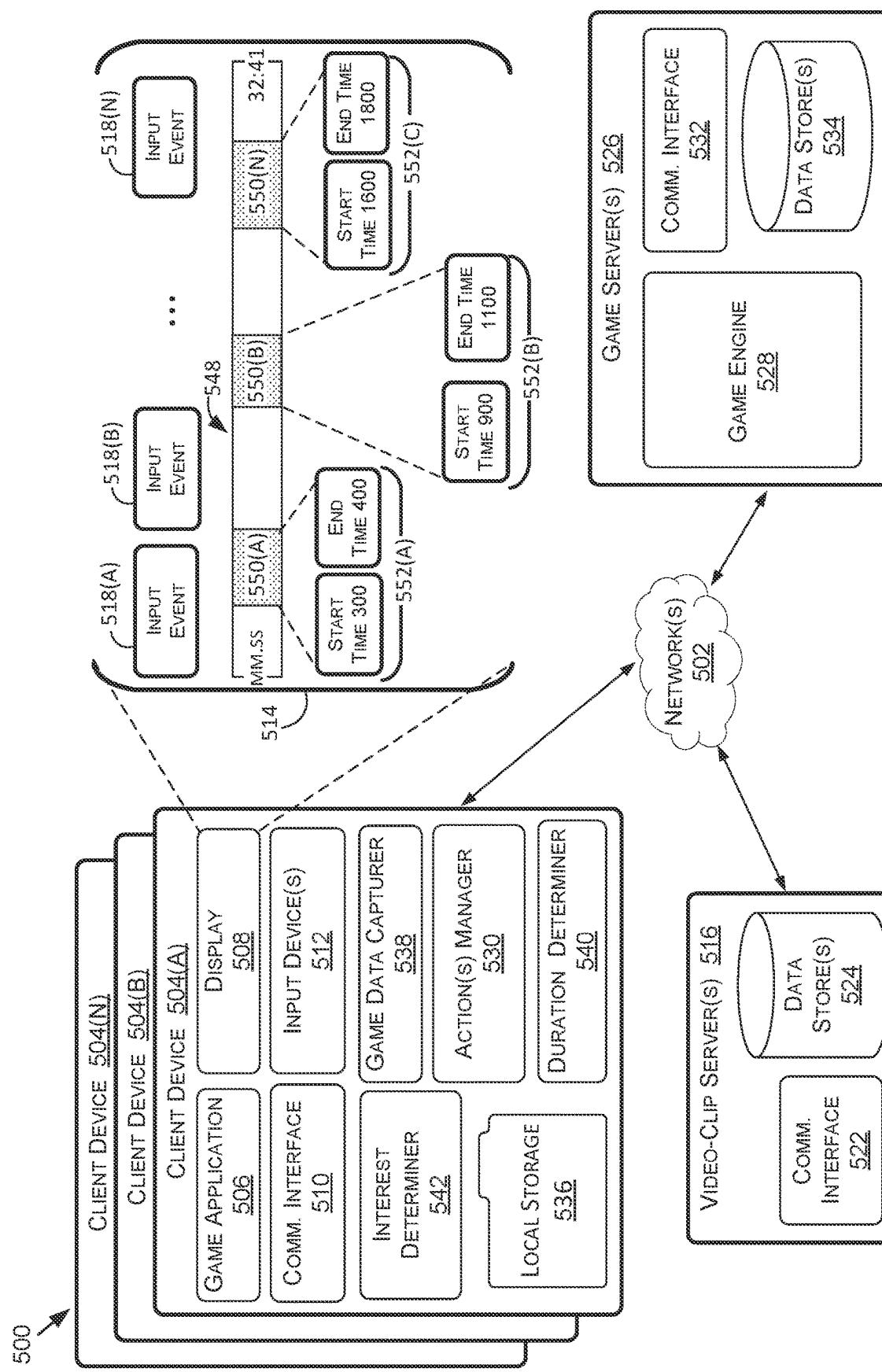


FIG. 5

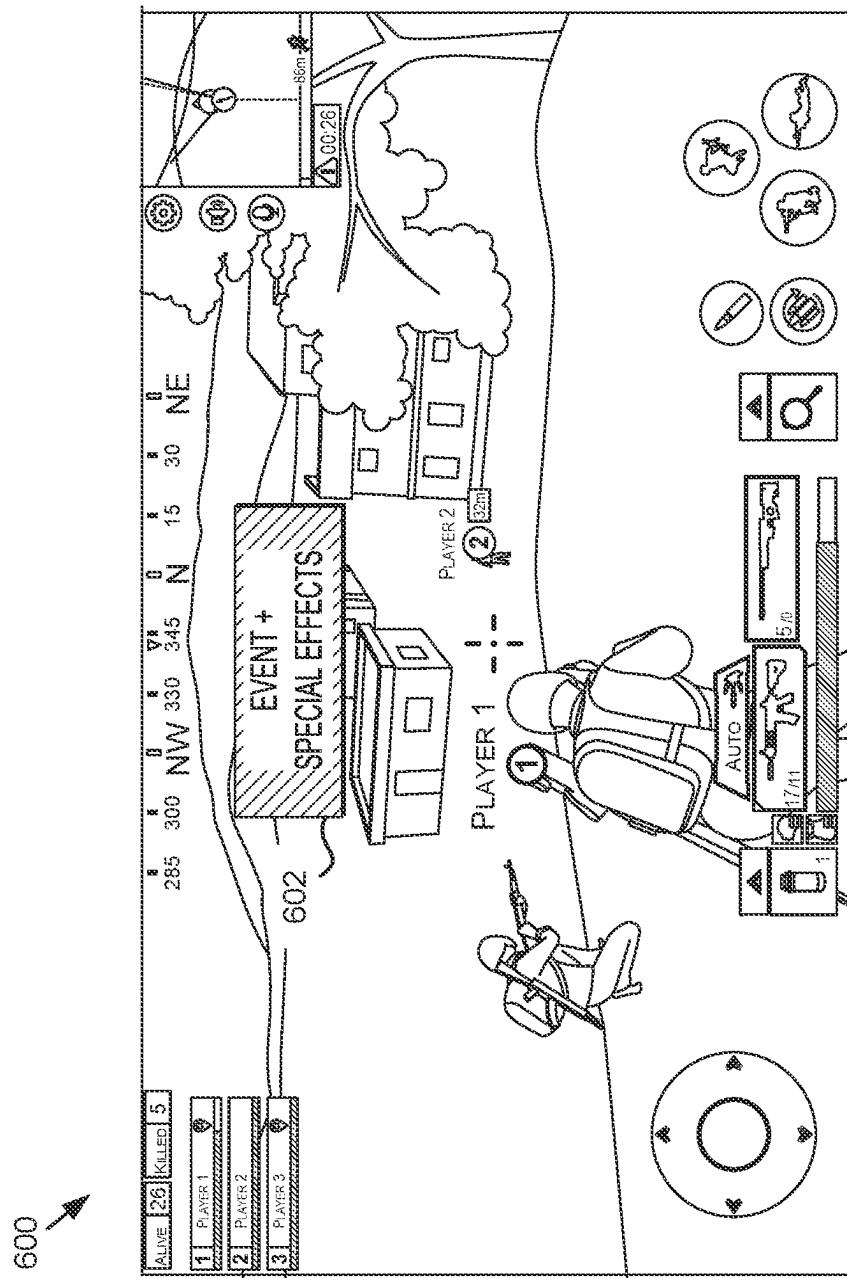


FIG. 6

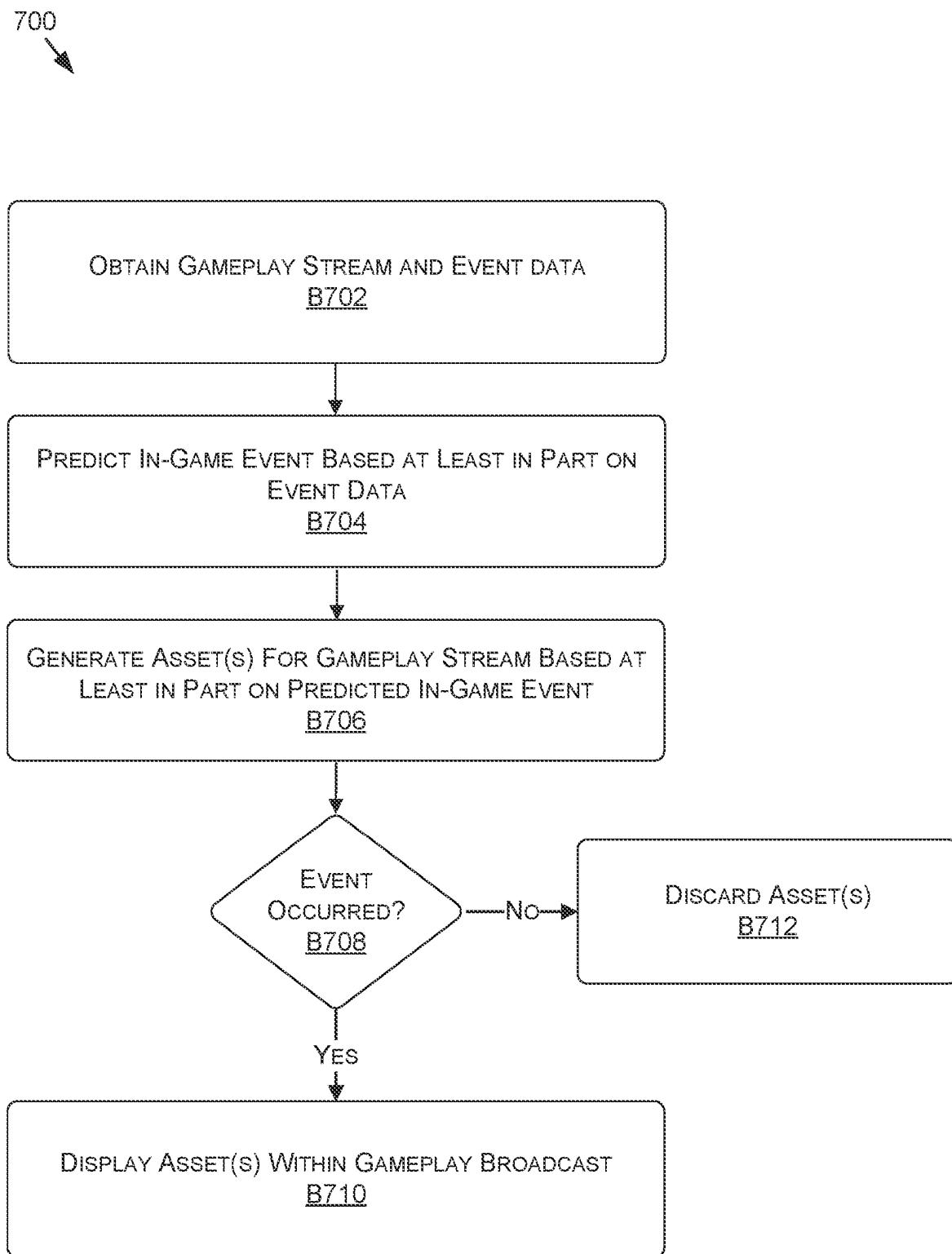


FIG. 7

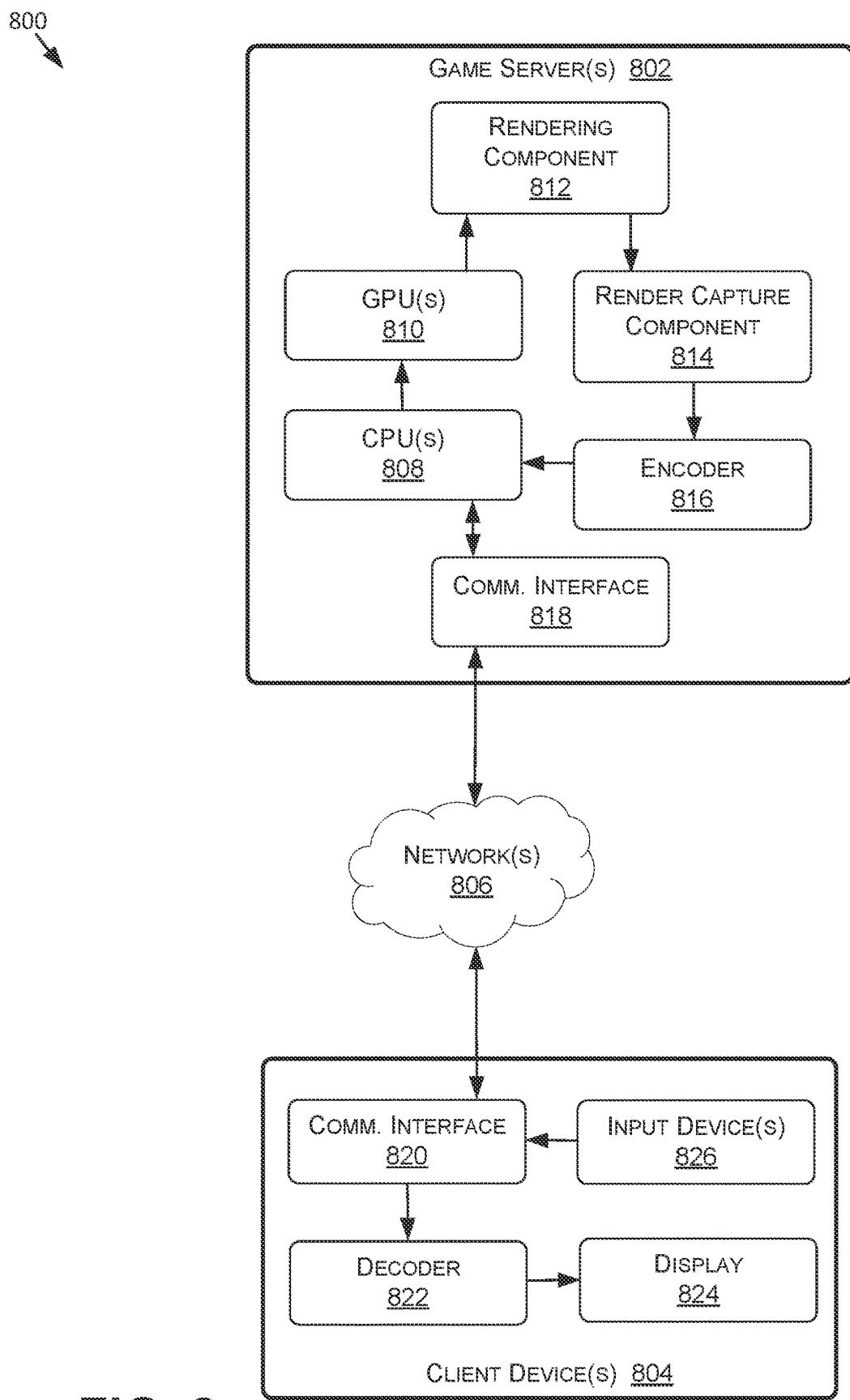


FIG. 8

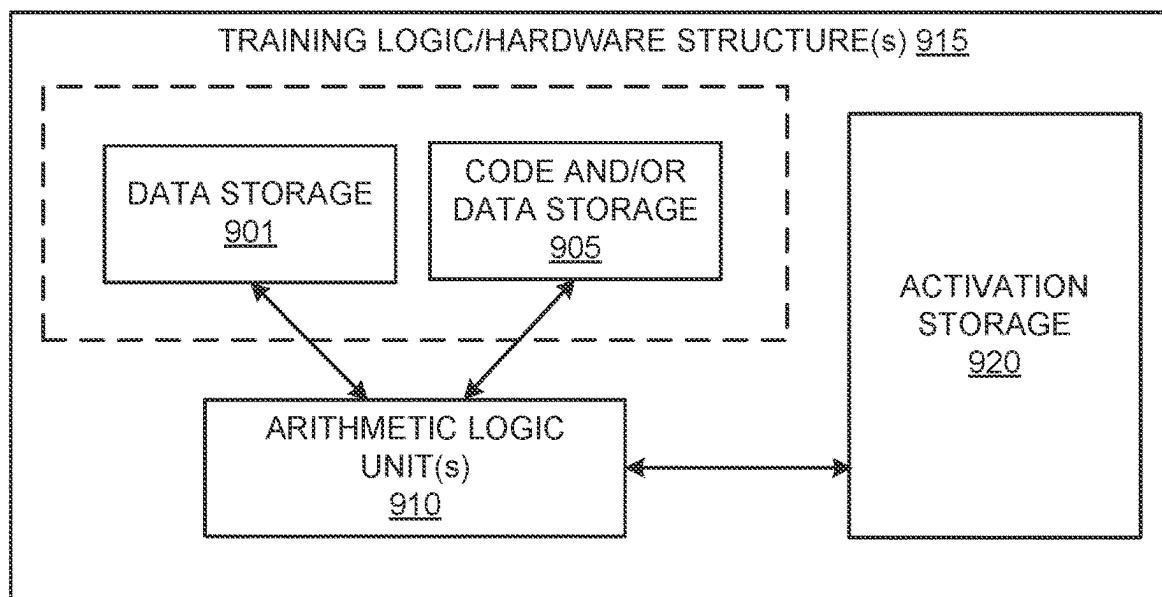


FIG. 9A

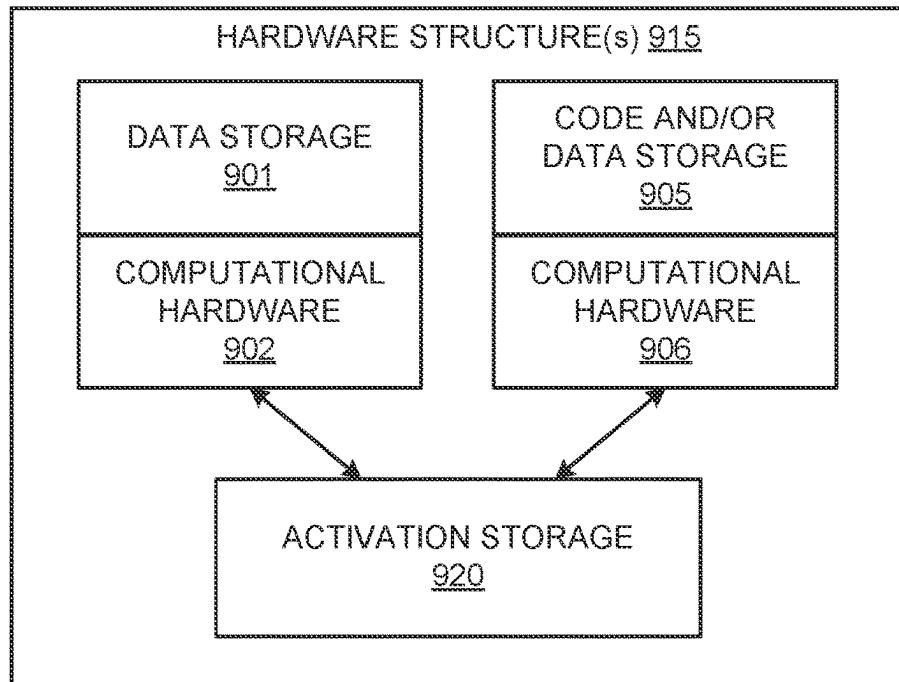
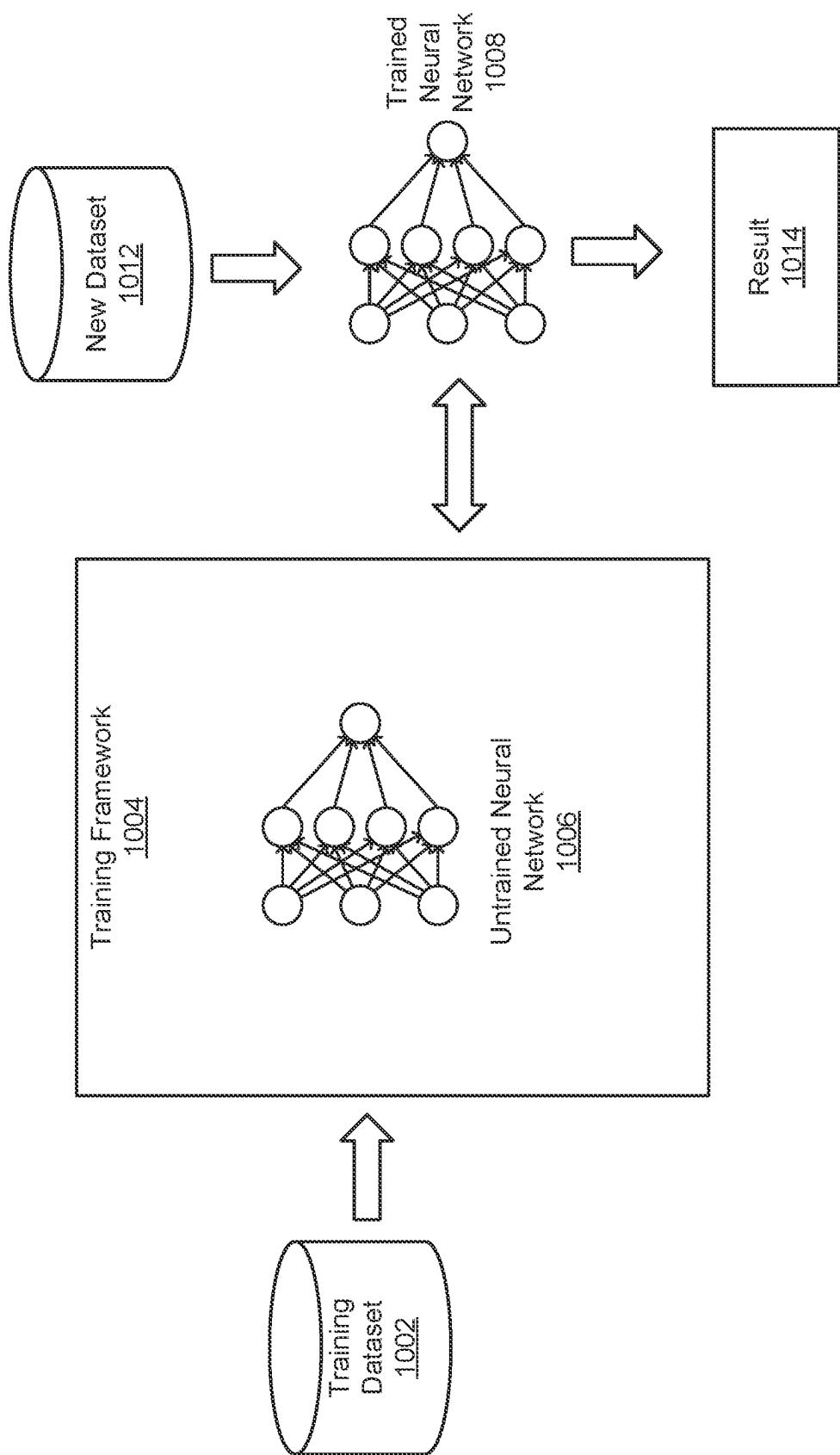


FIG. 9B

**FIG. 10**

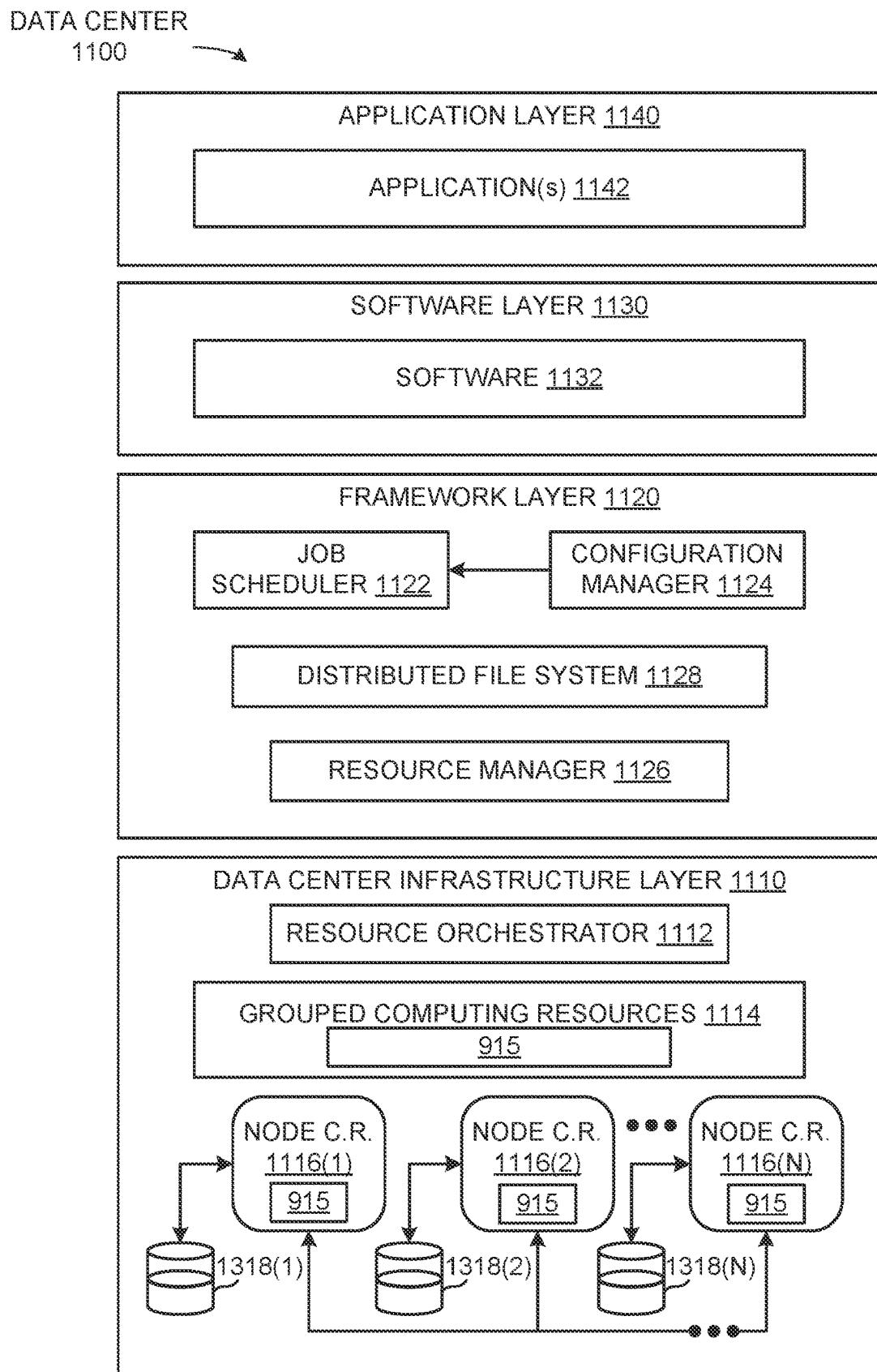


FIG. 11

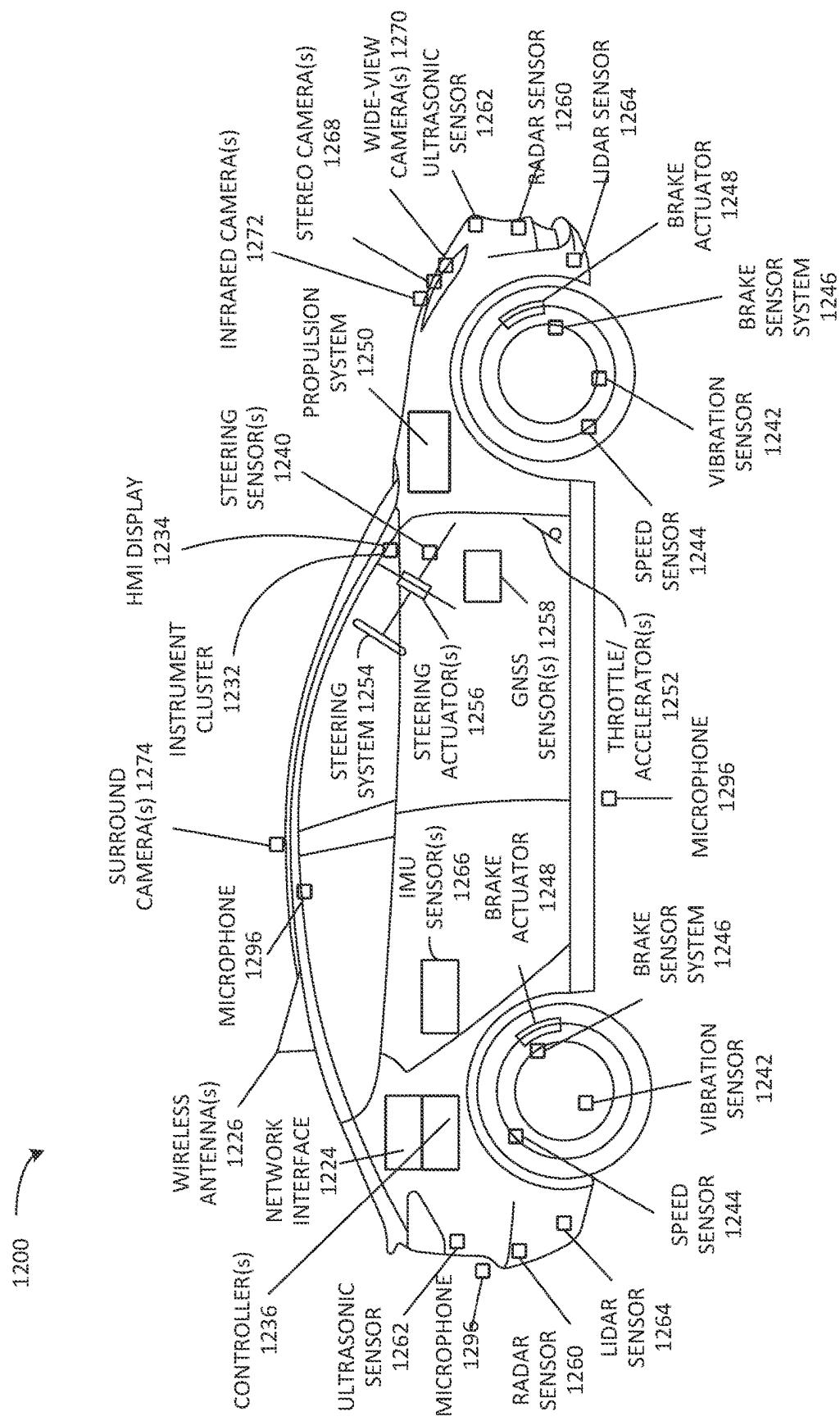


FIG. 12 A

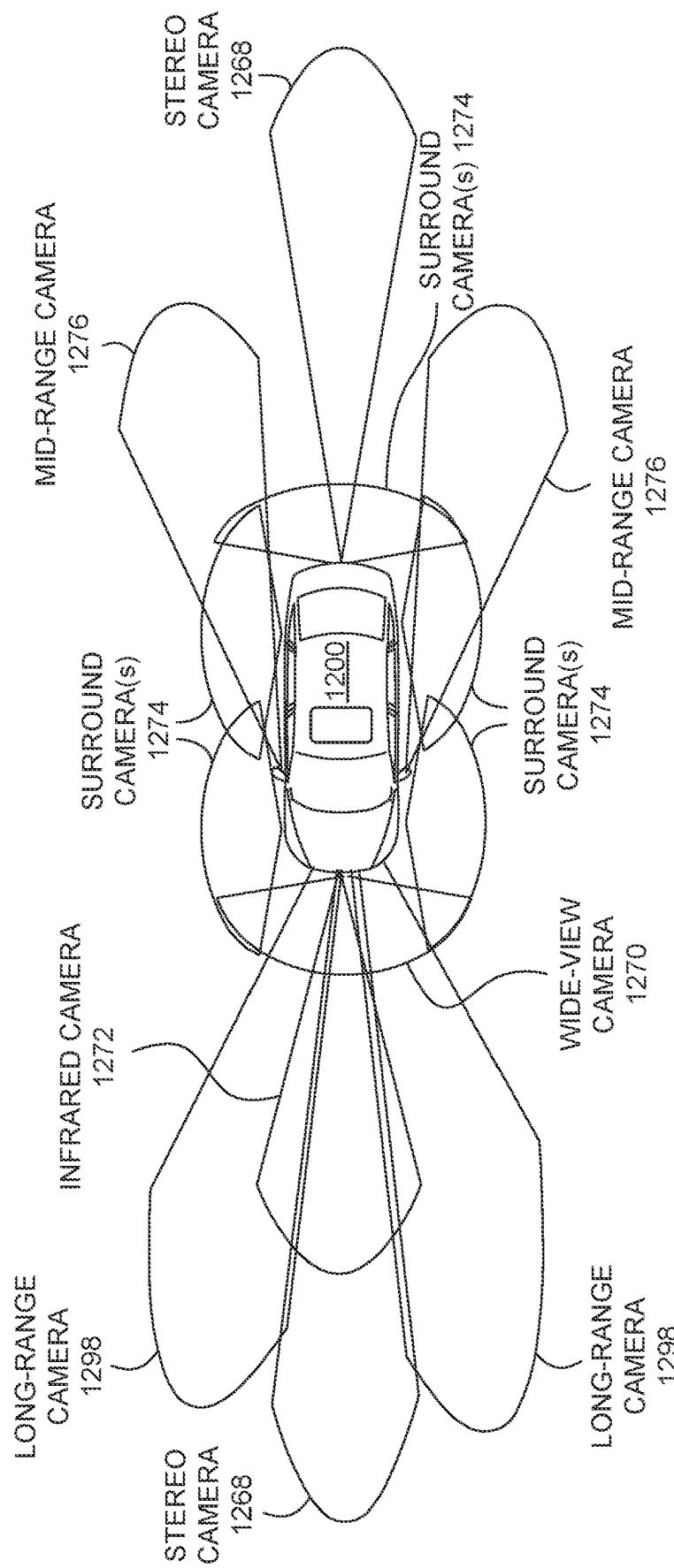


FIG. 12 B

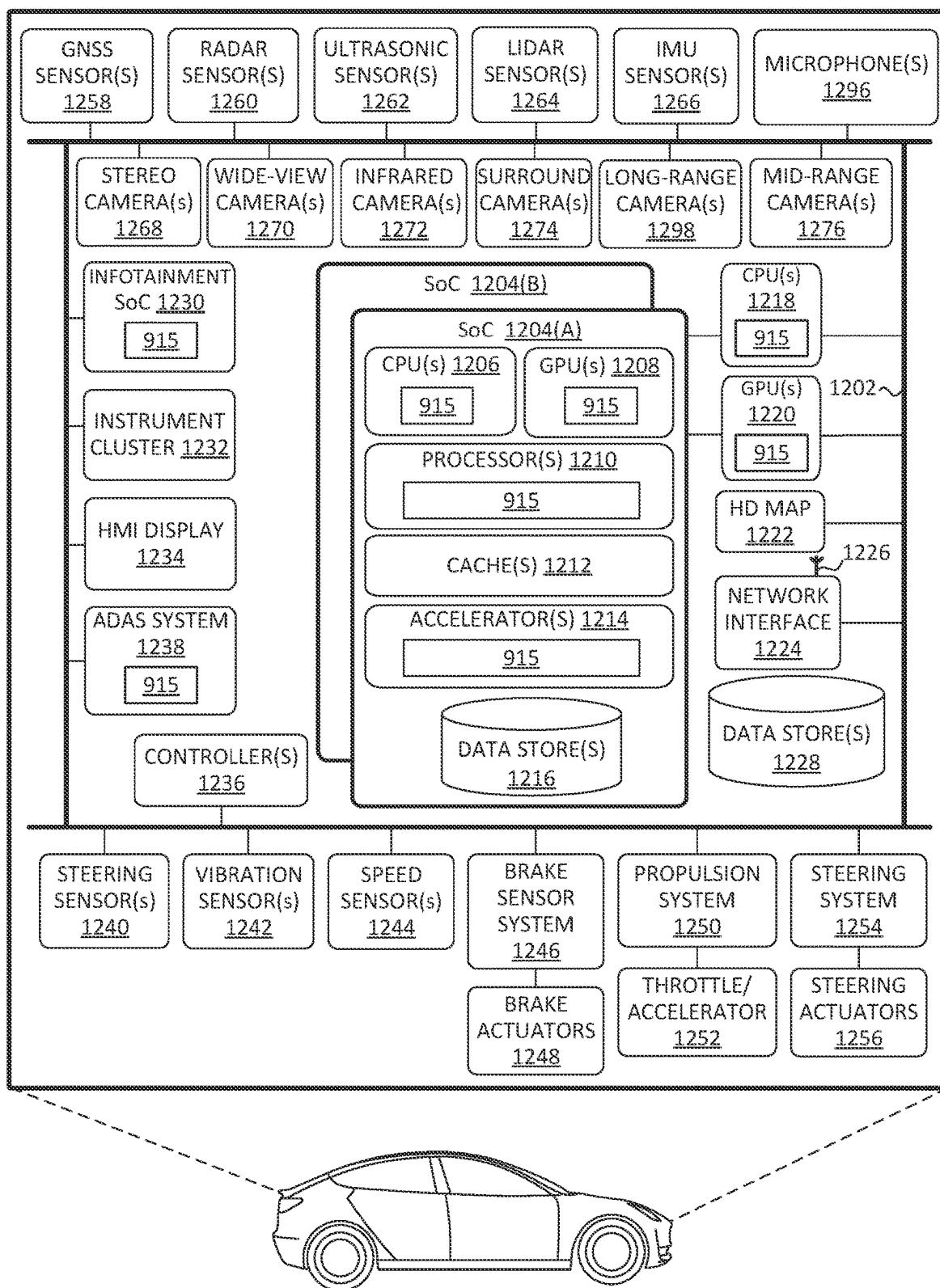
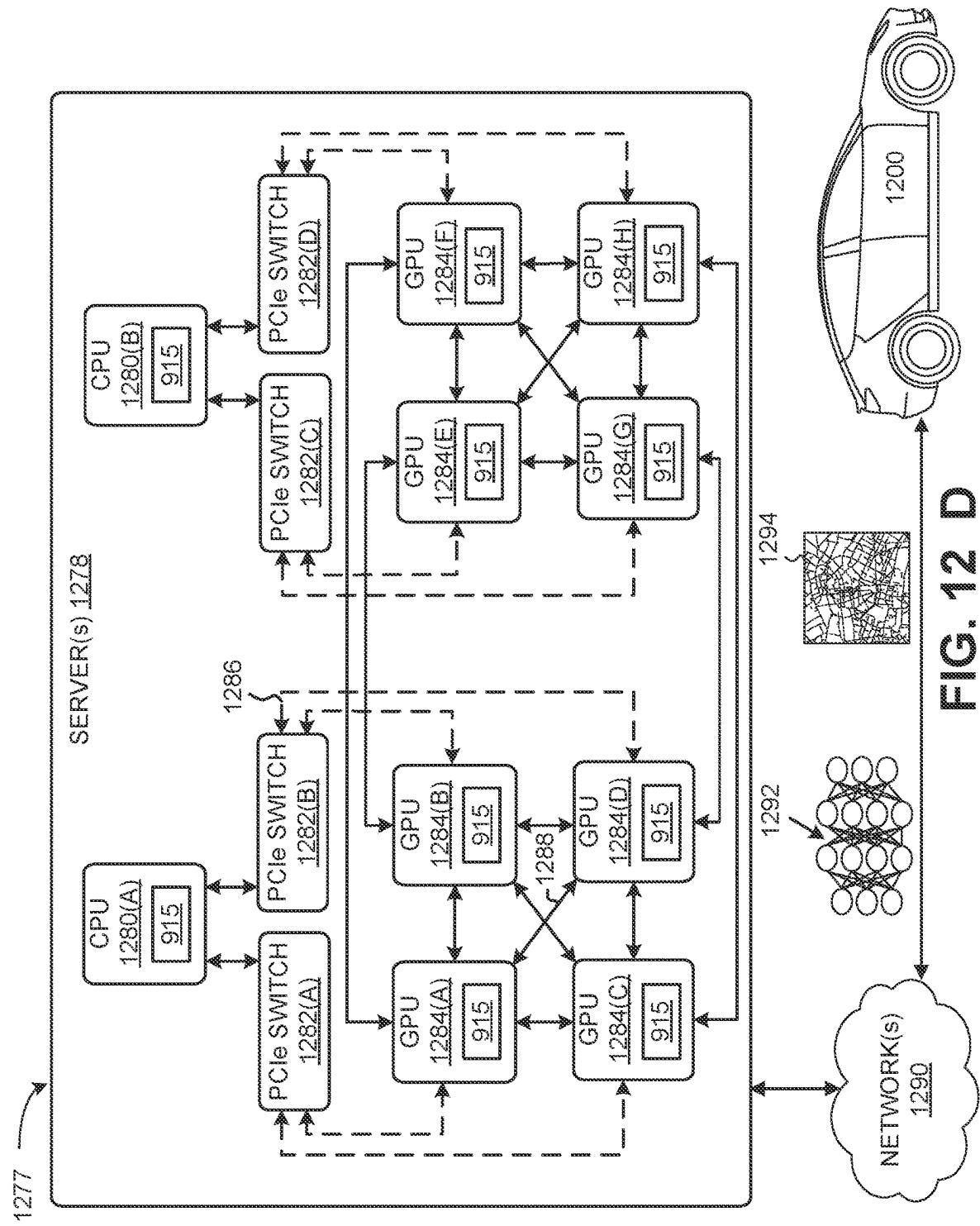


FIG. 12 C



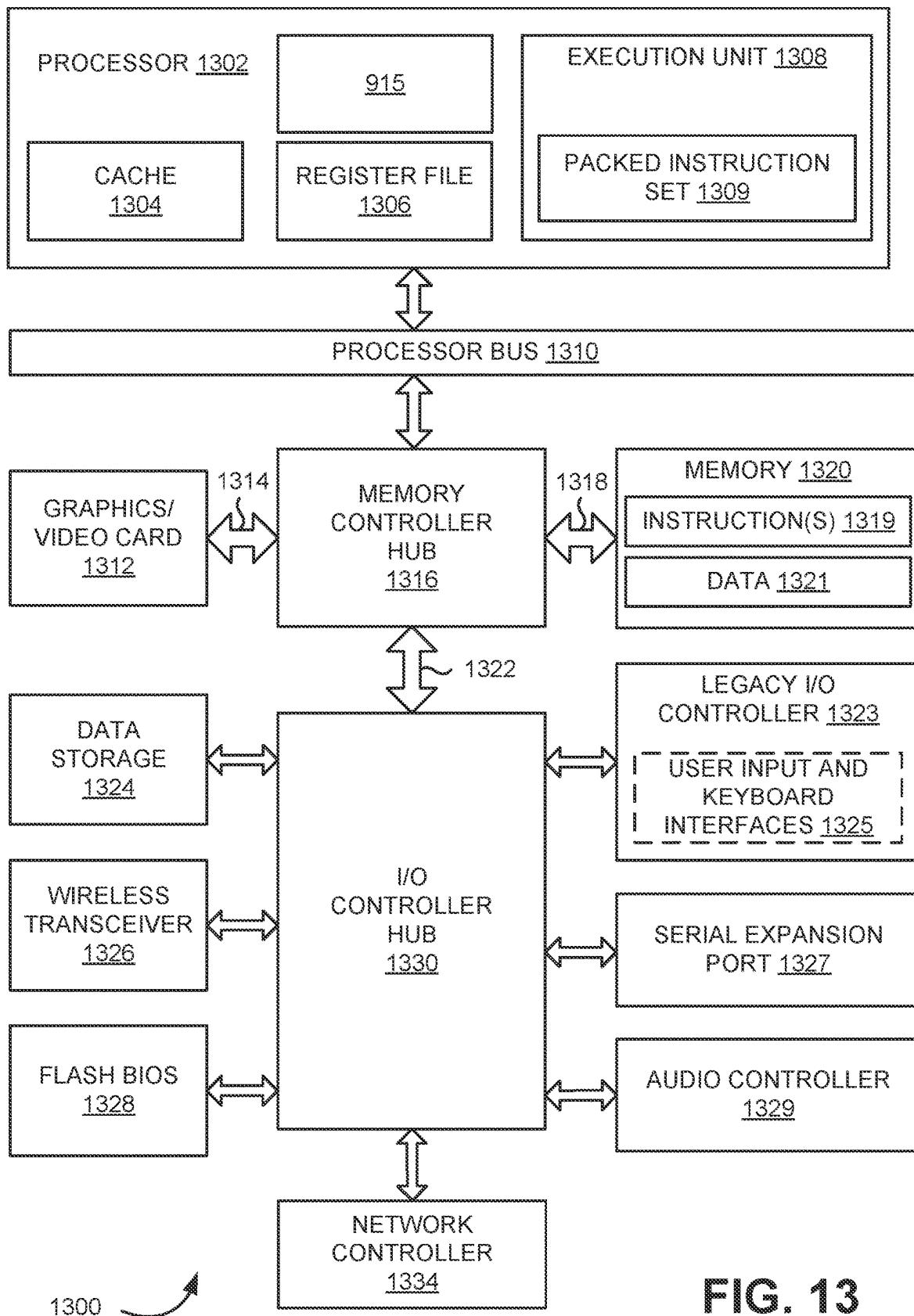
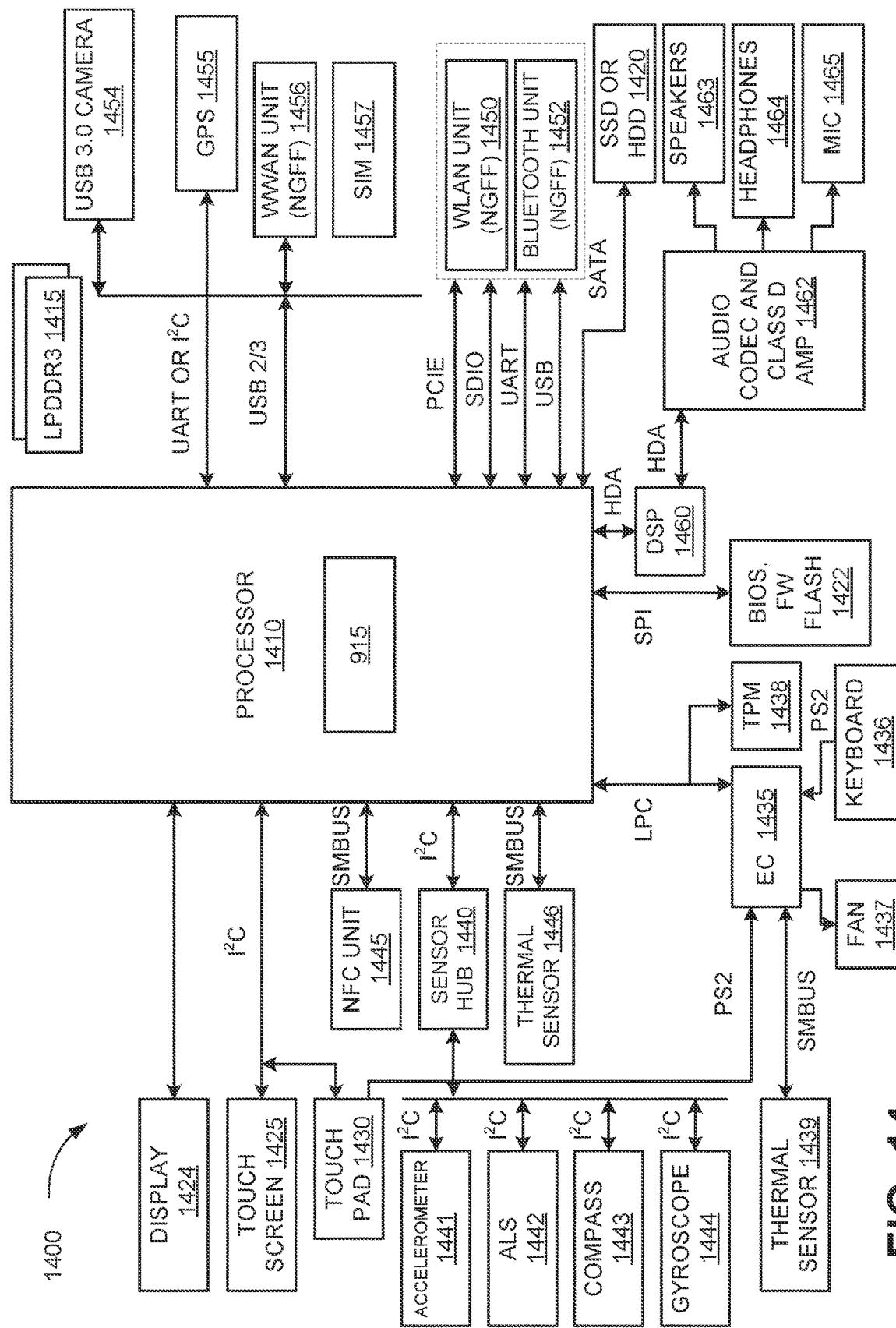


FIG. 13

**FIG. 14**

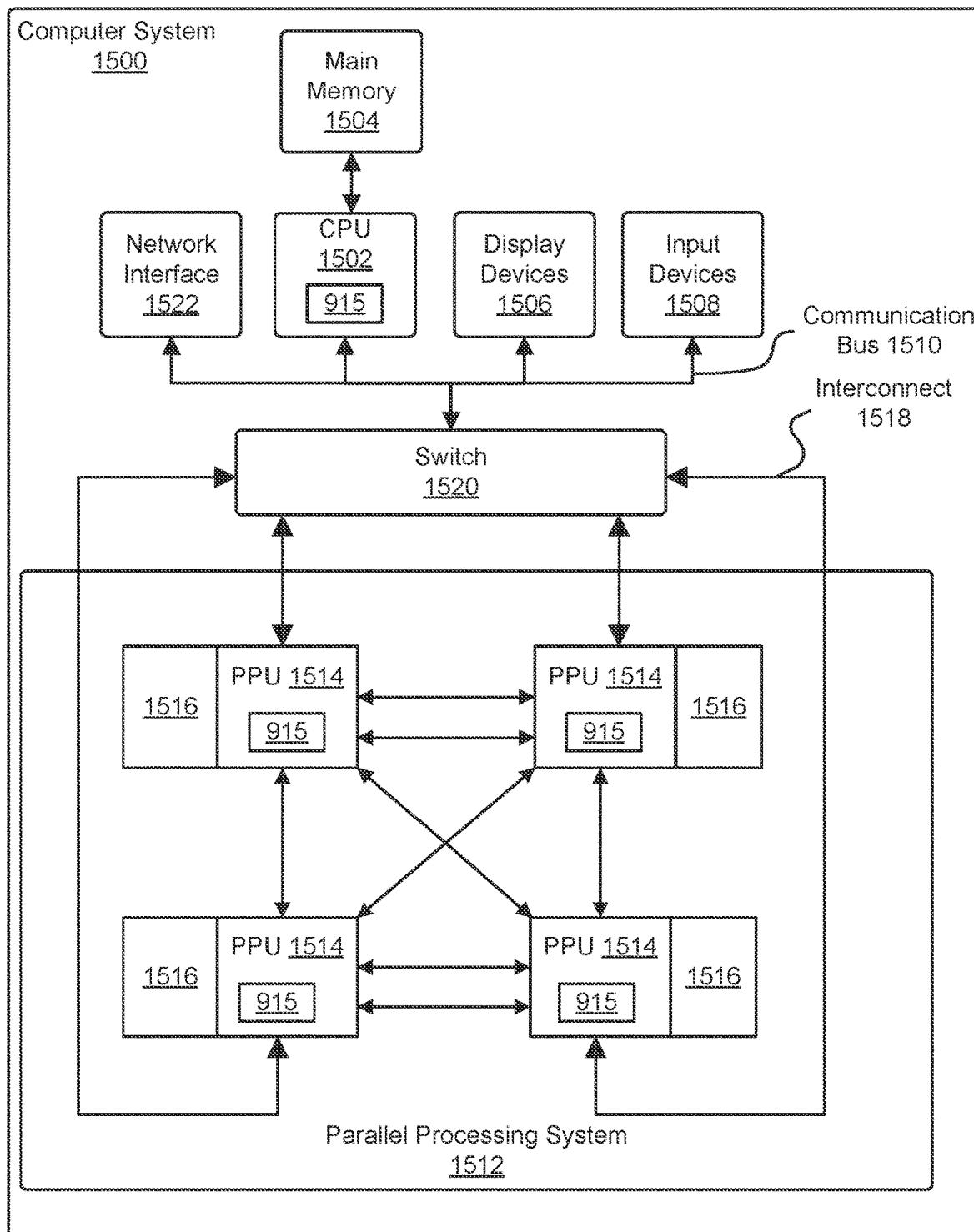


FIG. 15

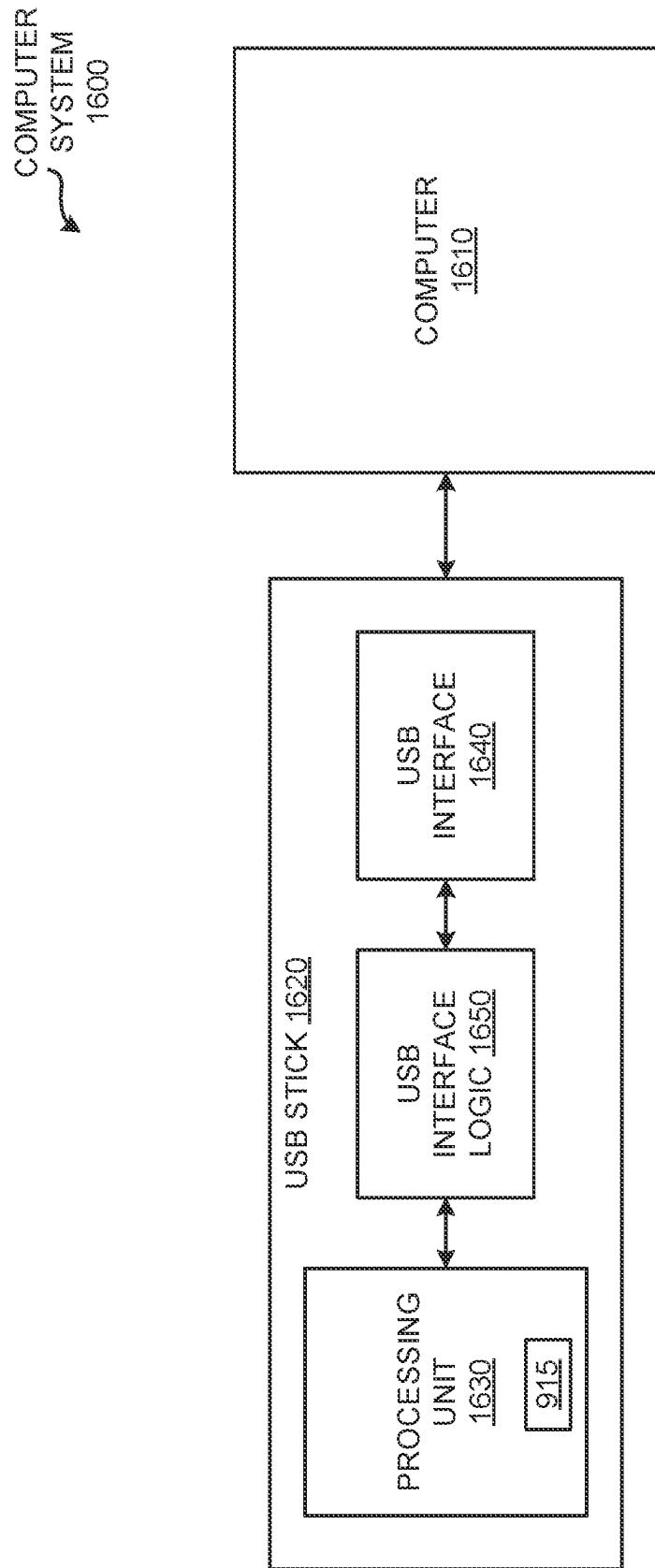


FIG. 16

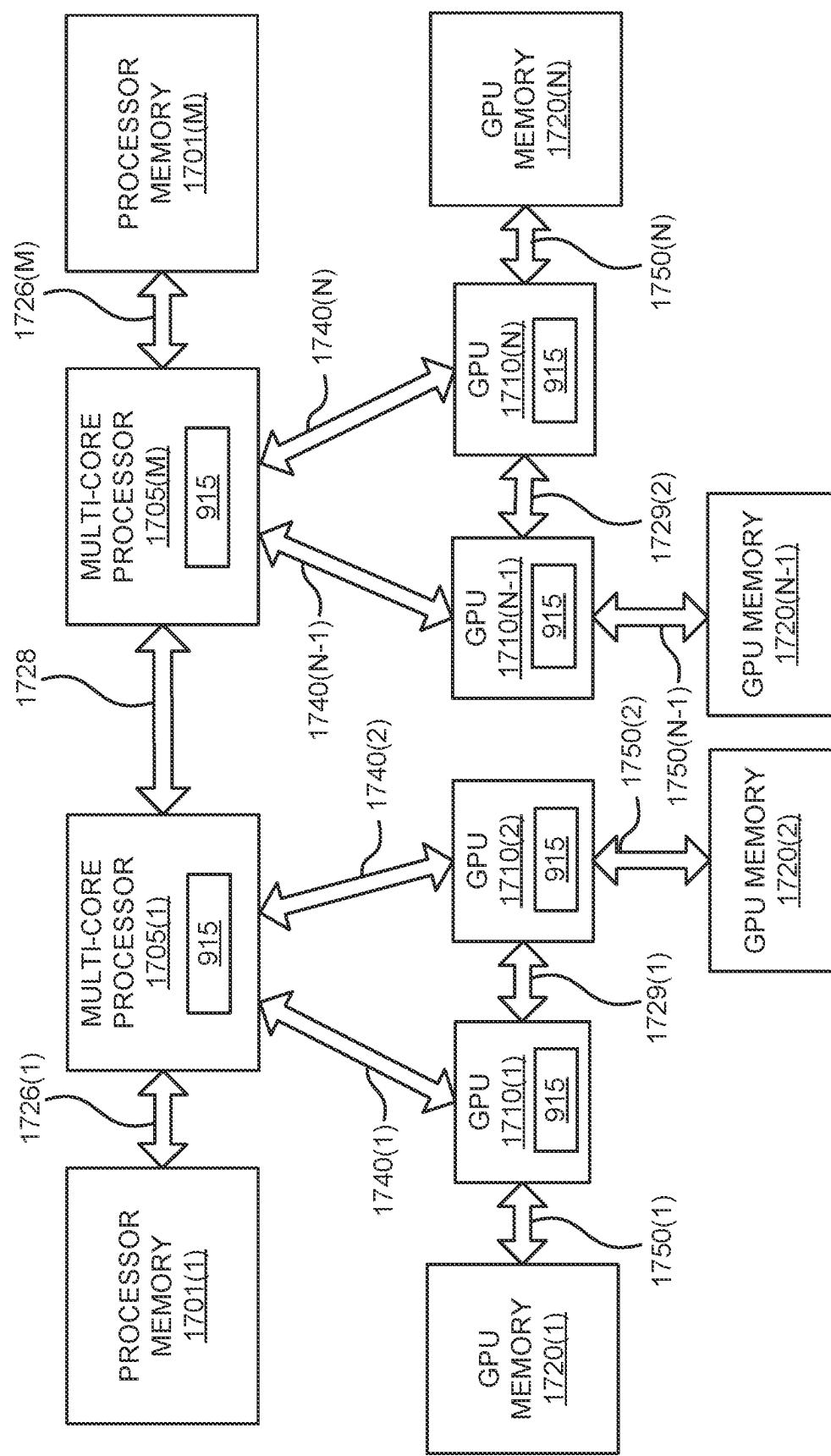
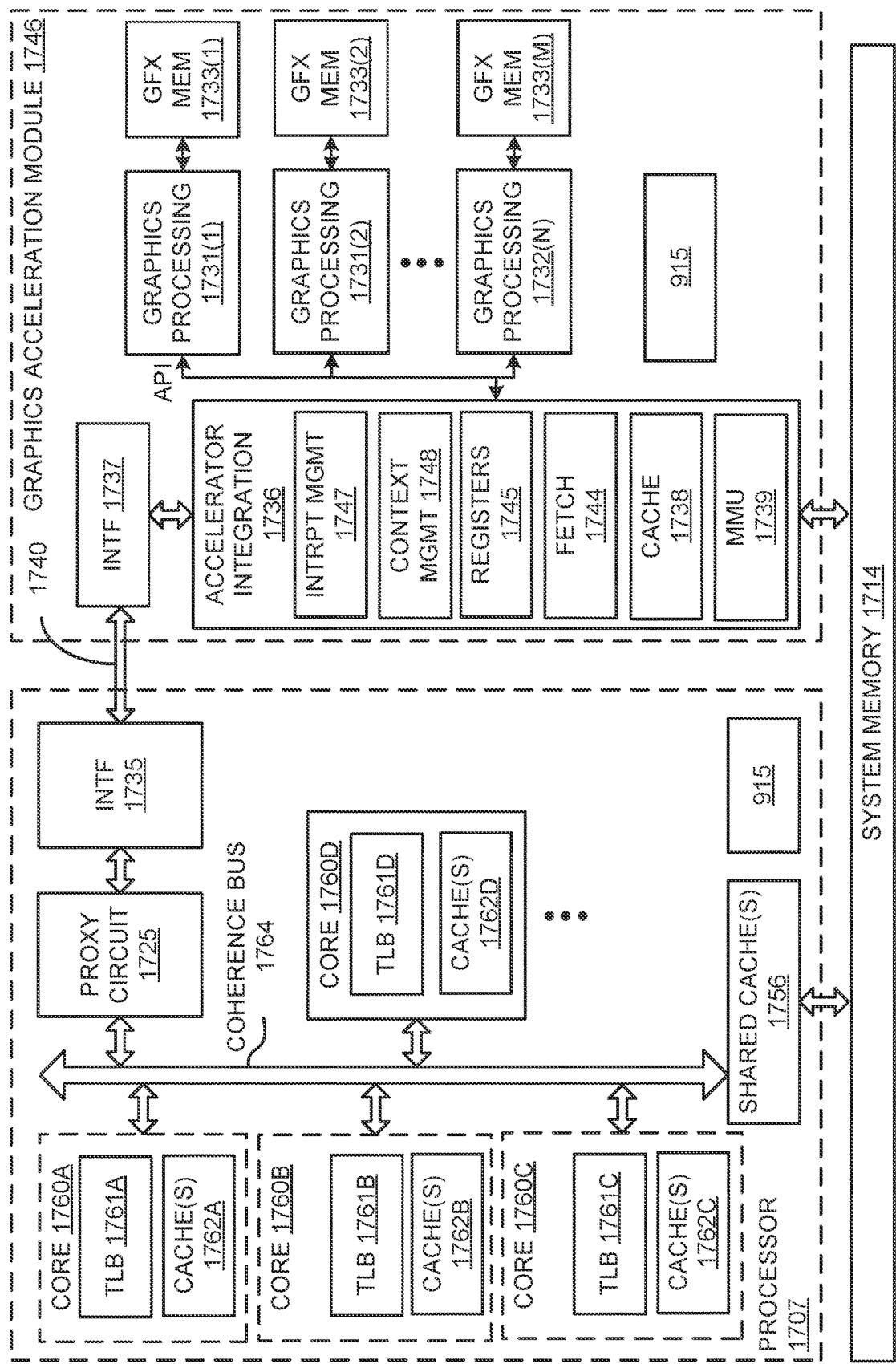
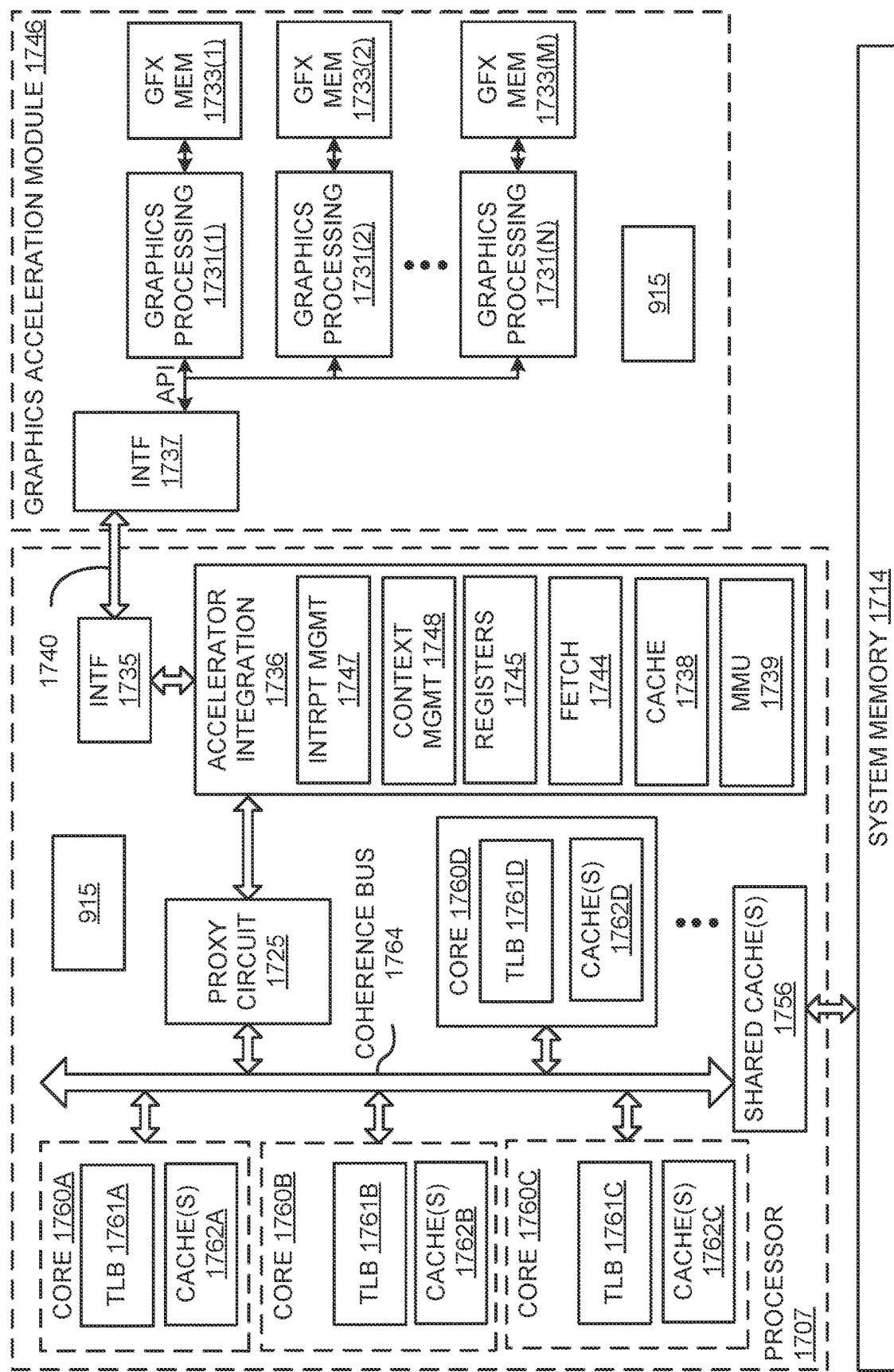


FIG. 17 A

**FIG. 17 B**

**FIG. 17 C**

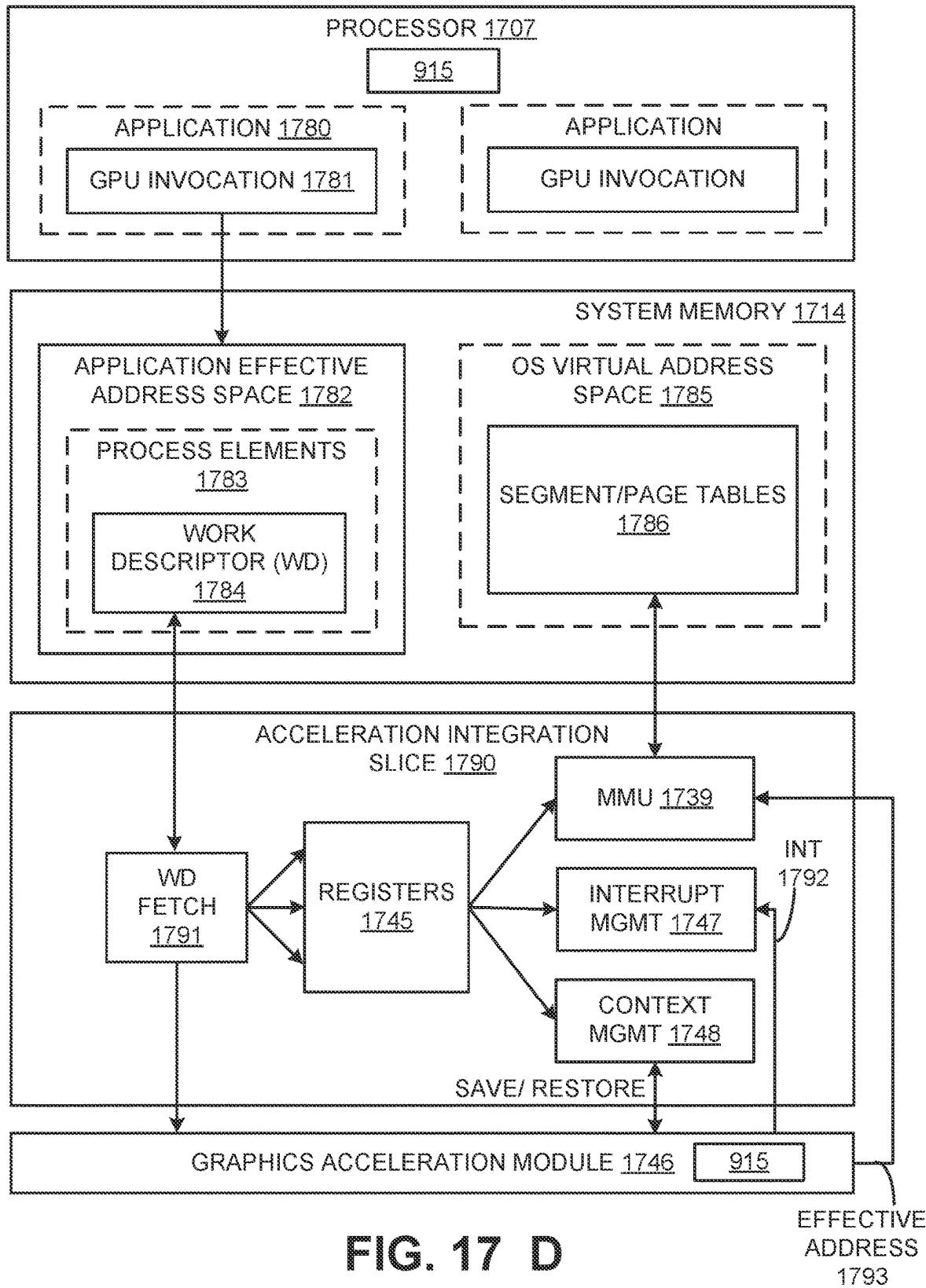
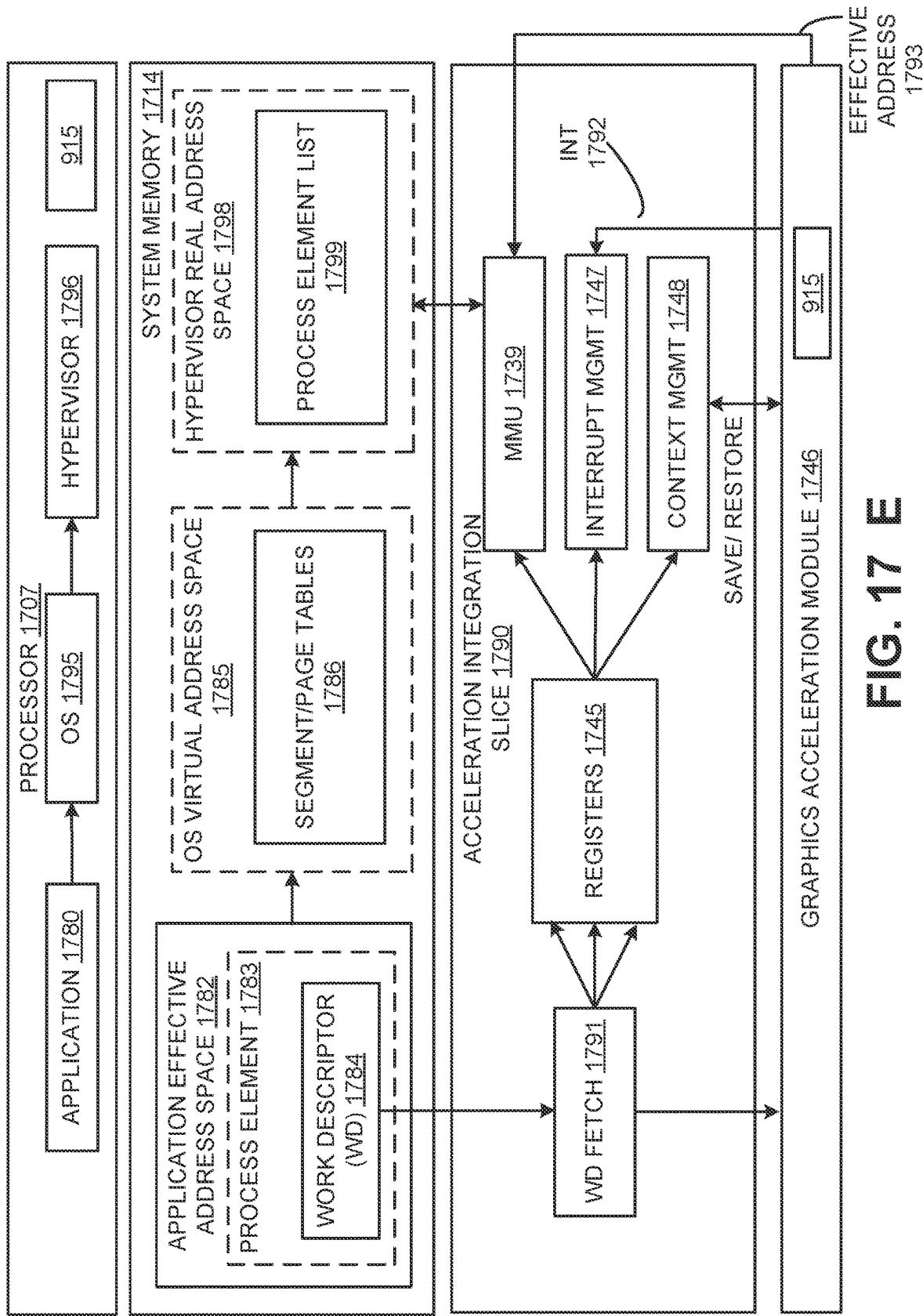


FIG. 17 D

EFFECTIVE  
ADDRESS  
1793

**FIG. 17 E**

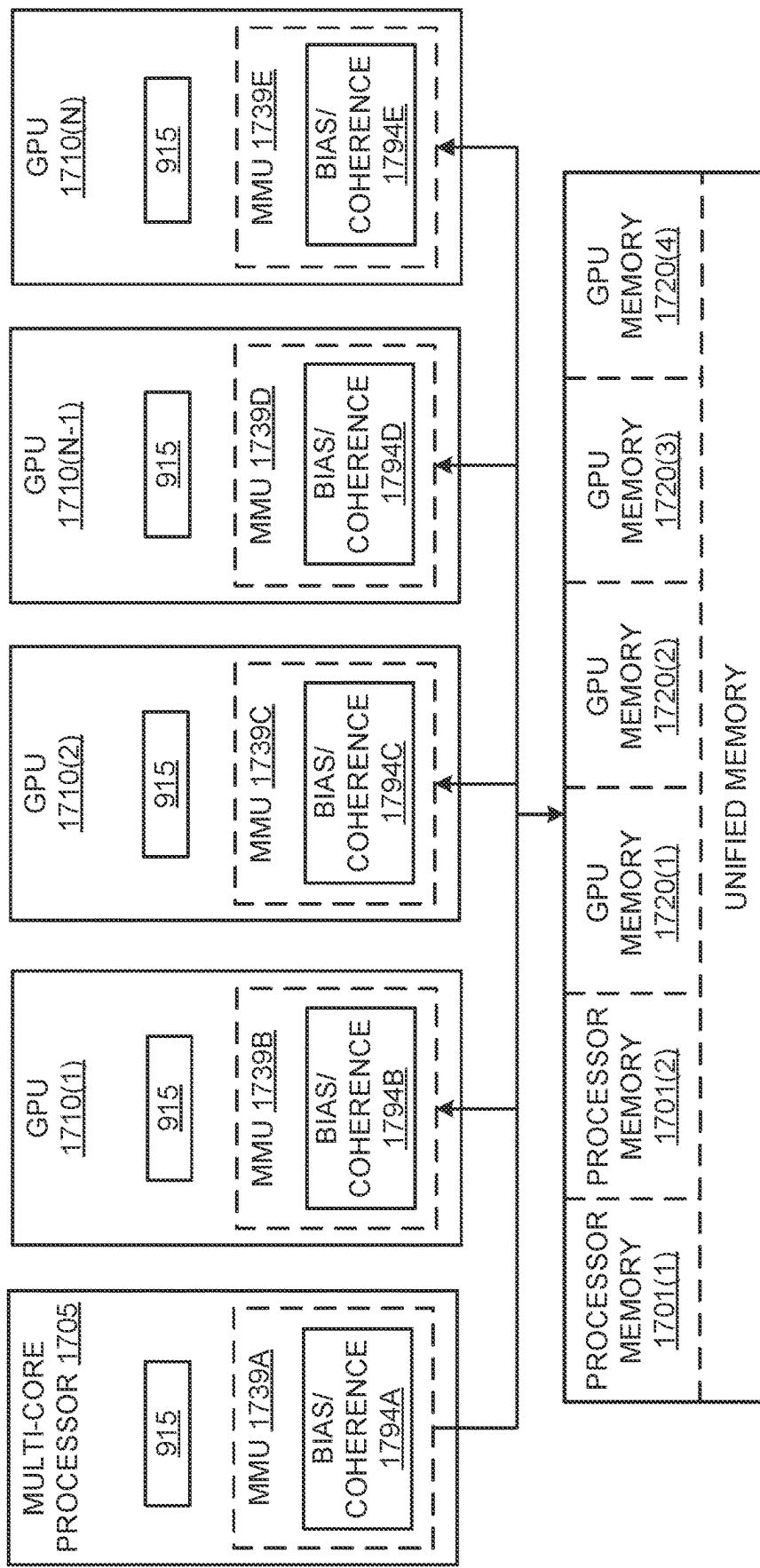


FIG. 17 F

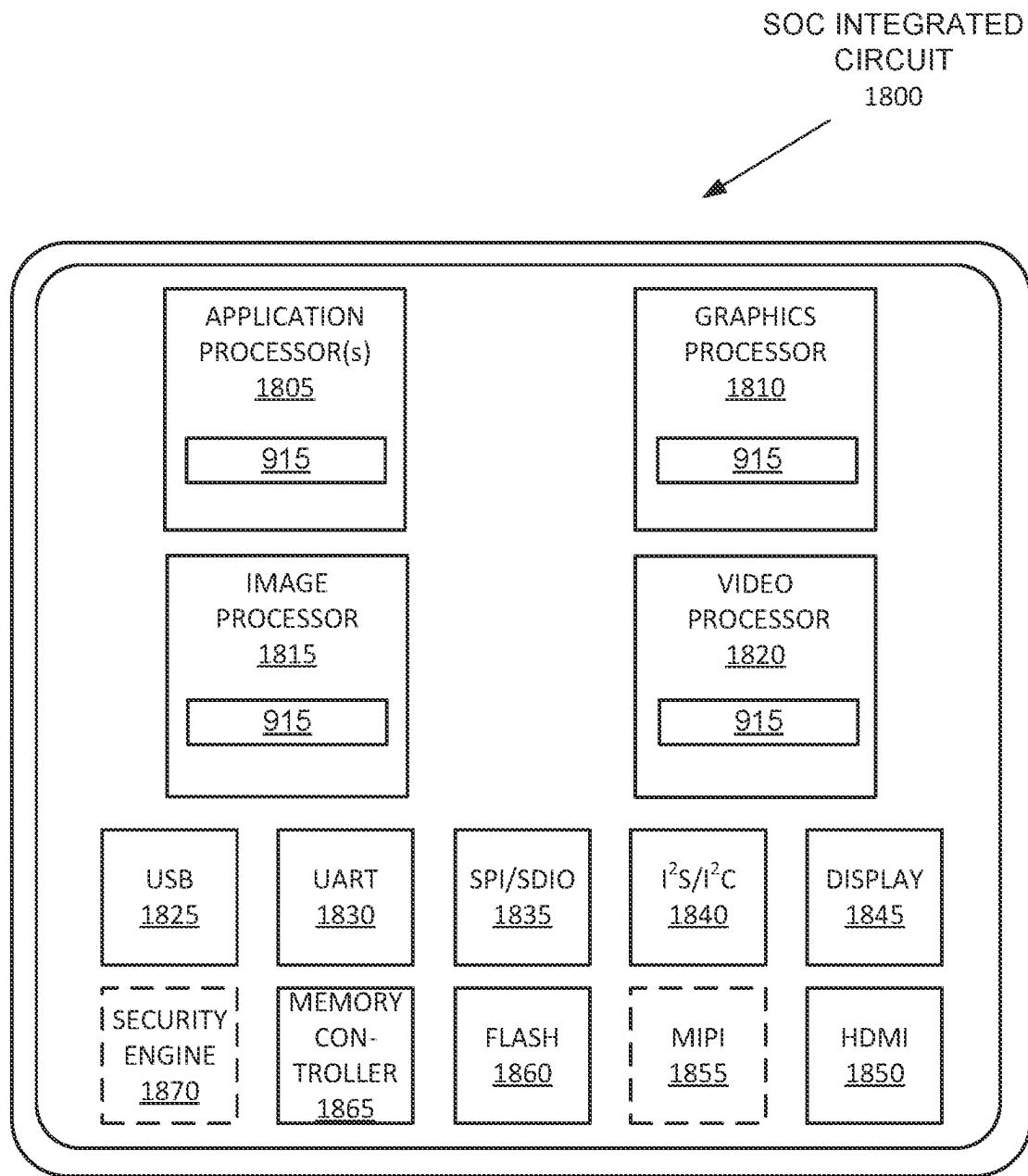
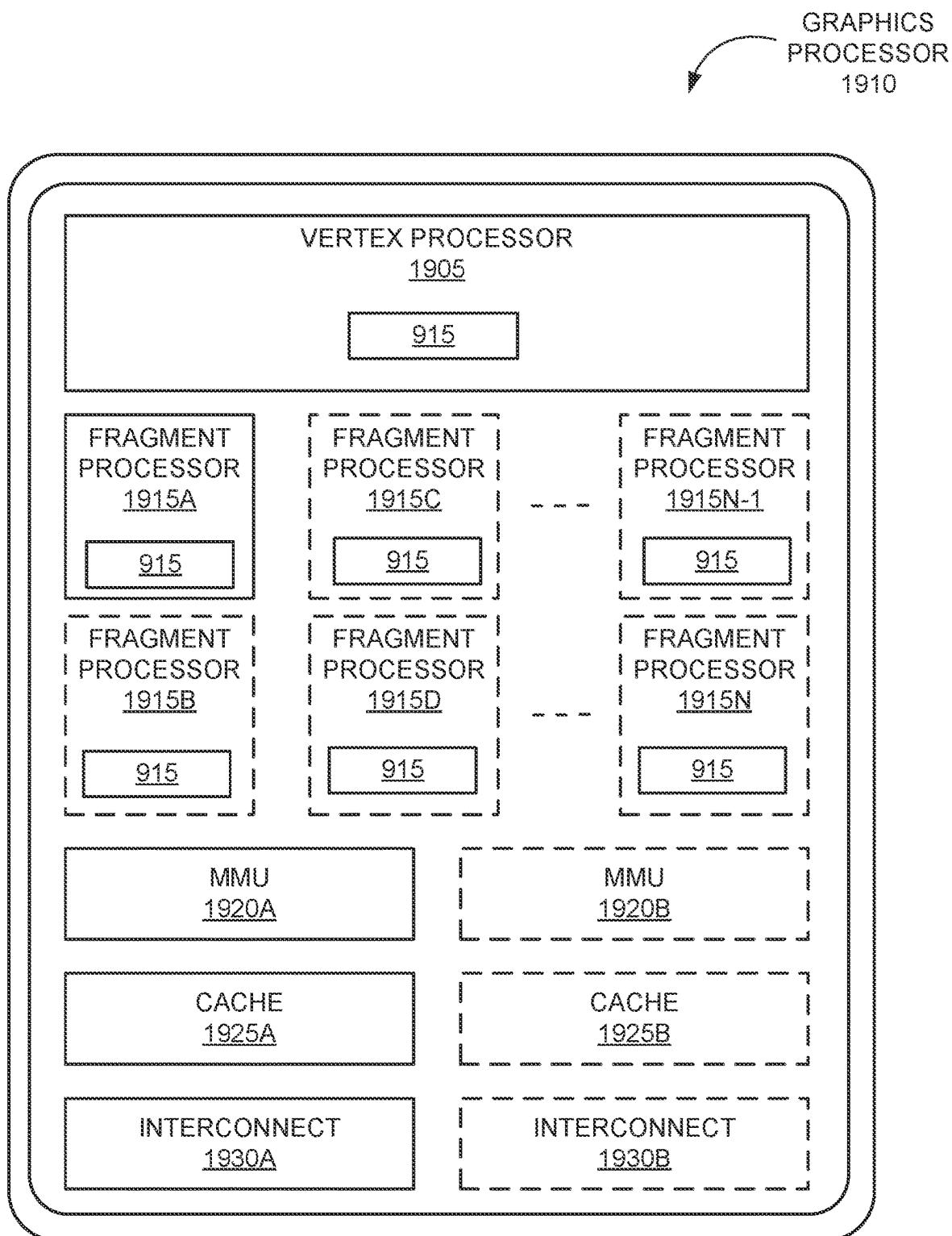


FIG. 18

**FIG. 19A**

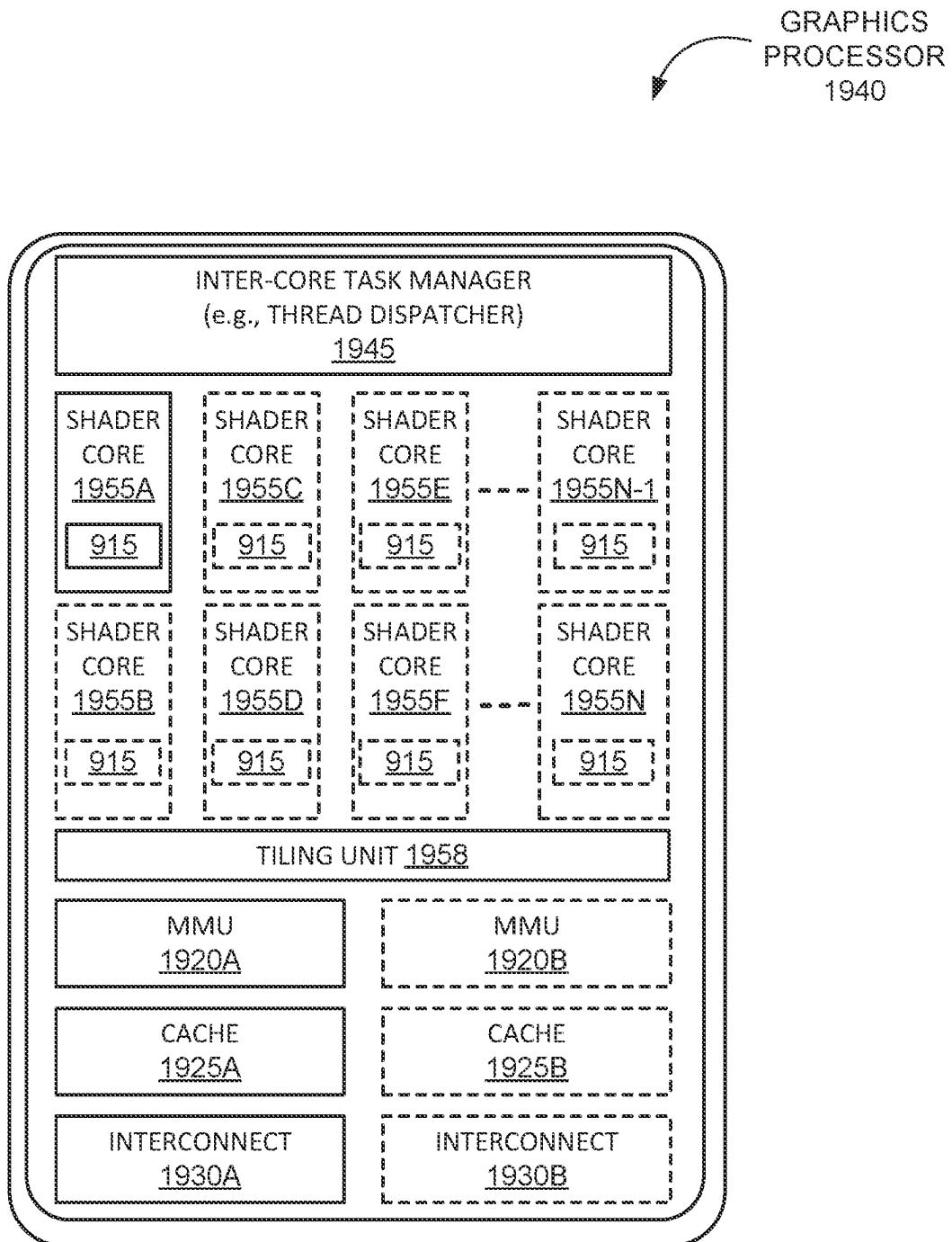
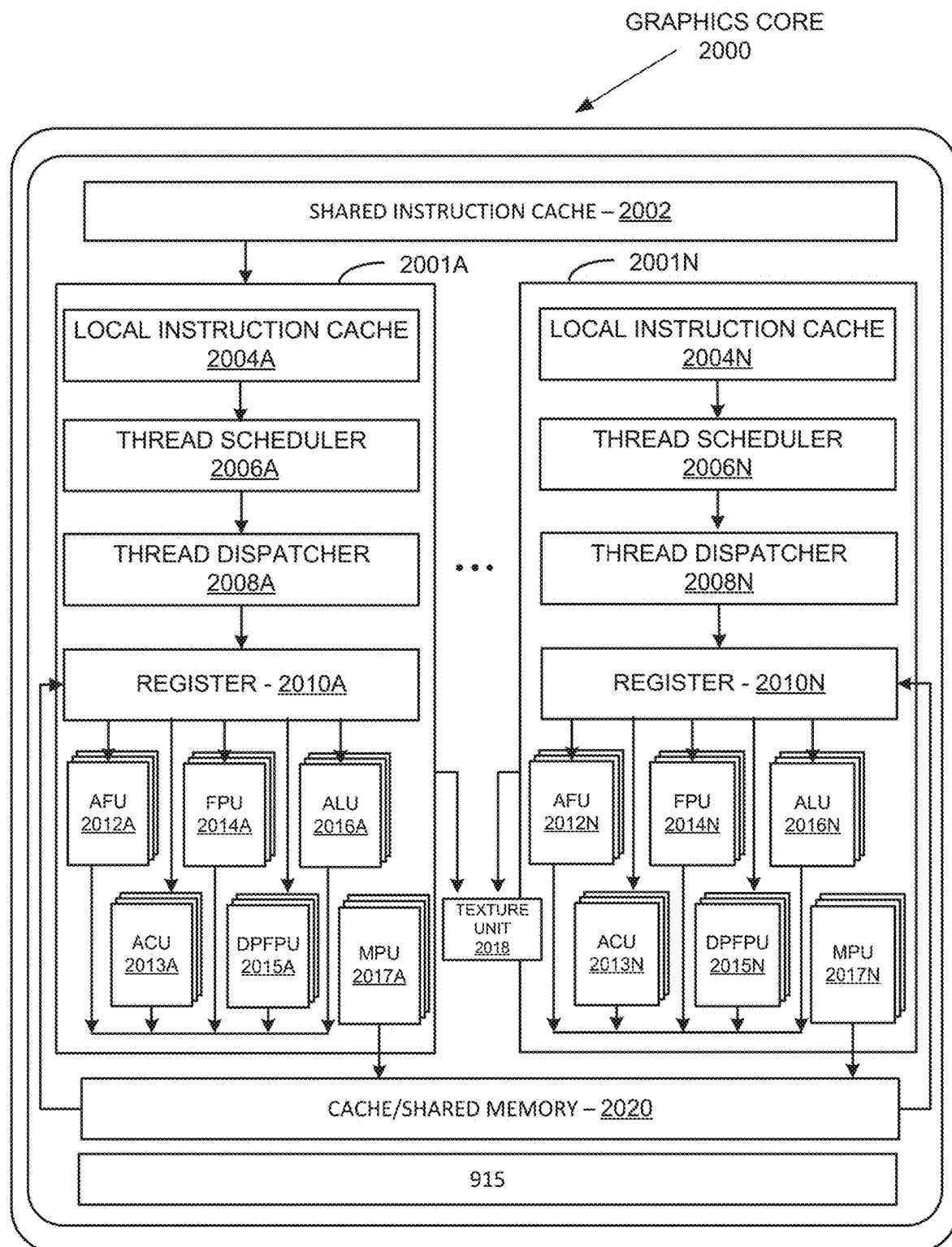


FIG. 19B

**FIG. 20A**

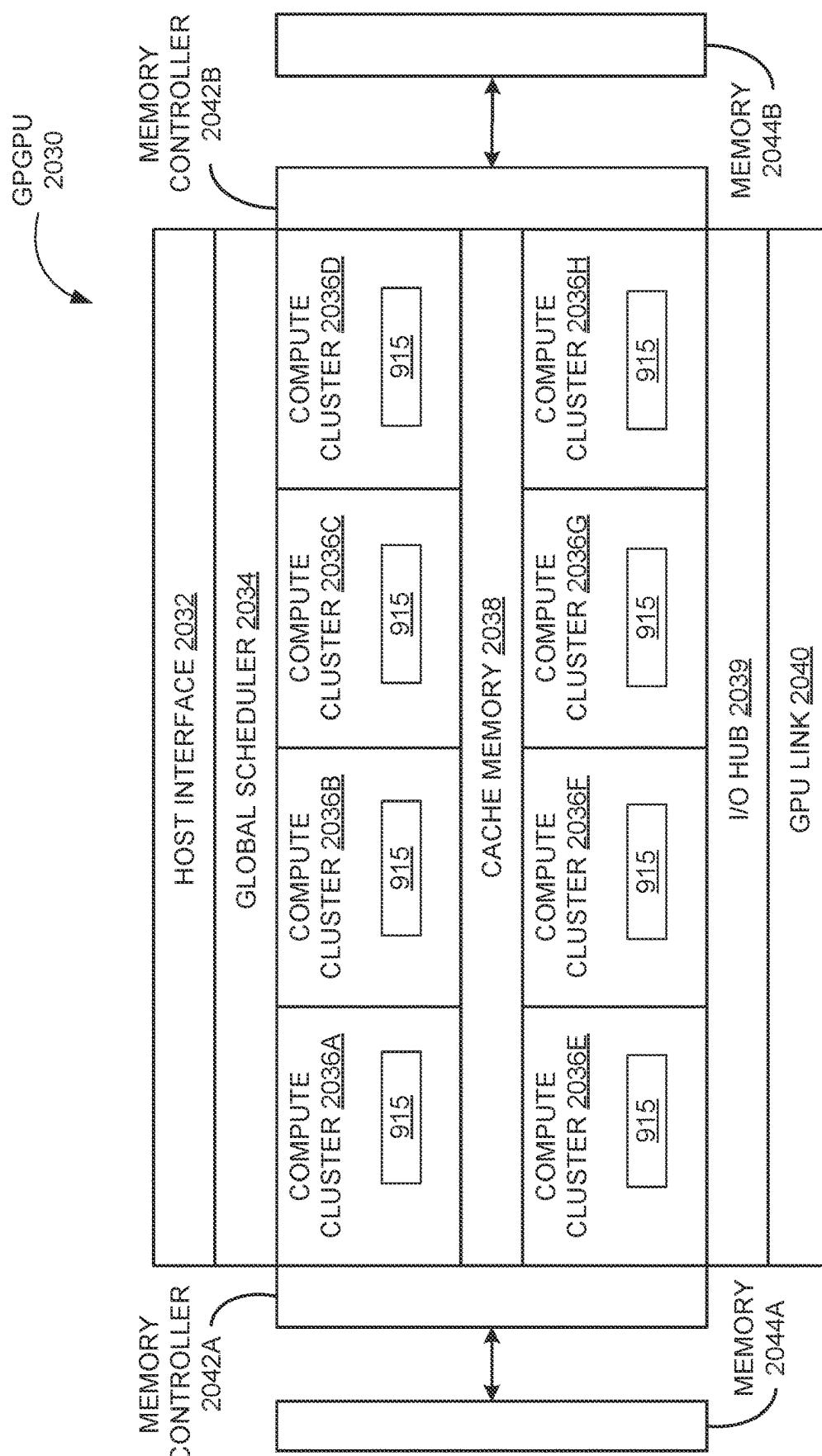


FIG. 20B

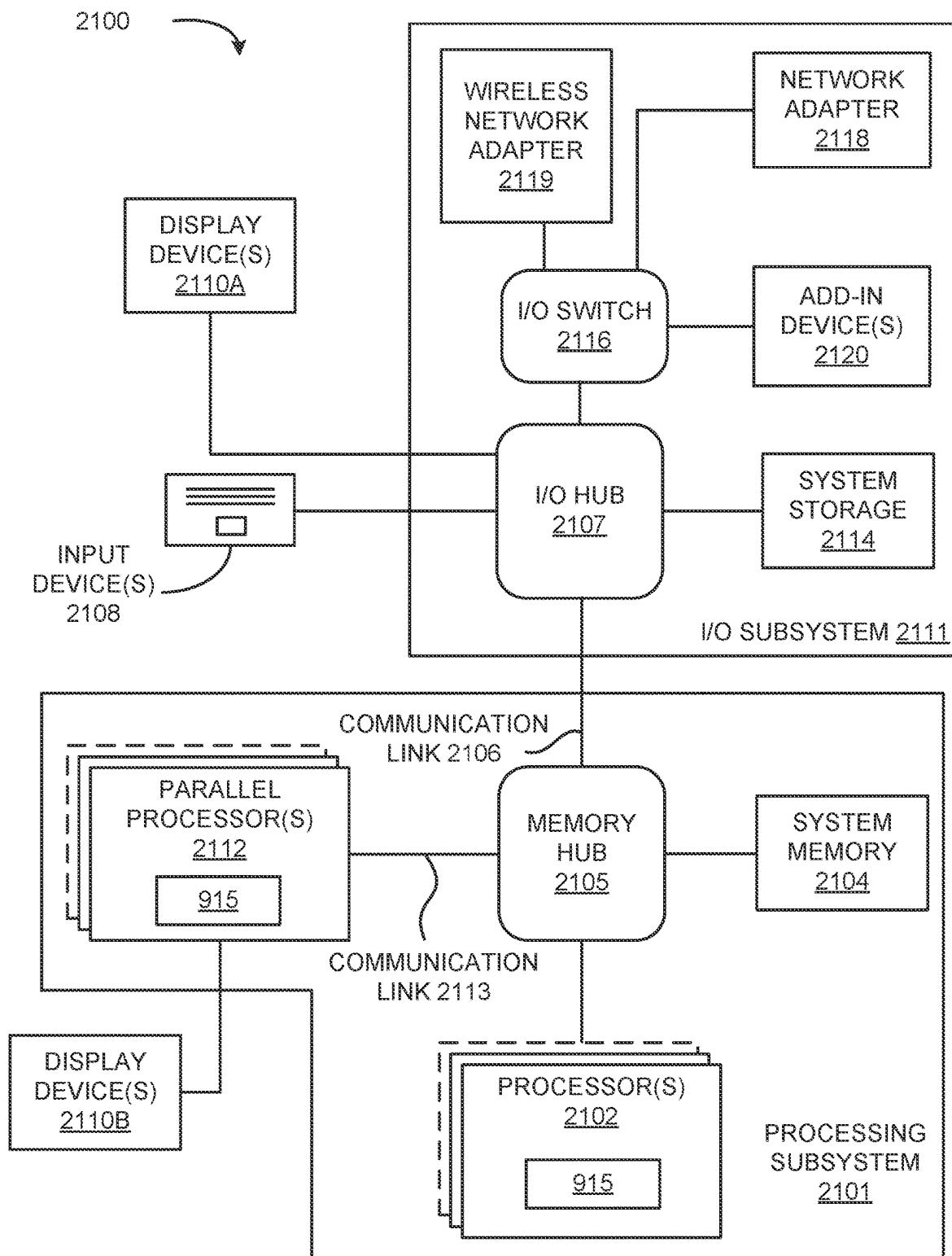
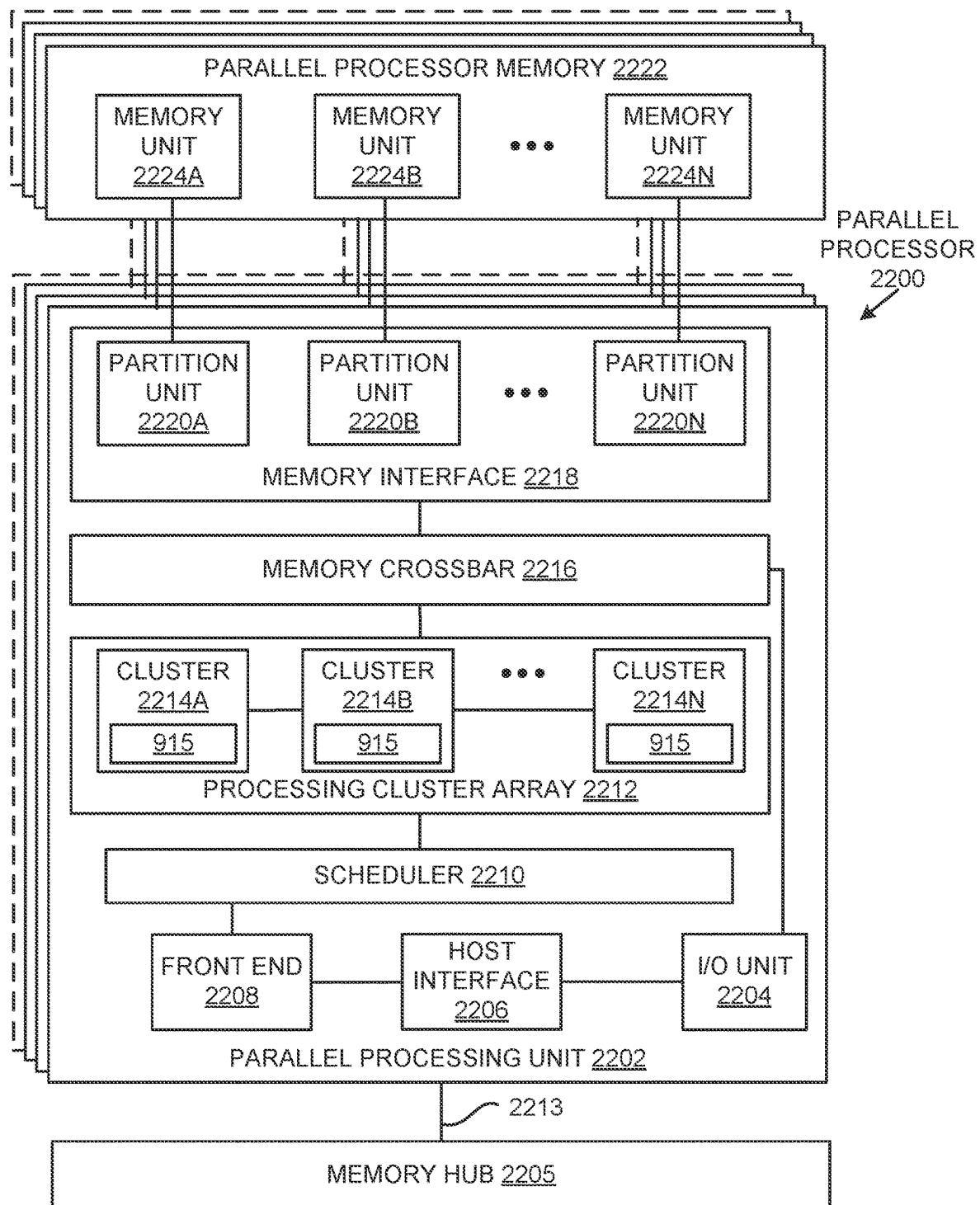
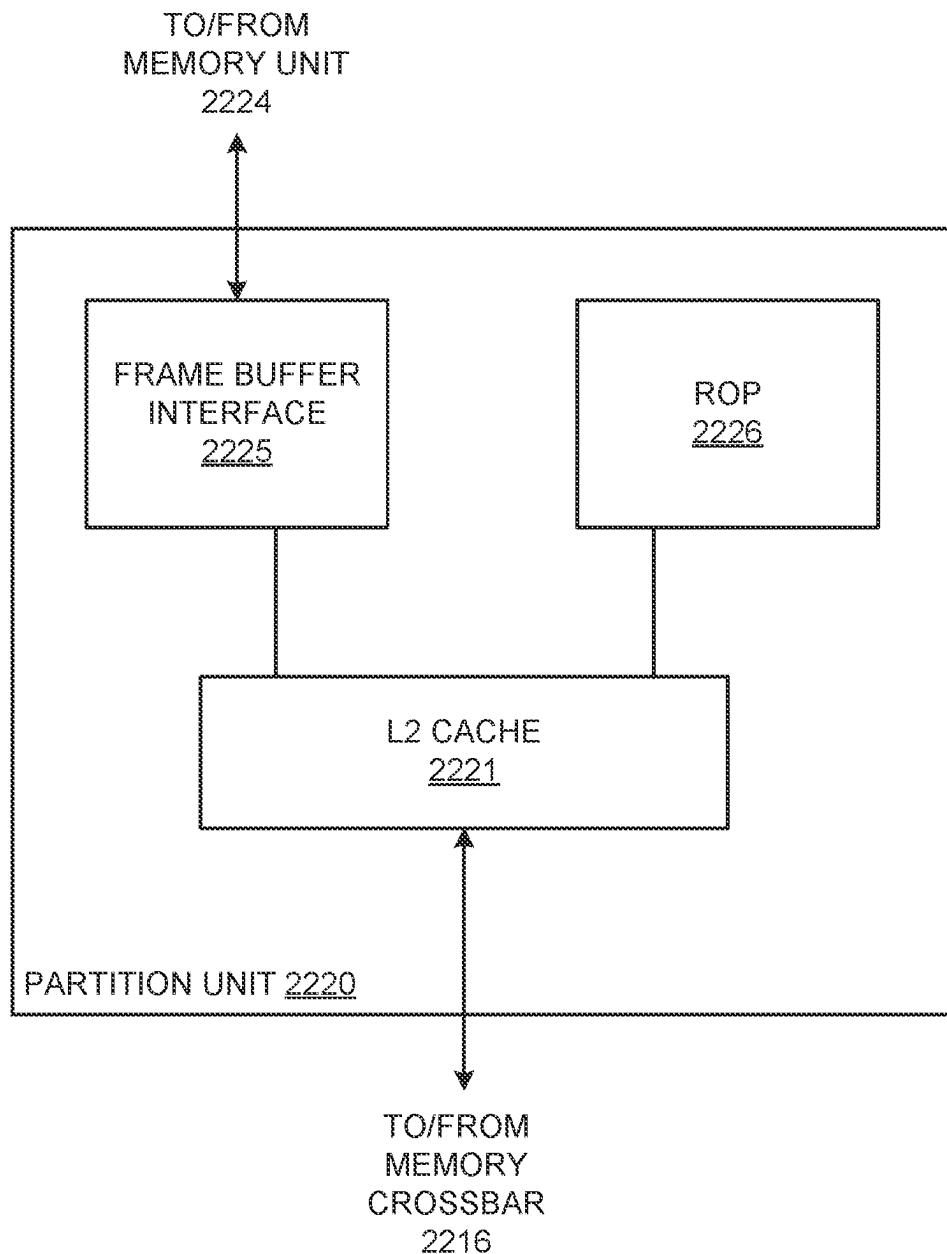


FIG. 21

**FIG. 22 A**

**FIG. 22 B**

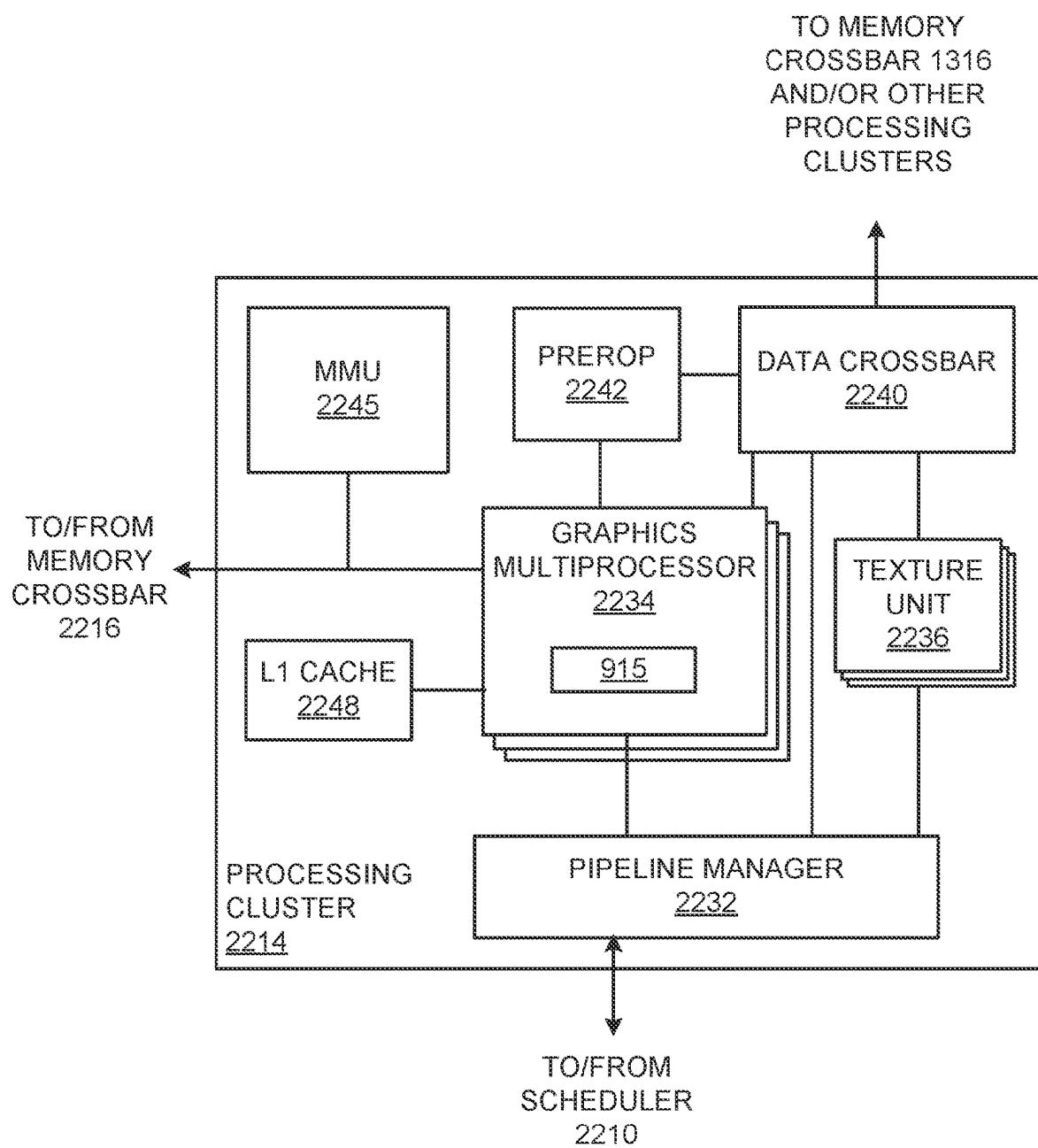


FIG. 22 C

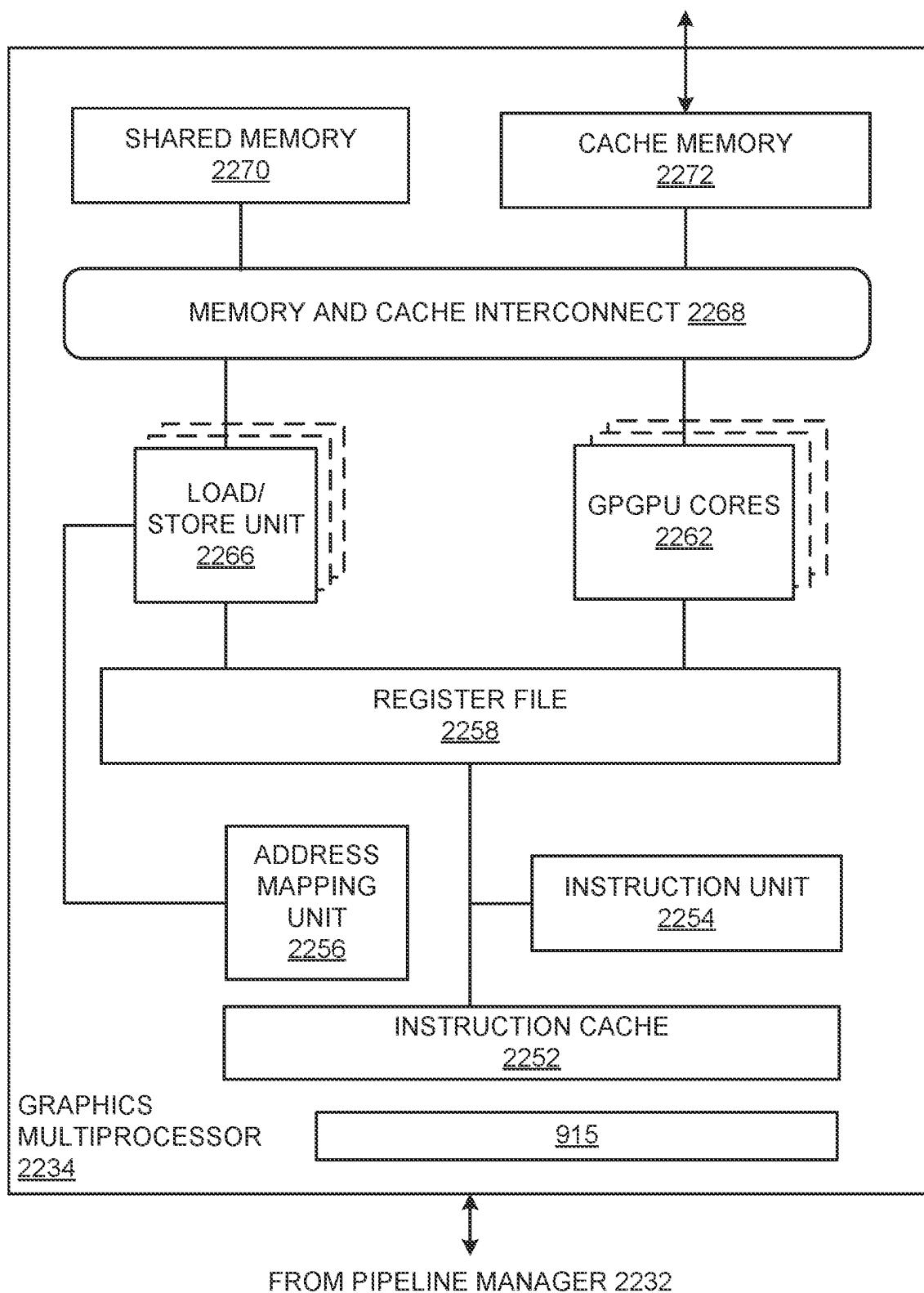
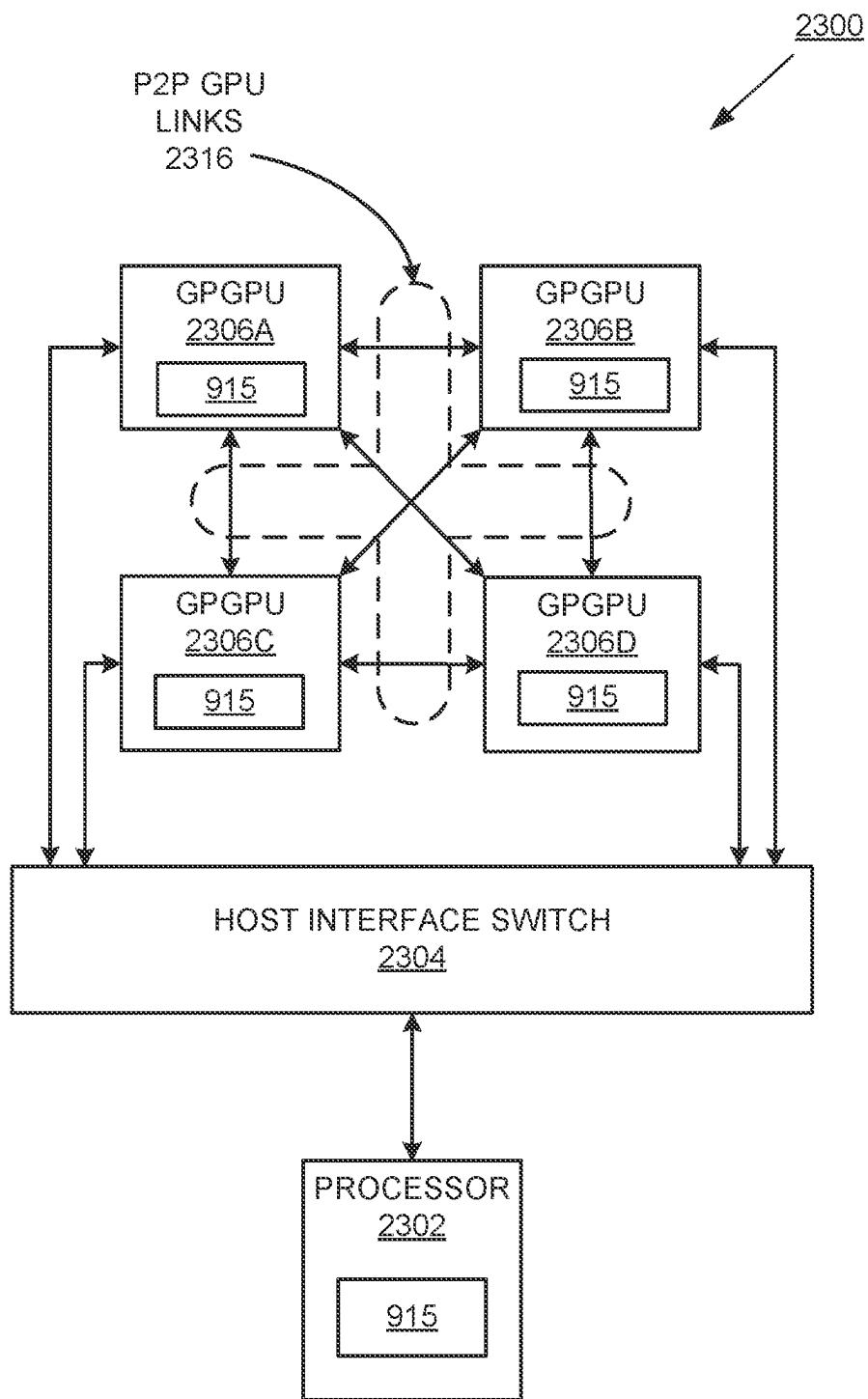


FIG. 22 D

**FIG. 23**

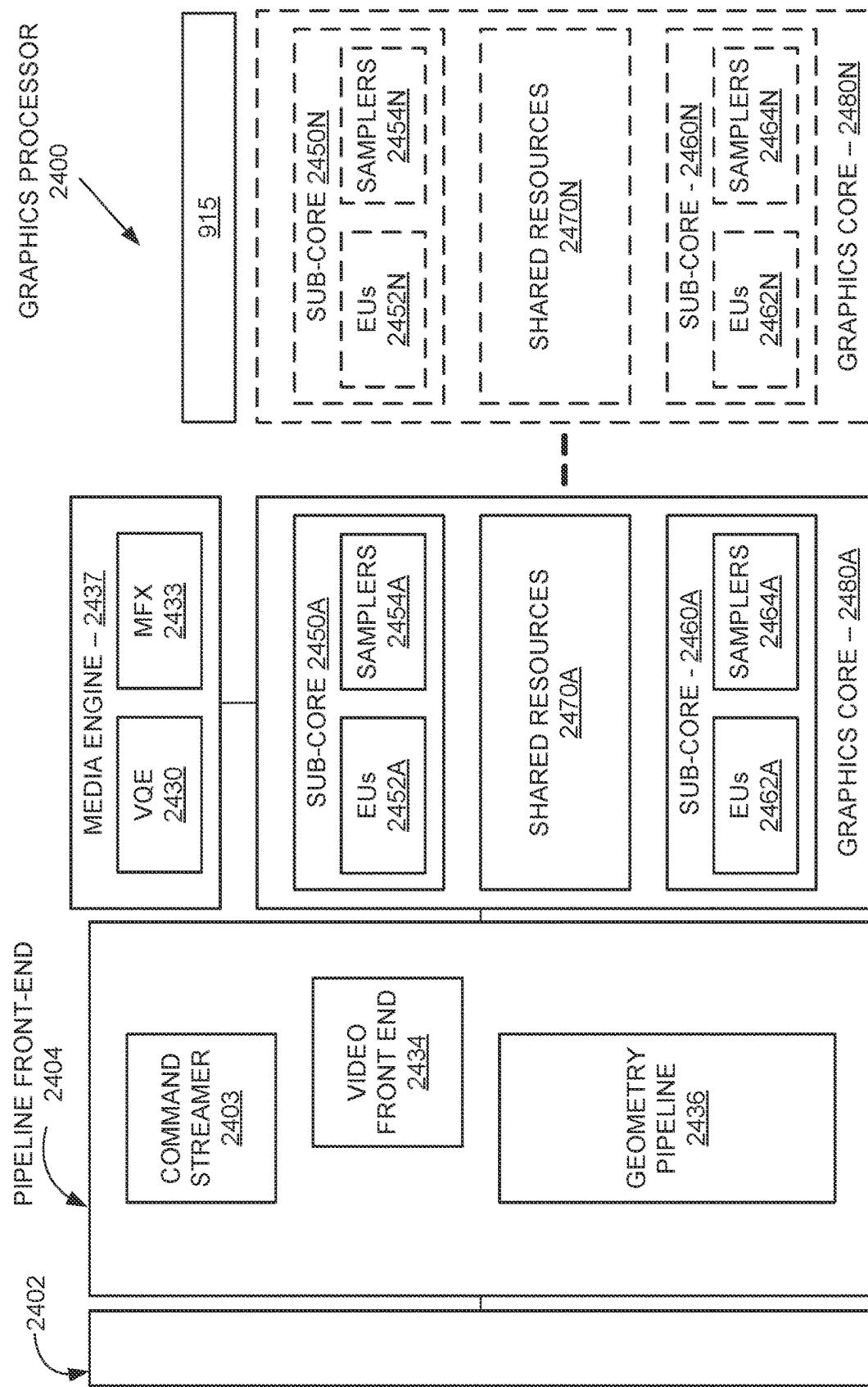
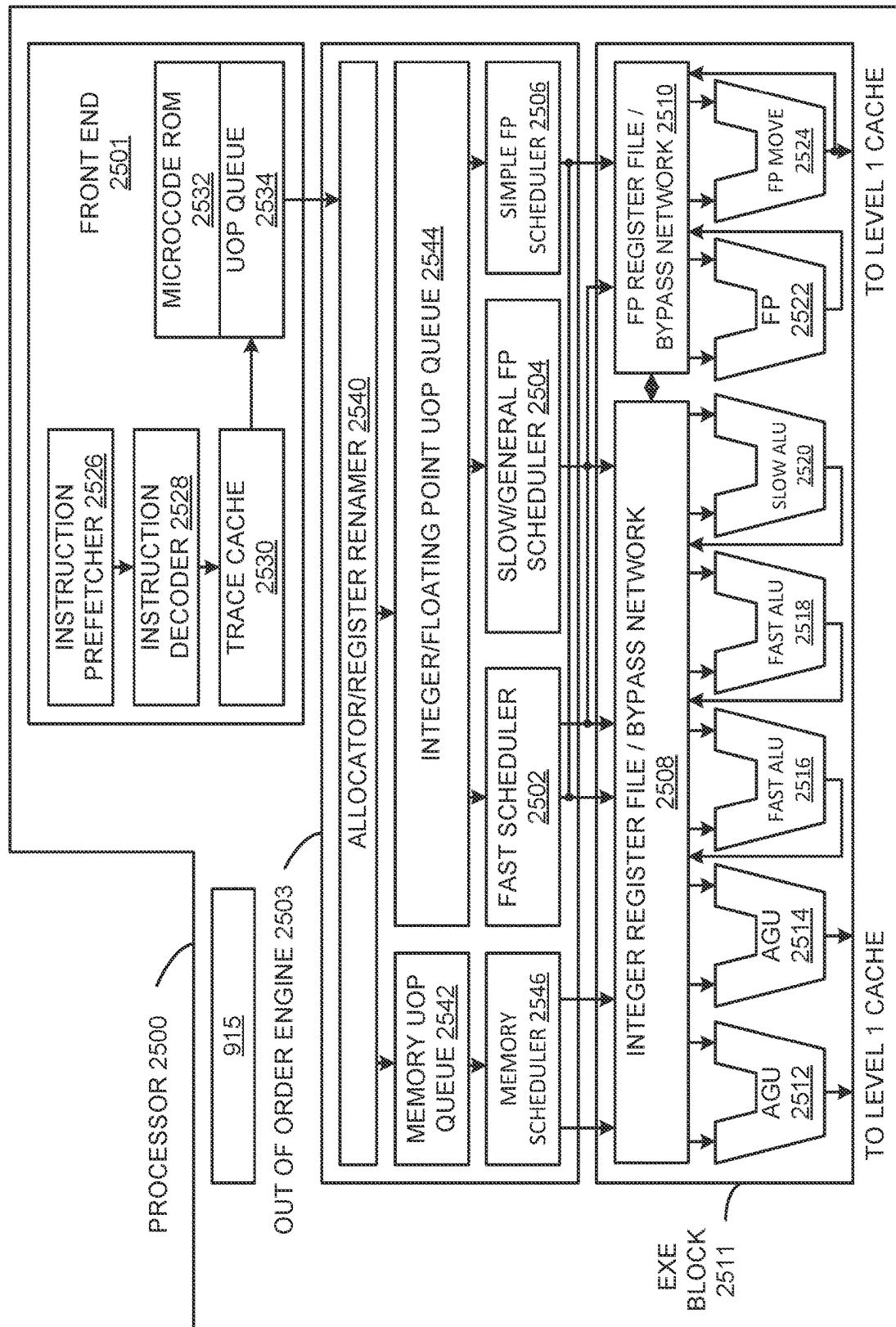


FIG. 24

**FIG. 25**

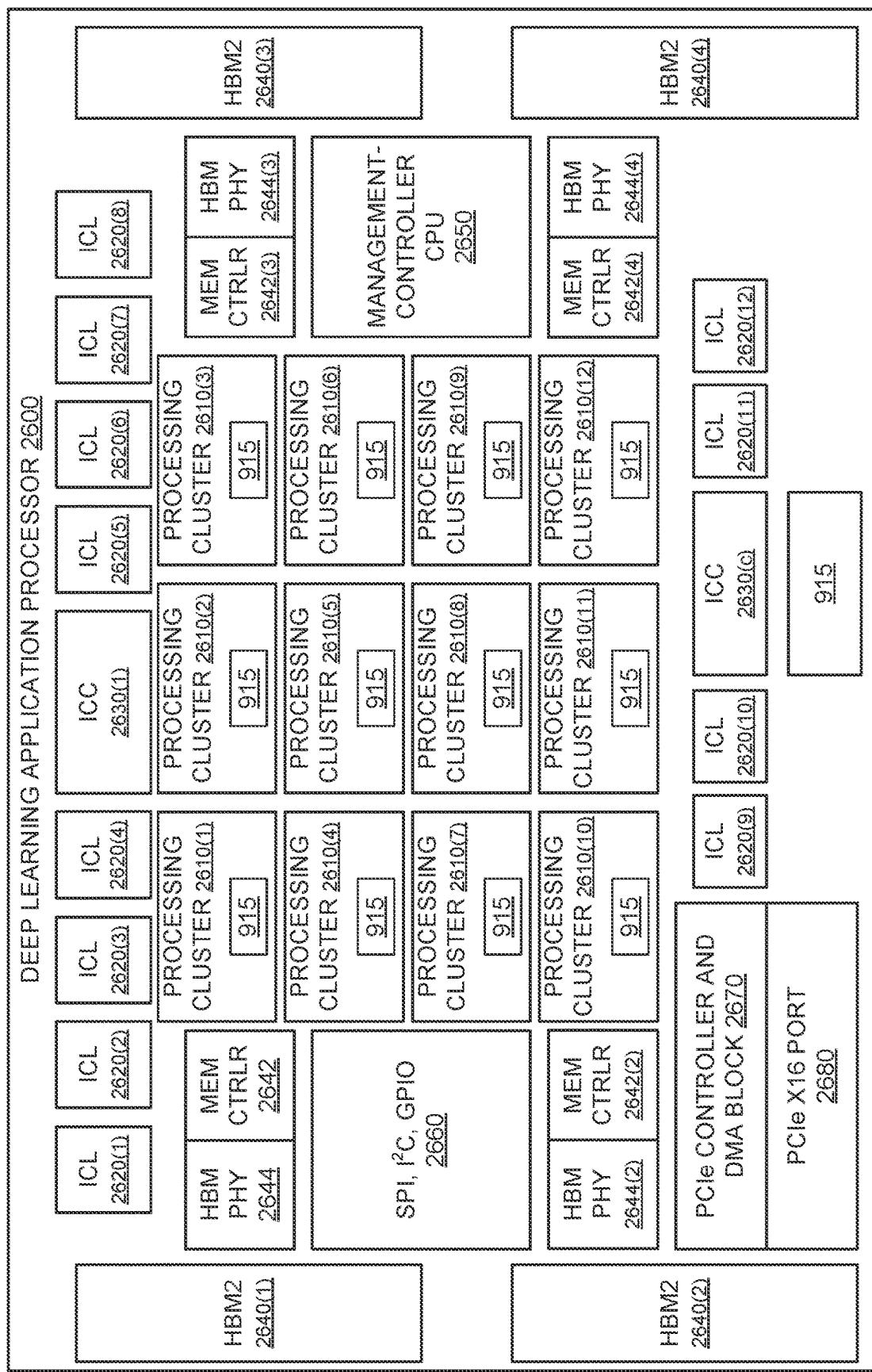
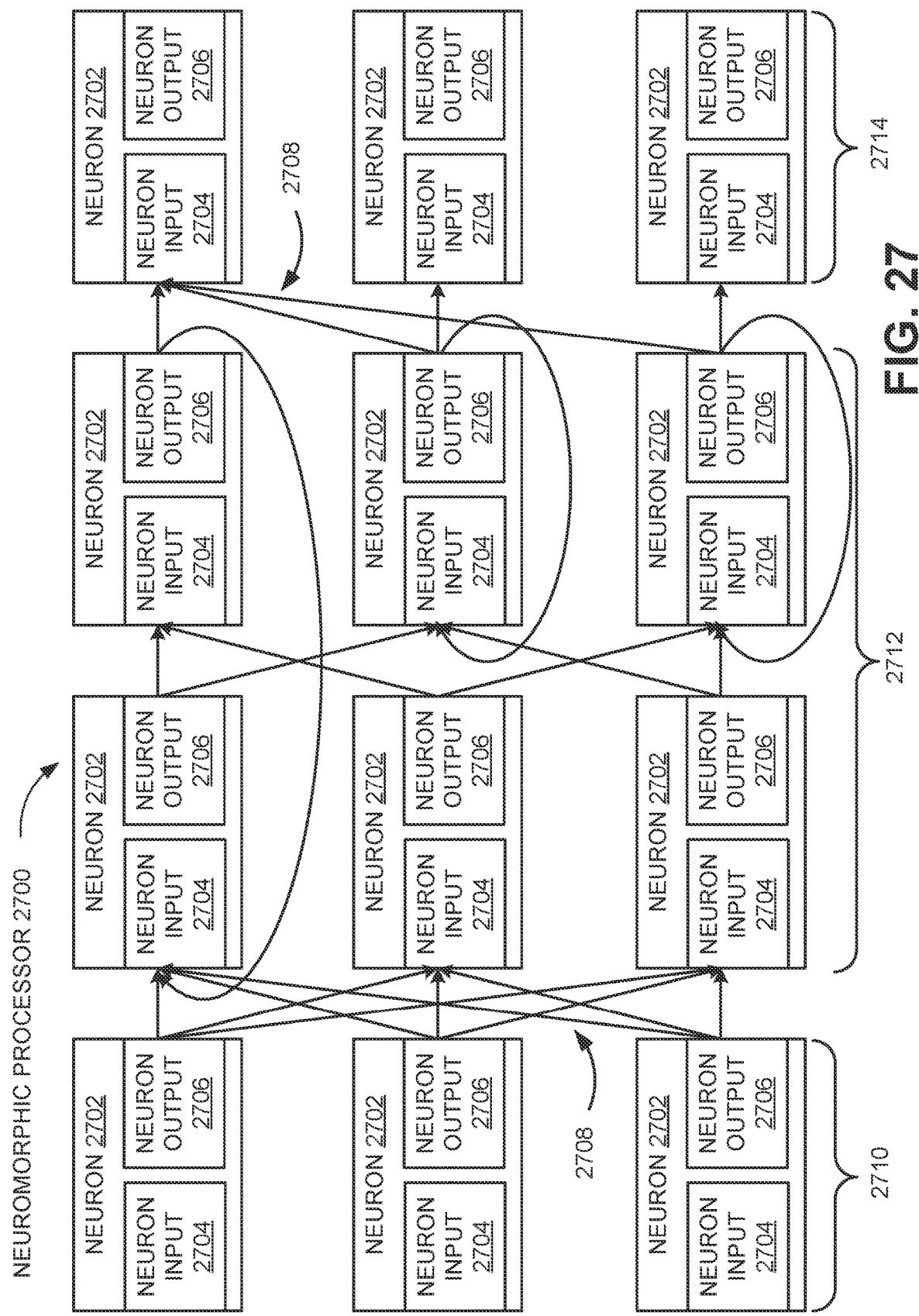


FIG. 26

**FIG. 27****2712****2710**

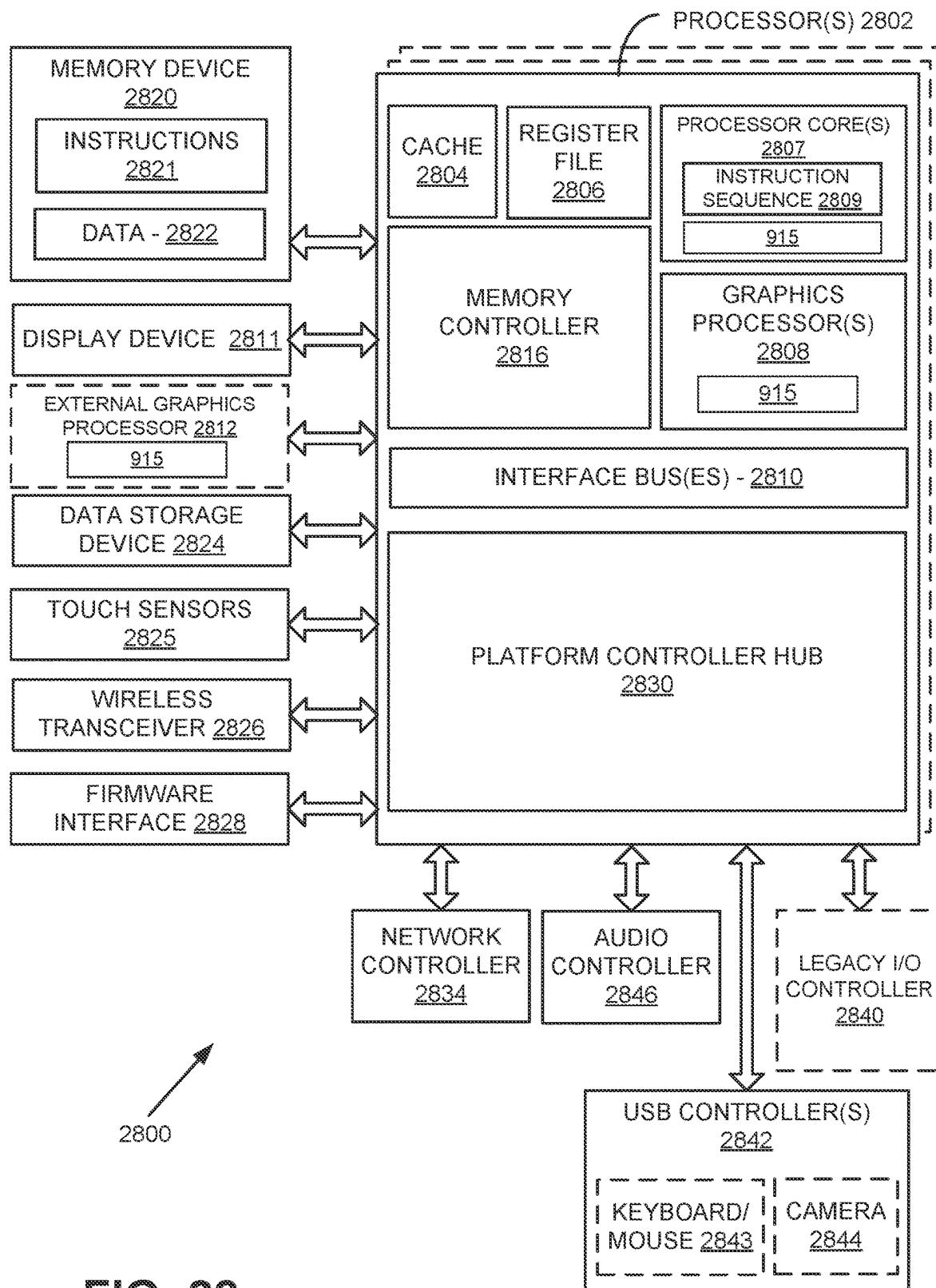


FIG. 28

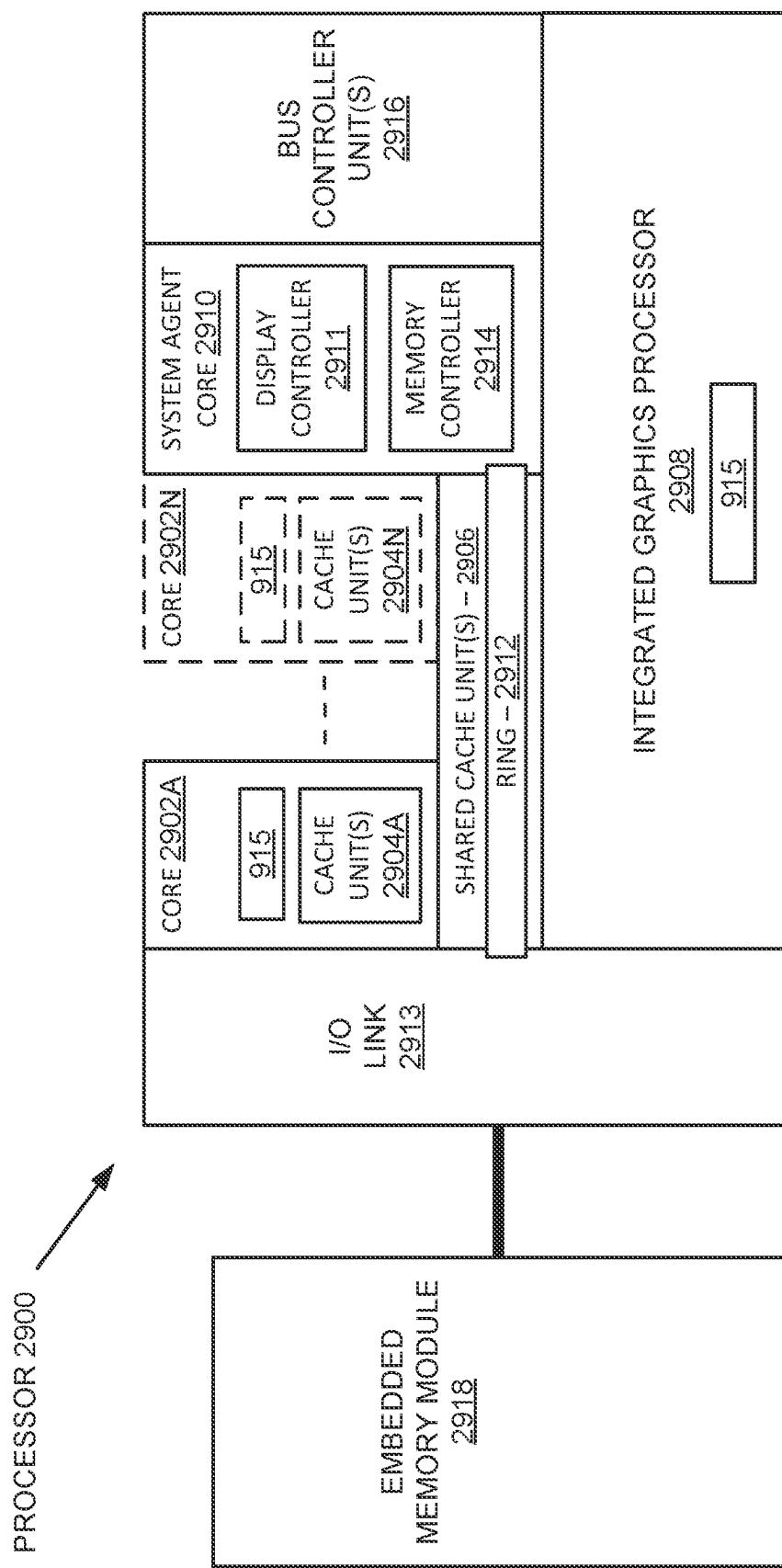


FIG. 29

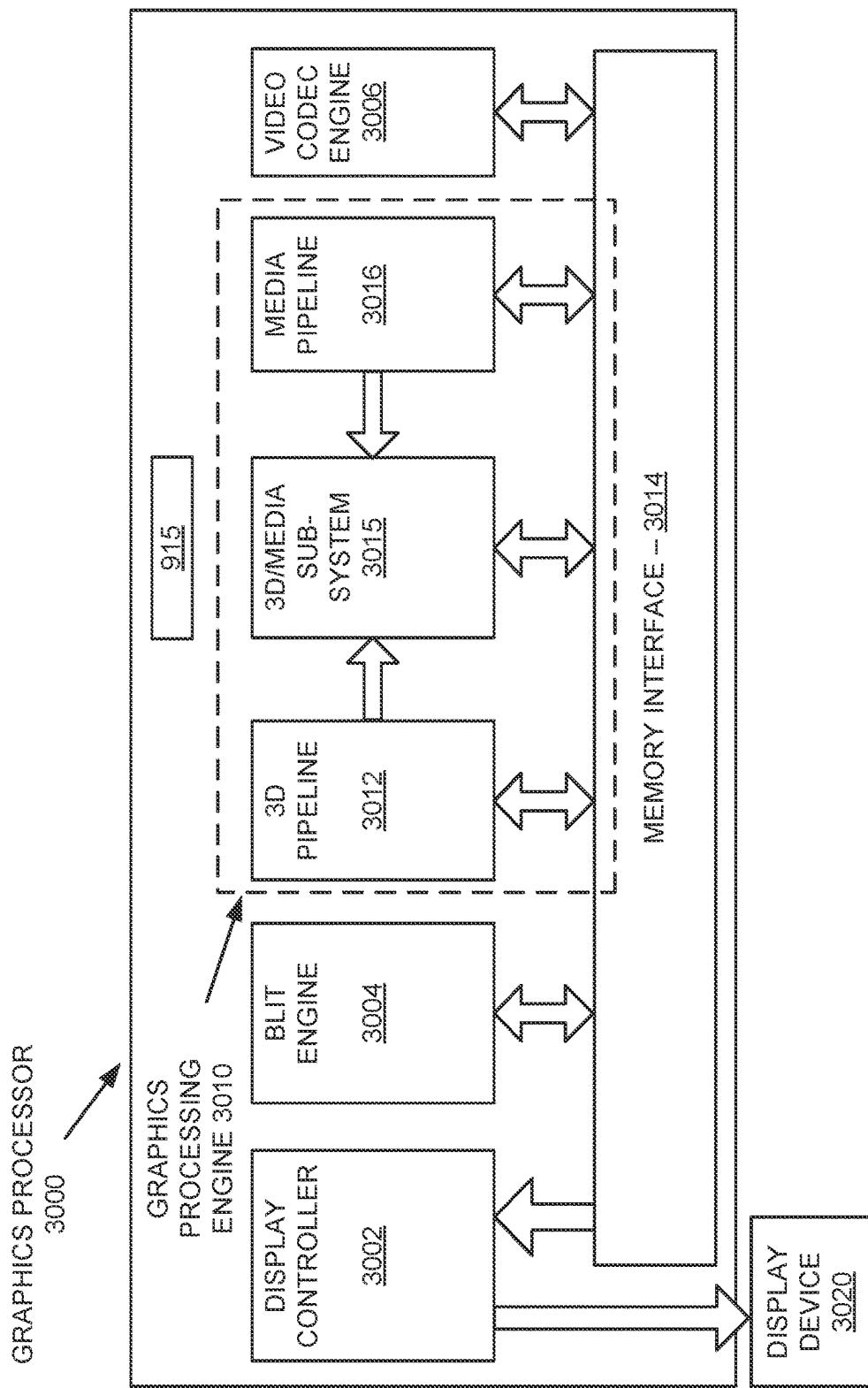


FIG. 30

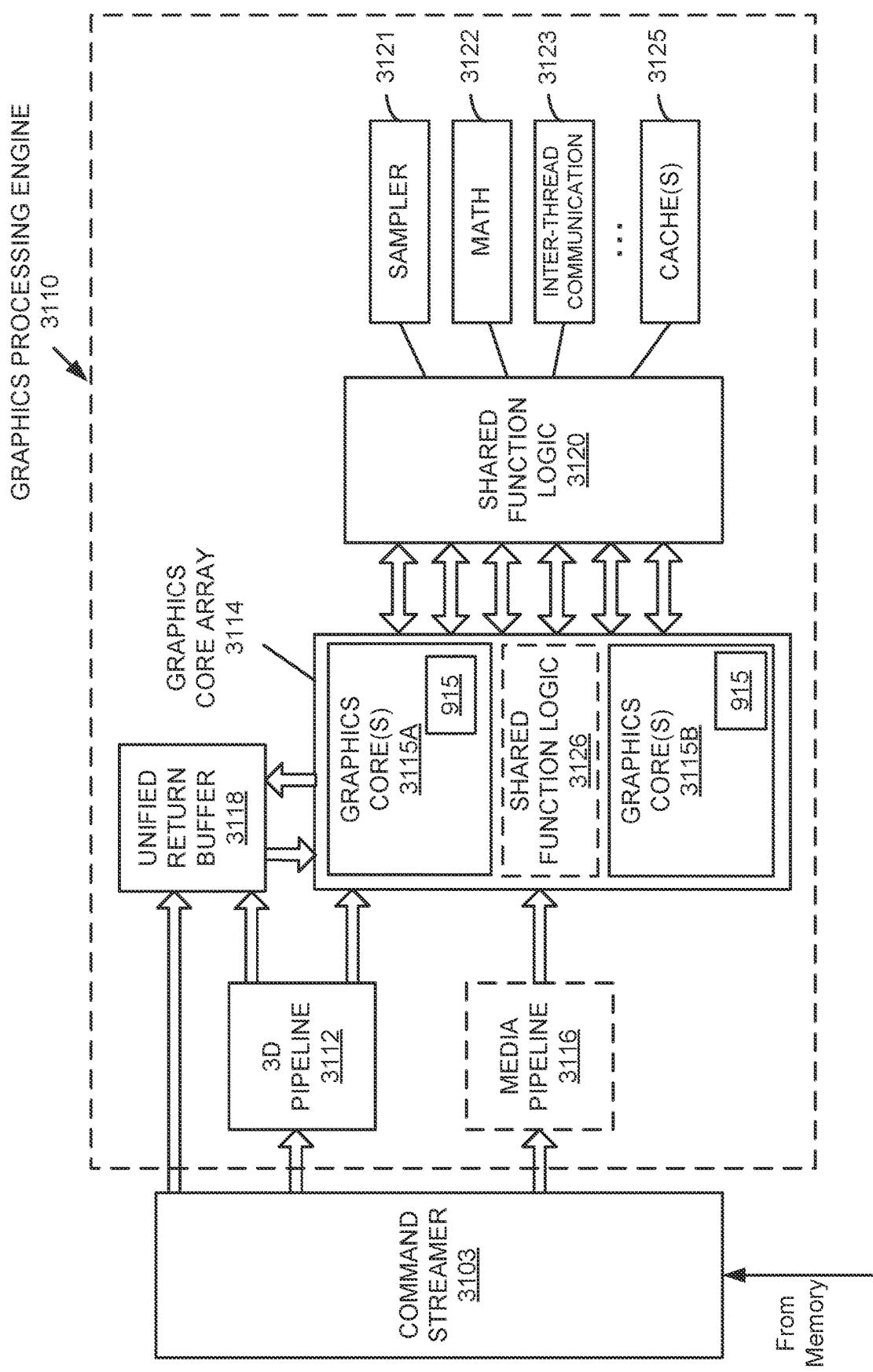
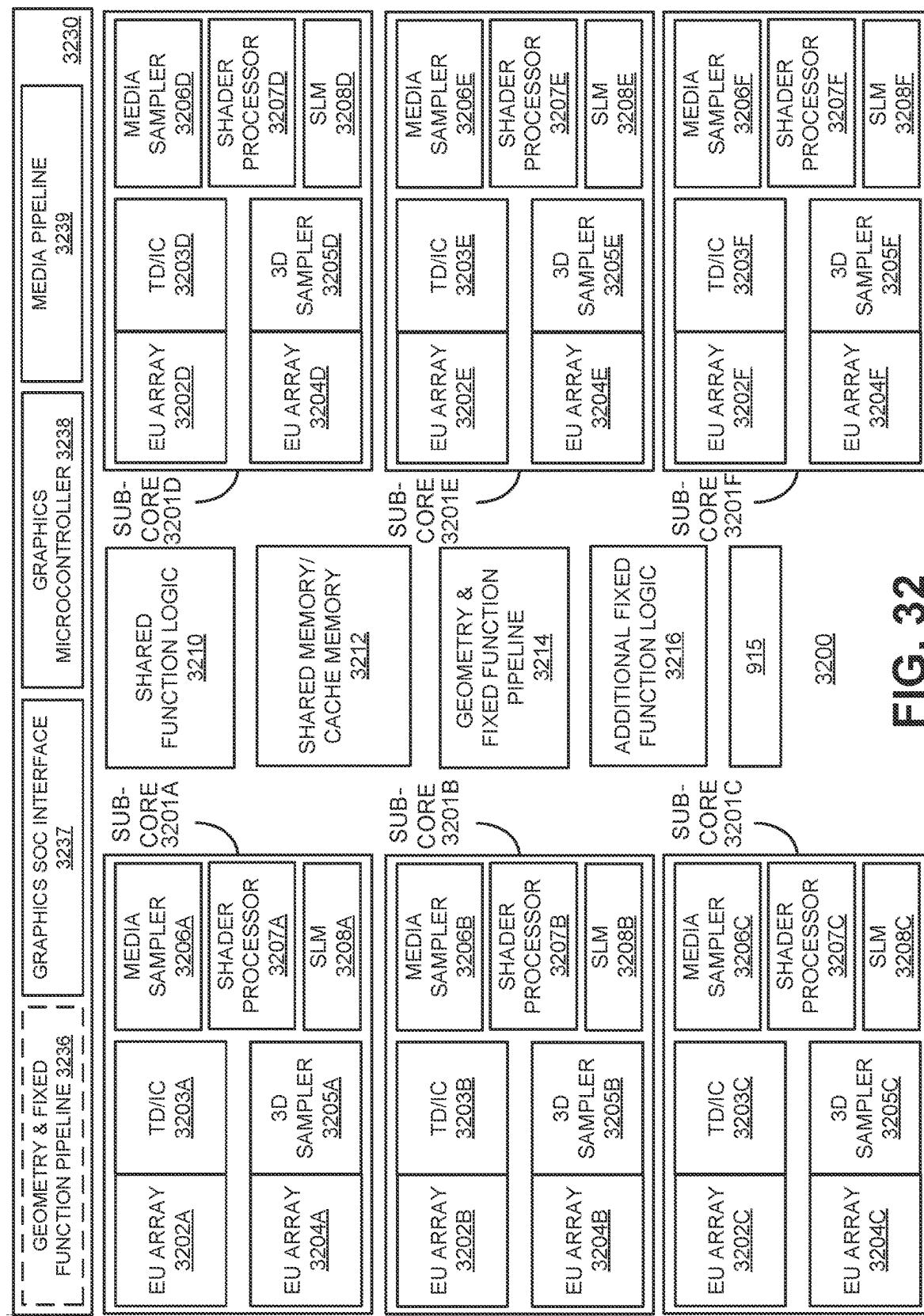


FIG. 31

**FIG. 32**

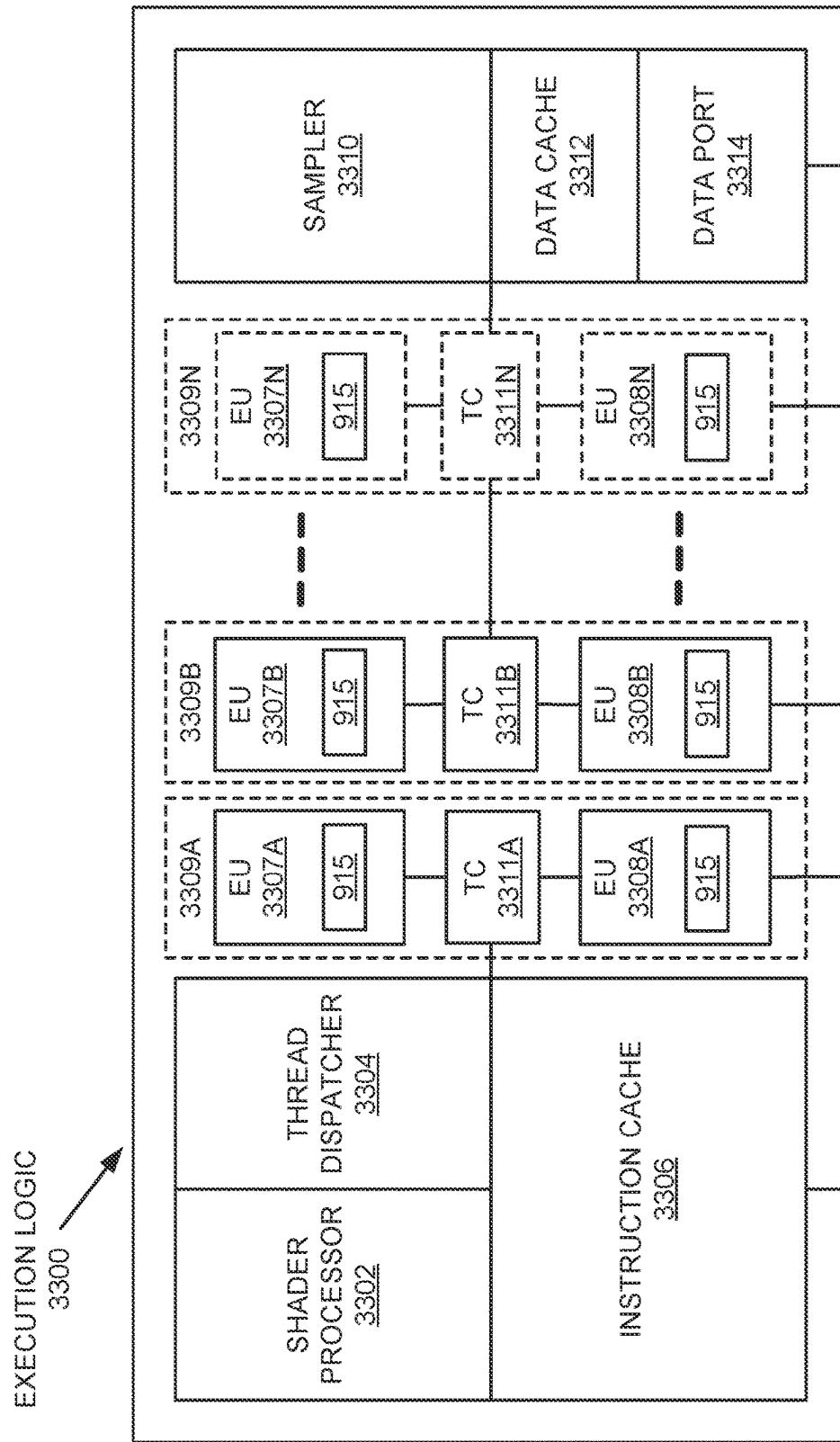
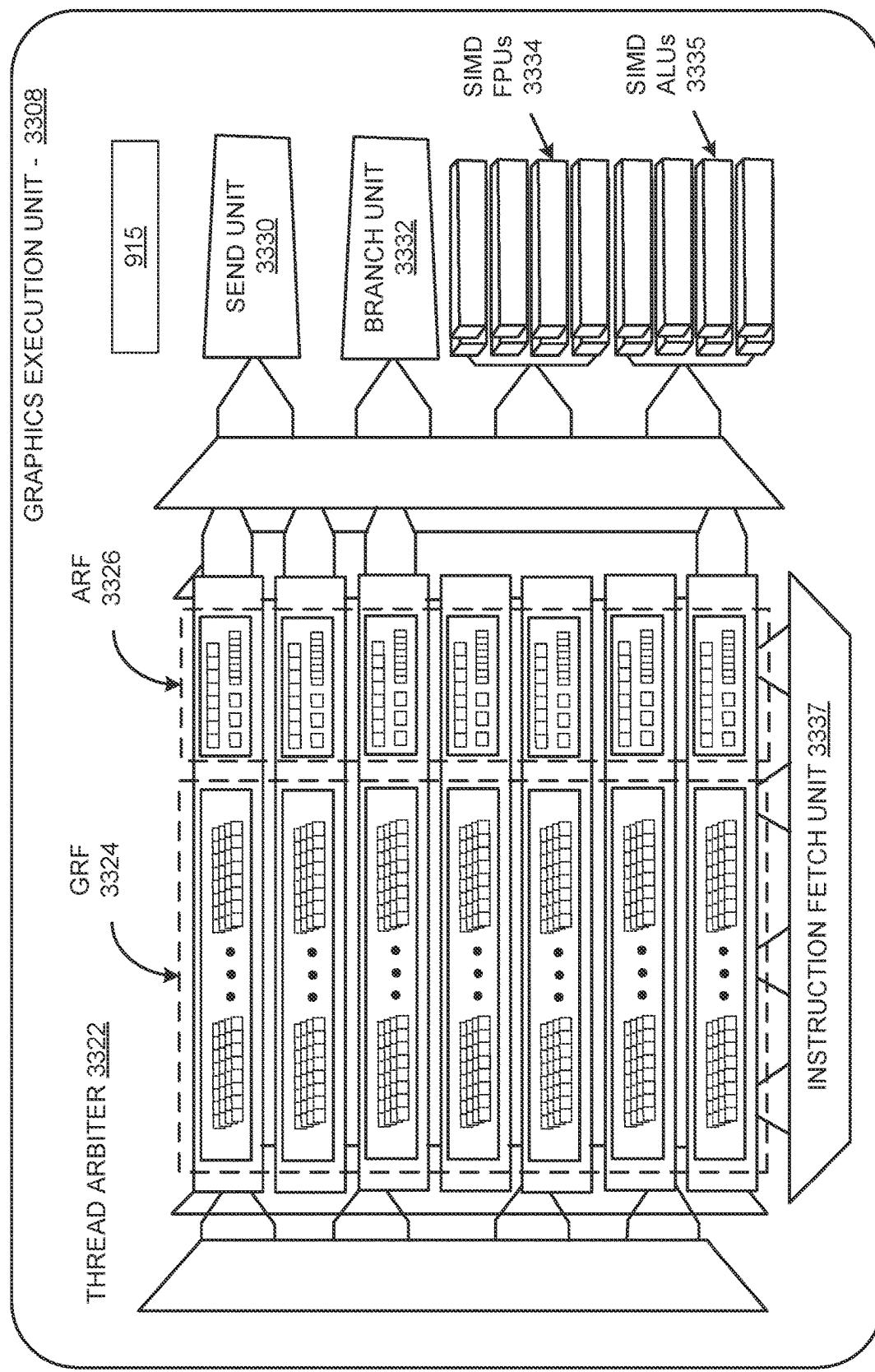


FIG. 33 A

**FIG. 33 B**

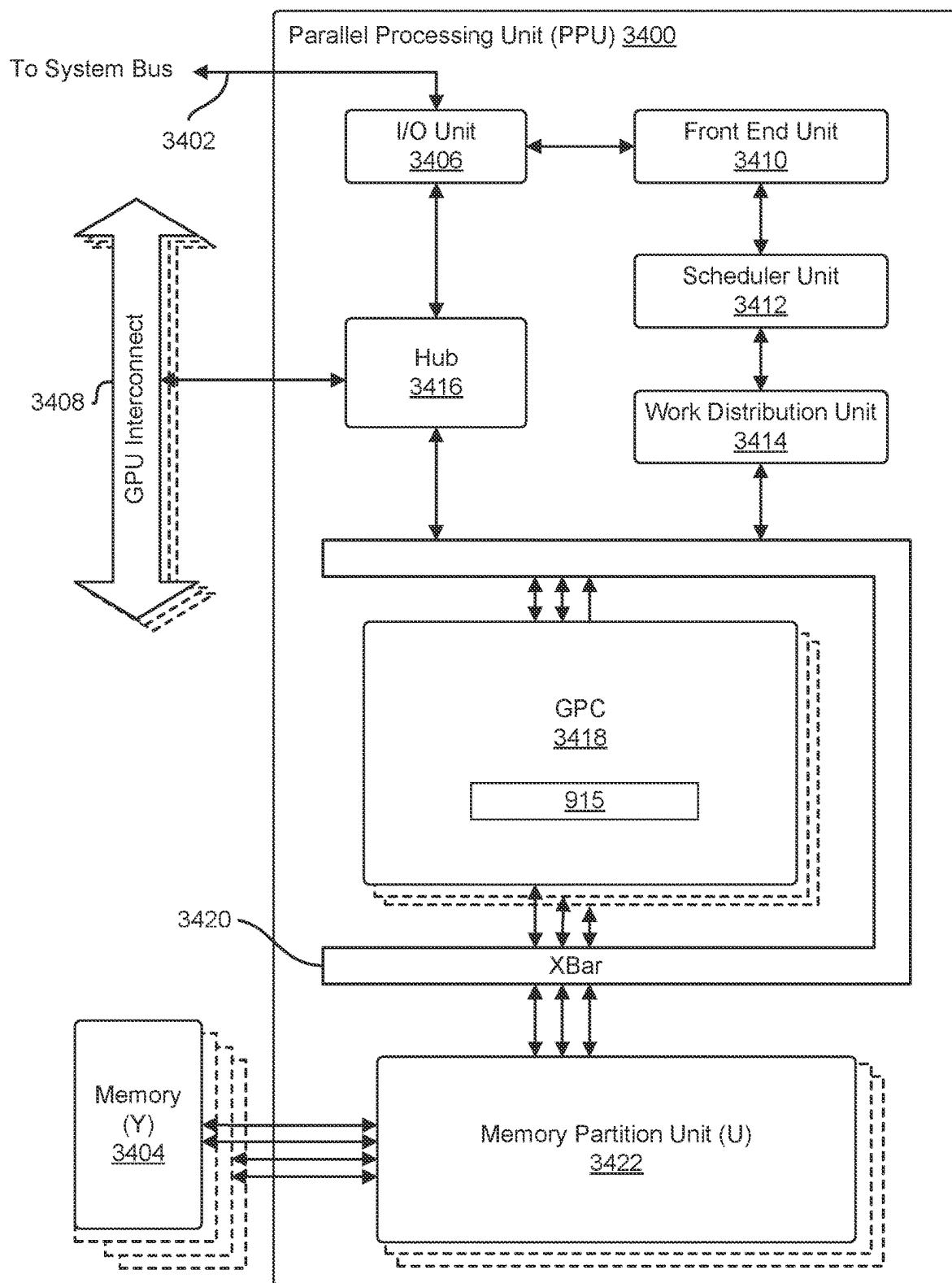


FIG. 34

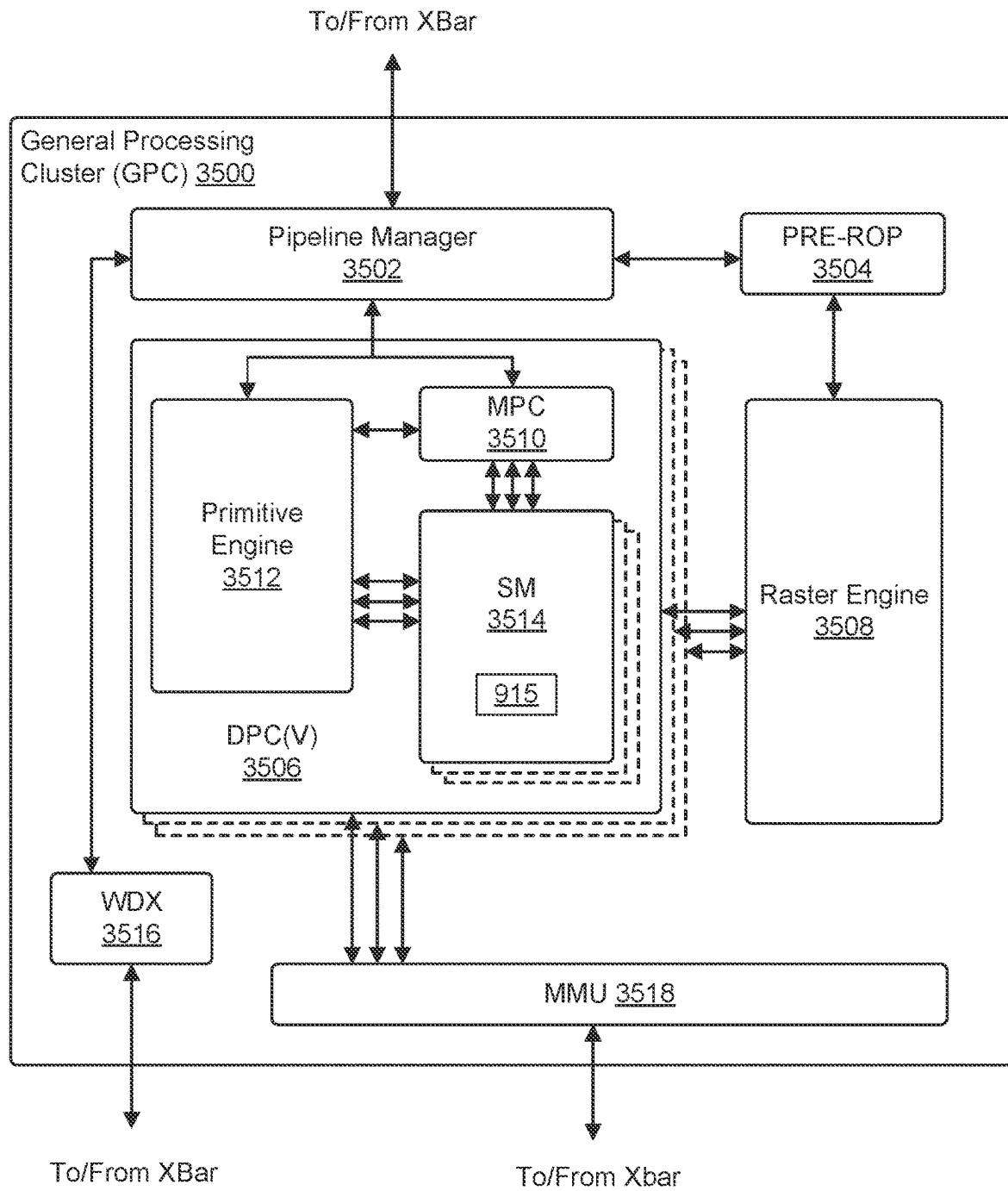
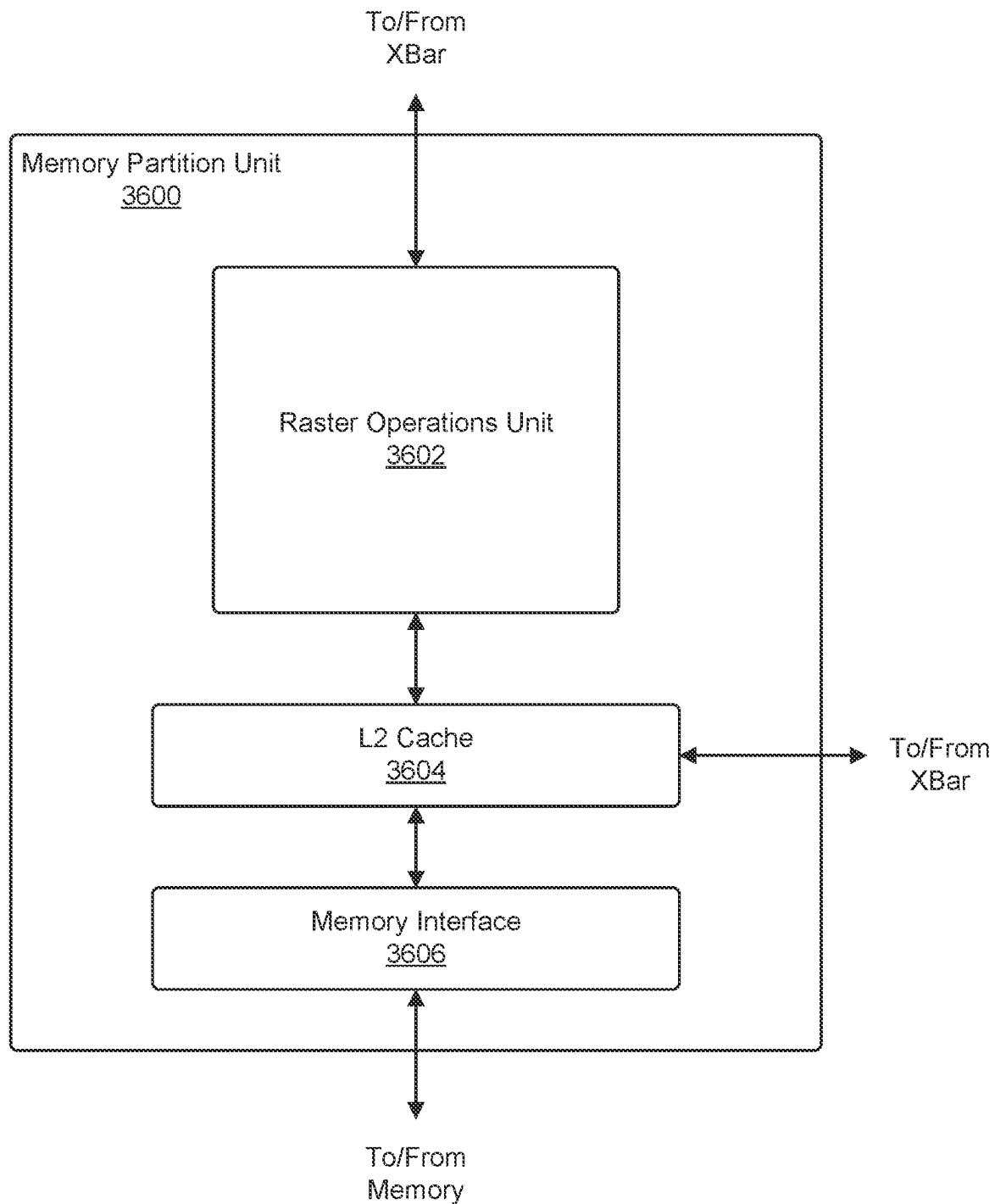
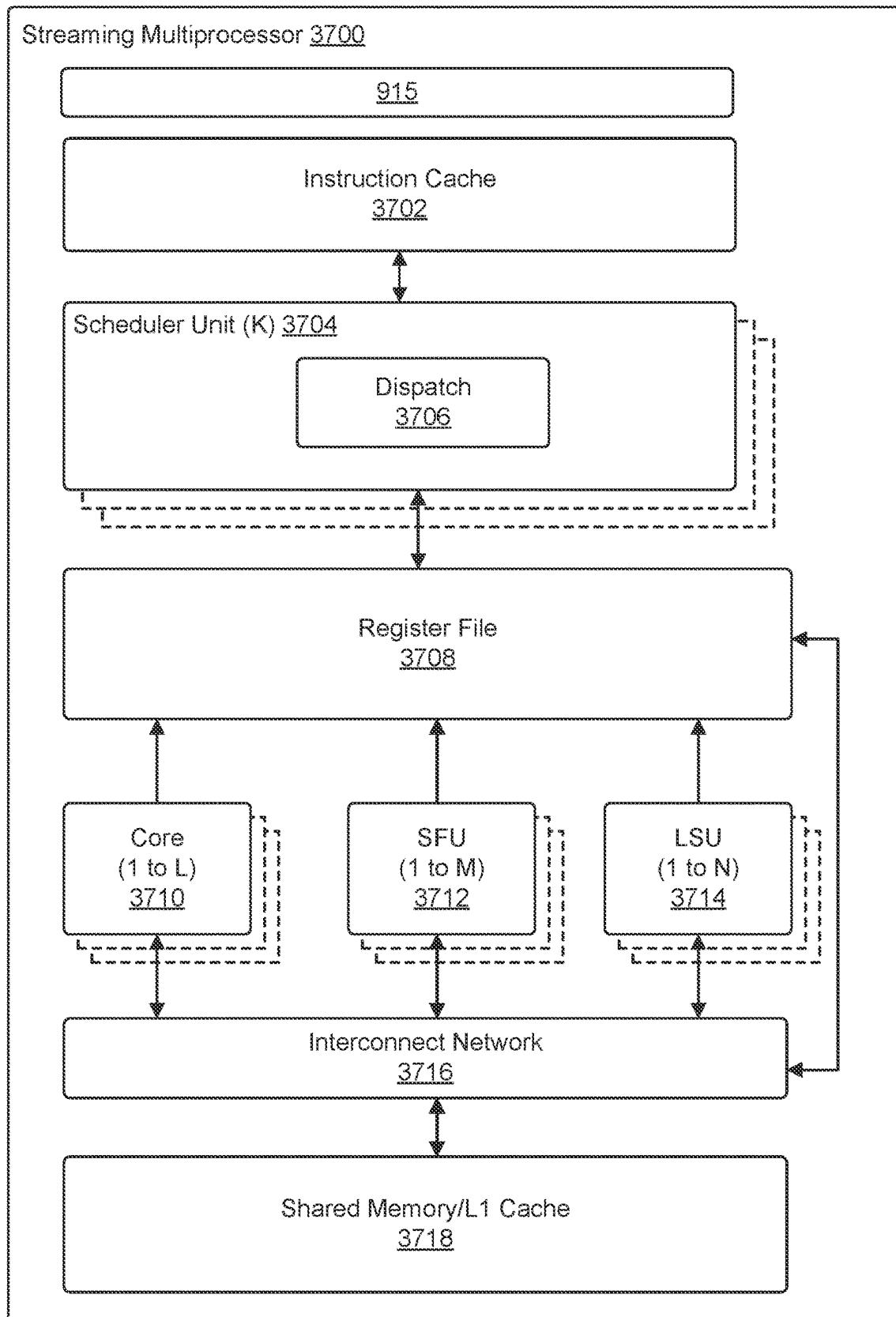


FIG. 35

**FIG. 36**

**FIG. 37**

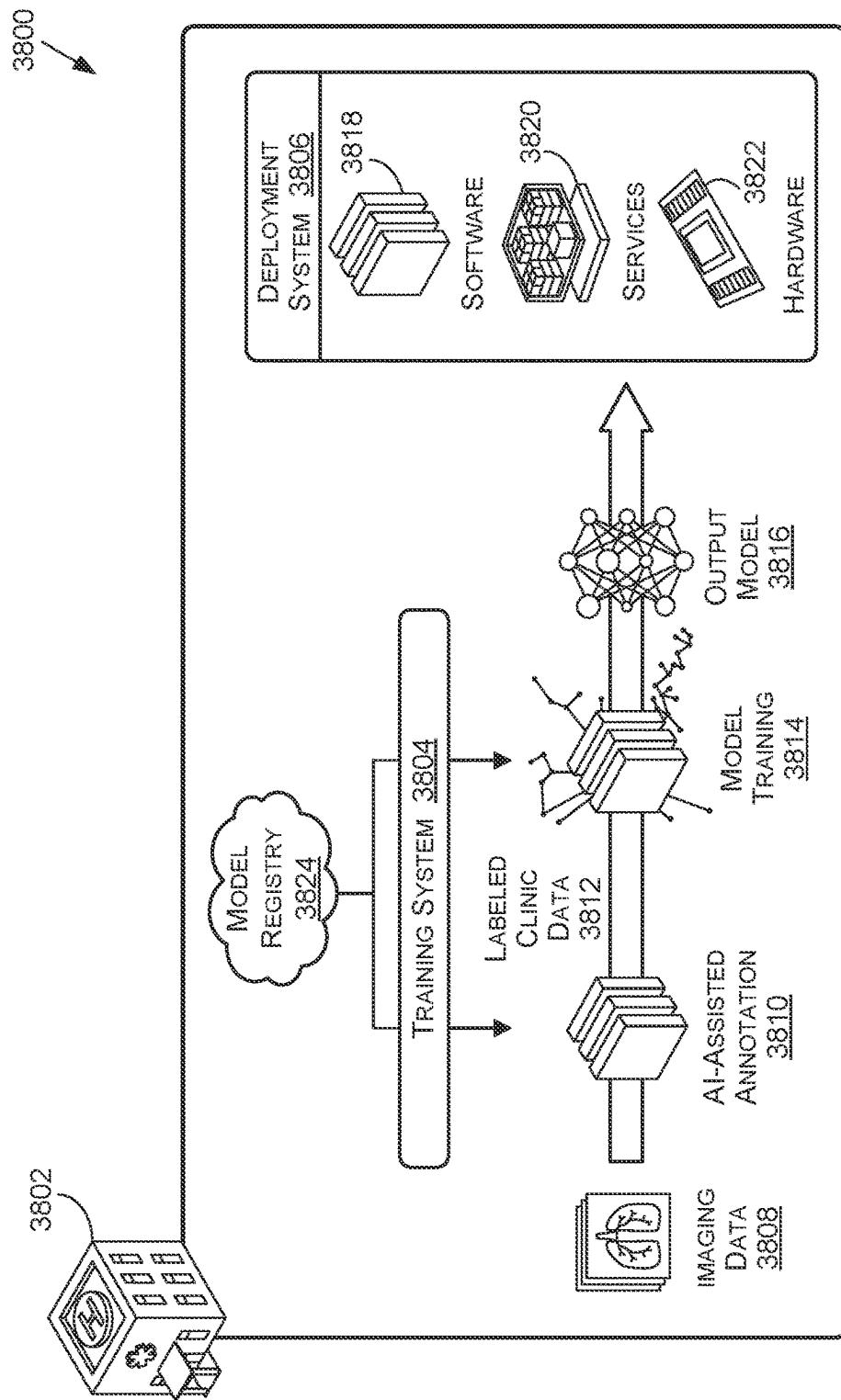
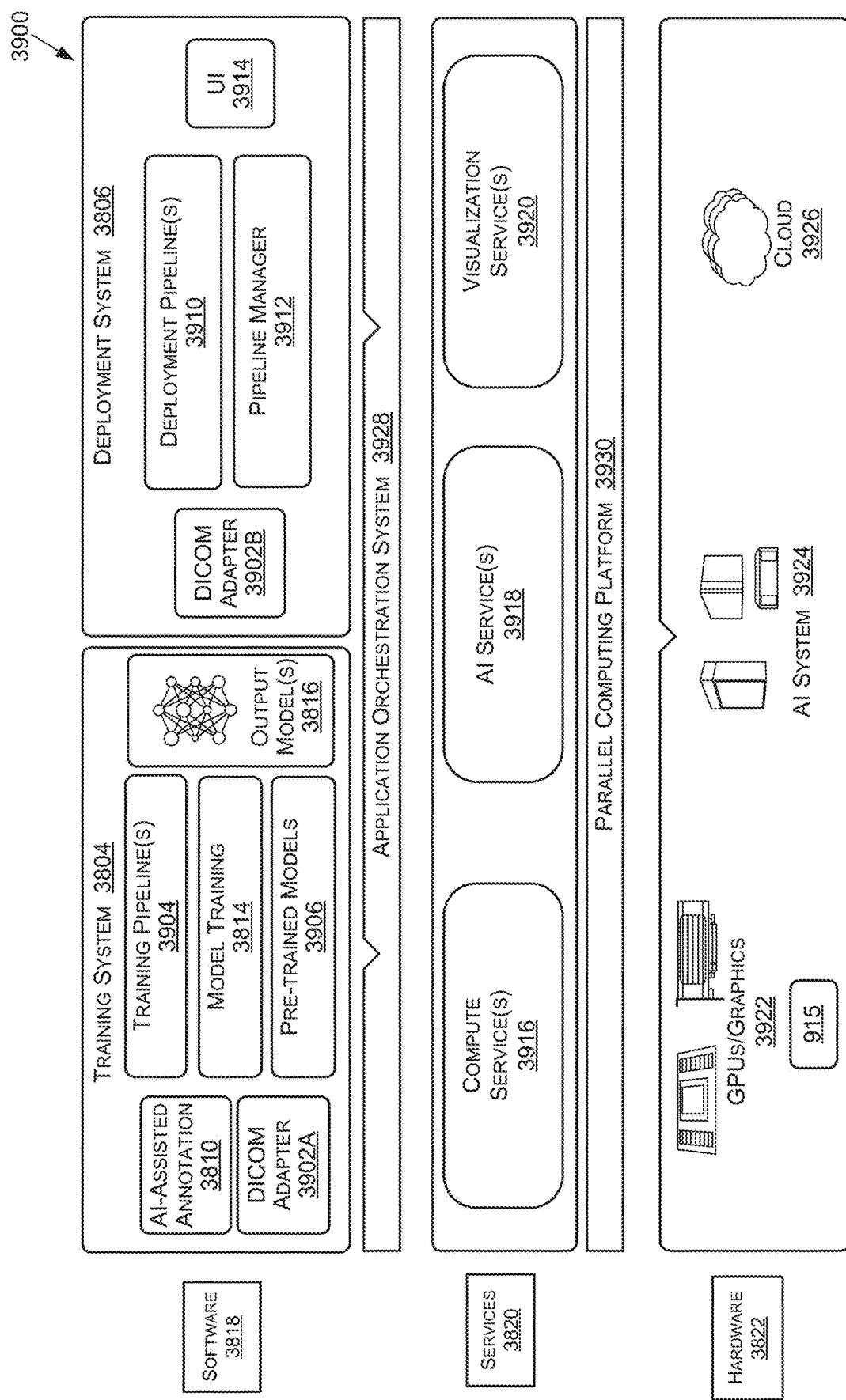


FIG. 38

**FIG. 39**

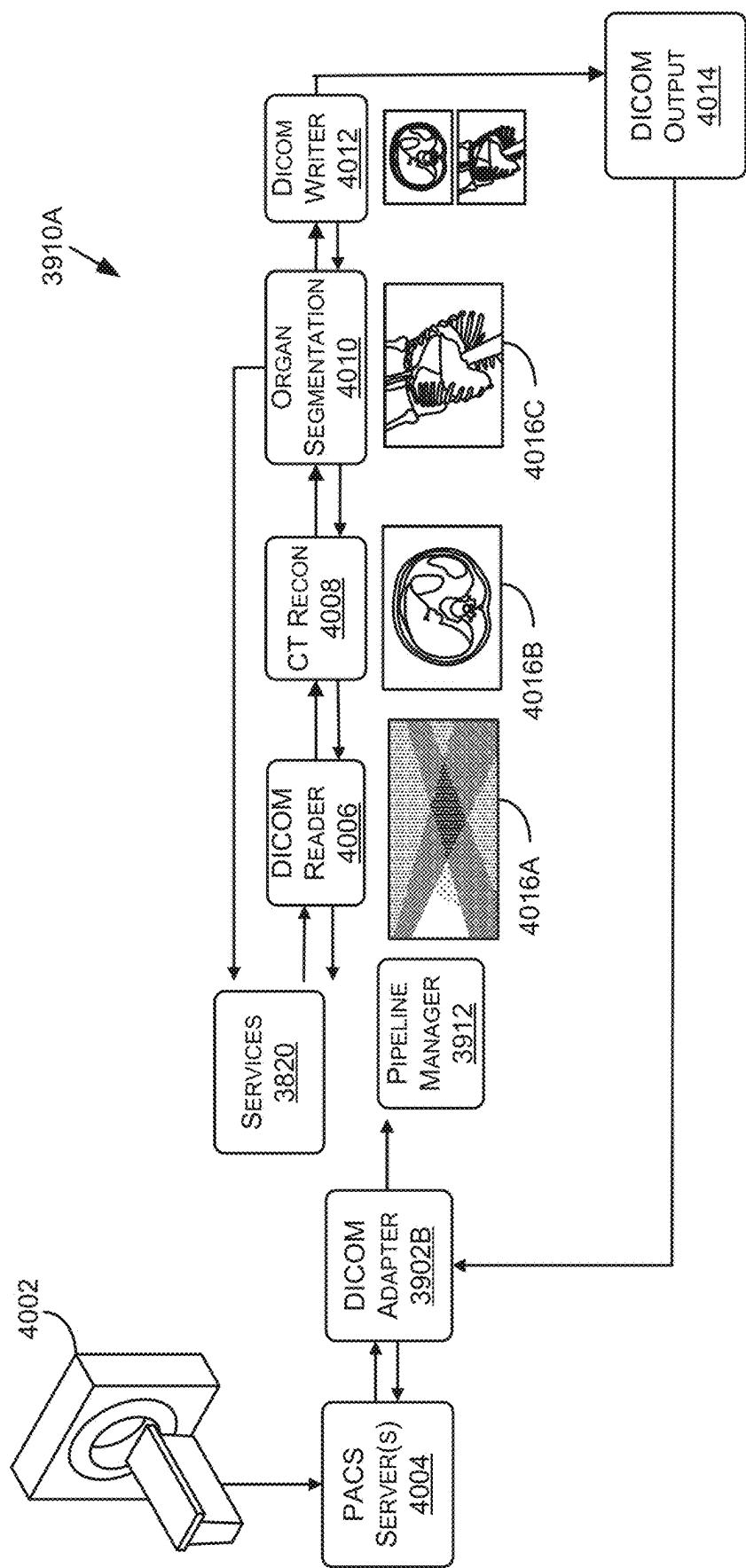
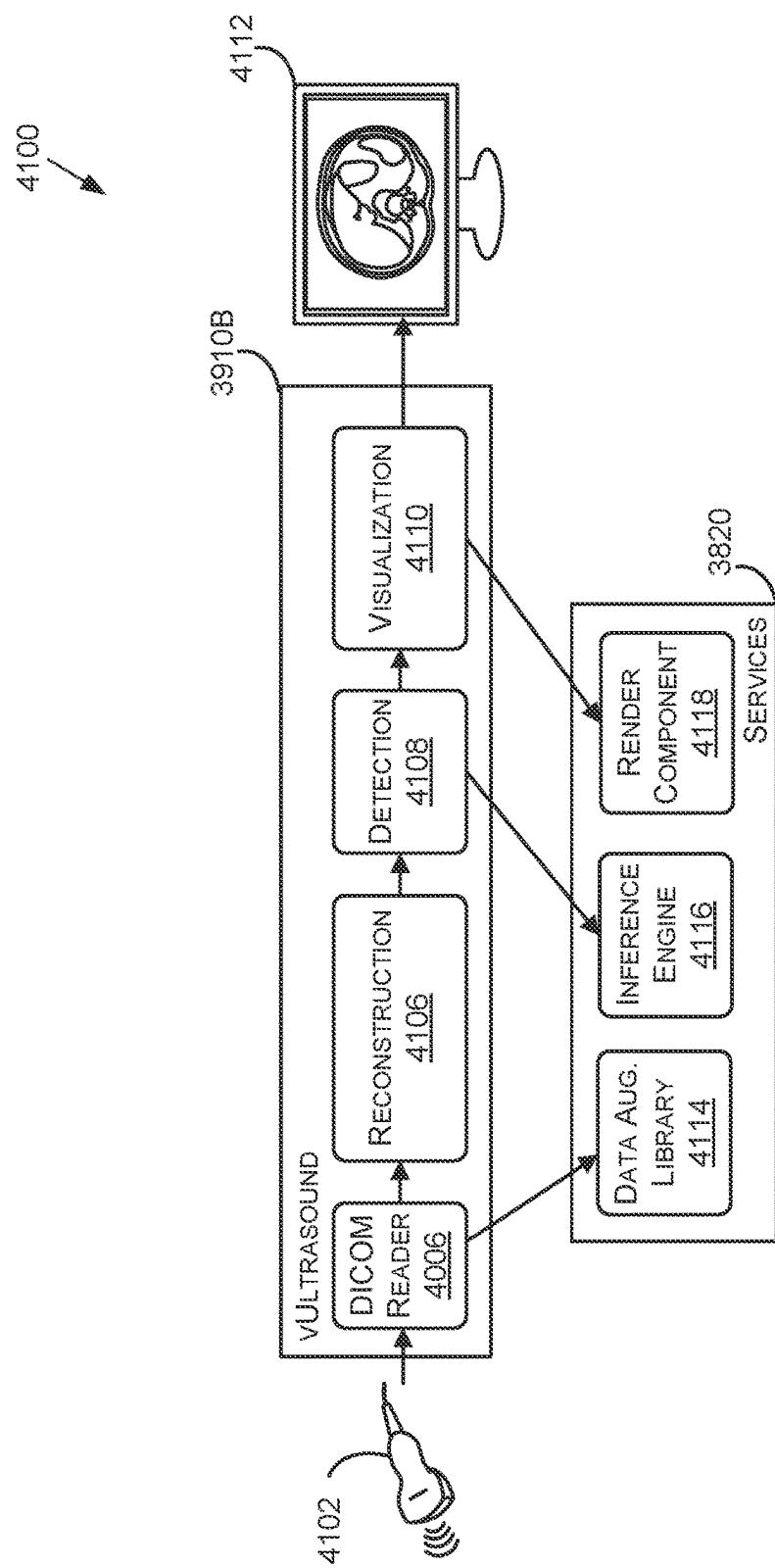


FIG. 40

**FIG. 41A**

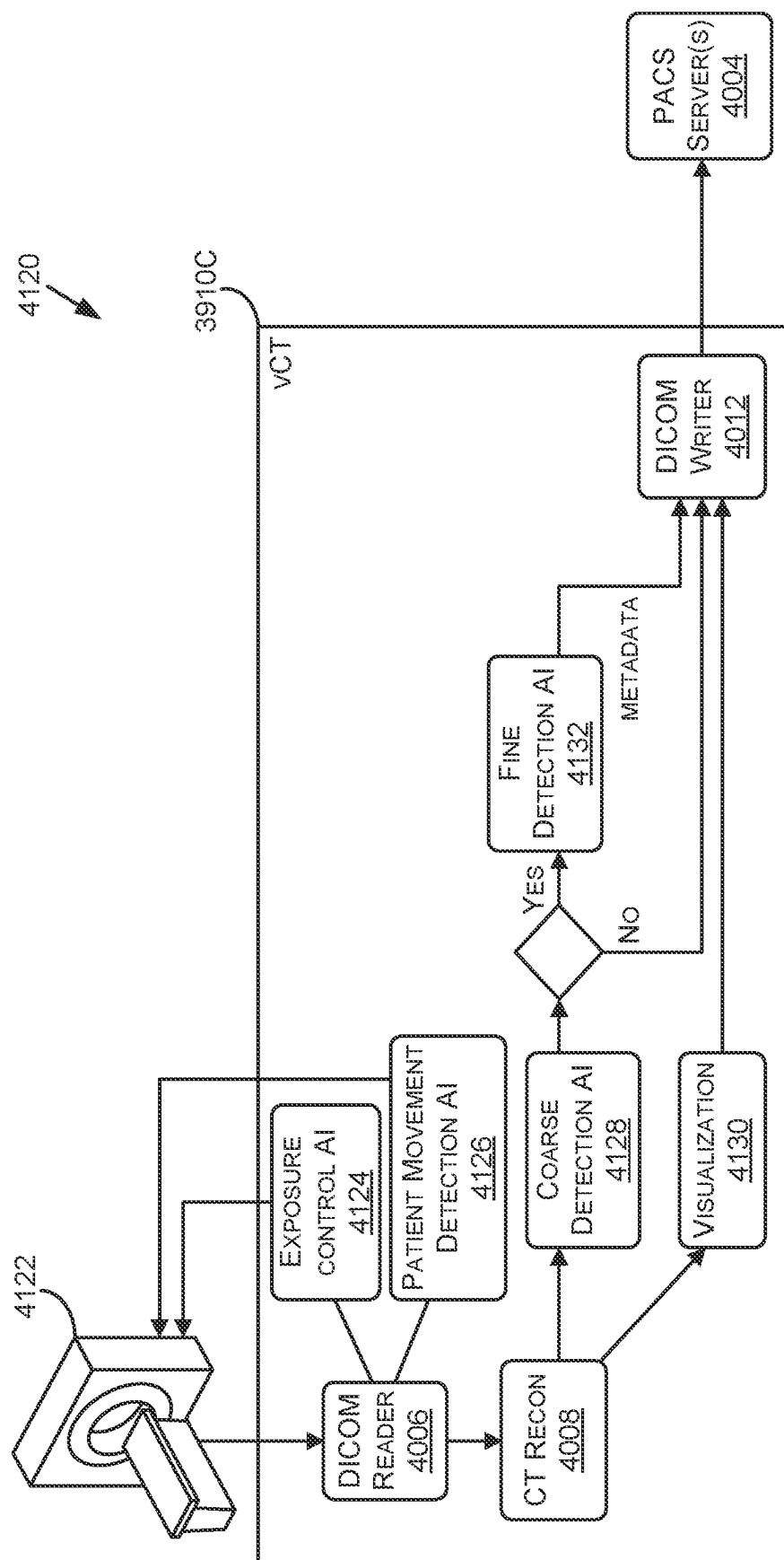


FIG. 41B

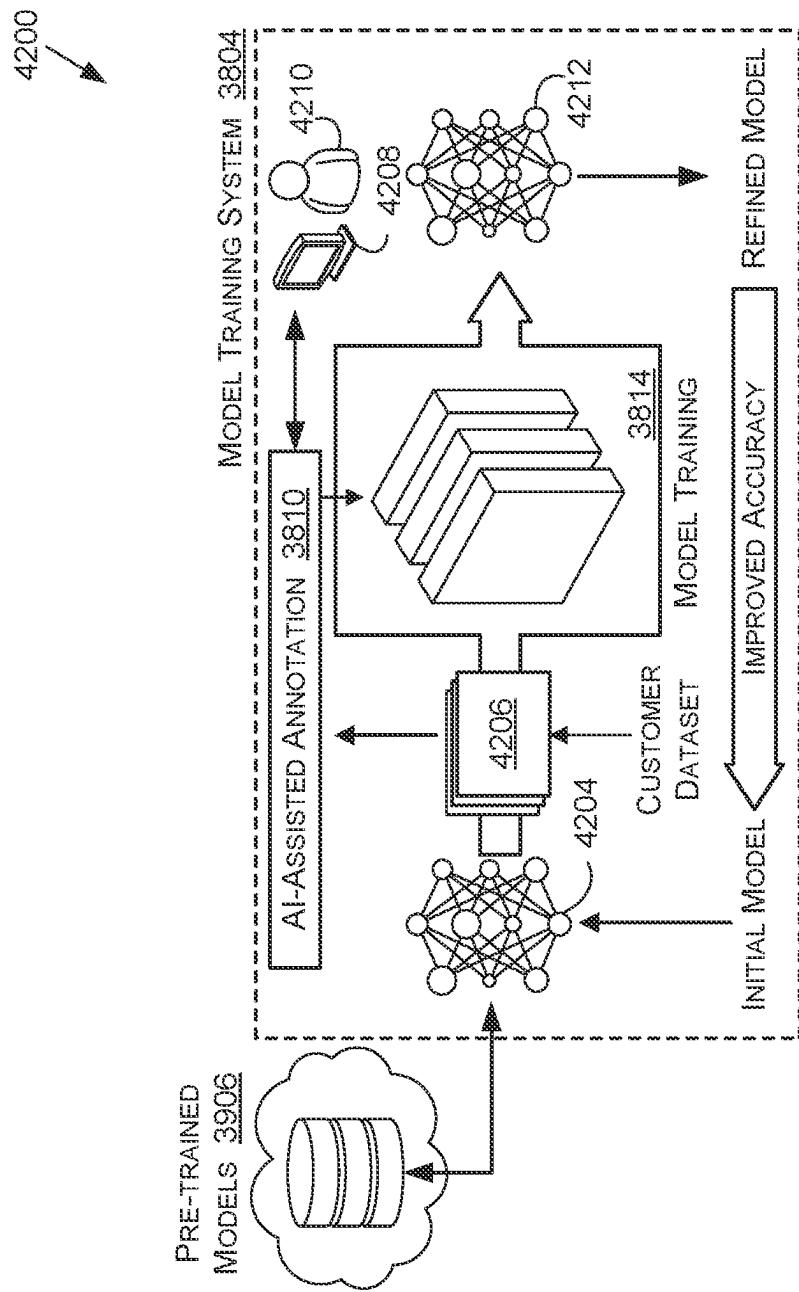


FIG. 42A

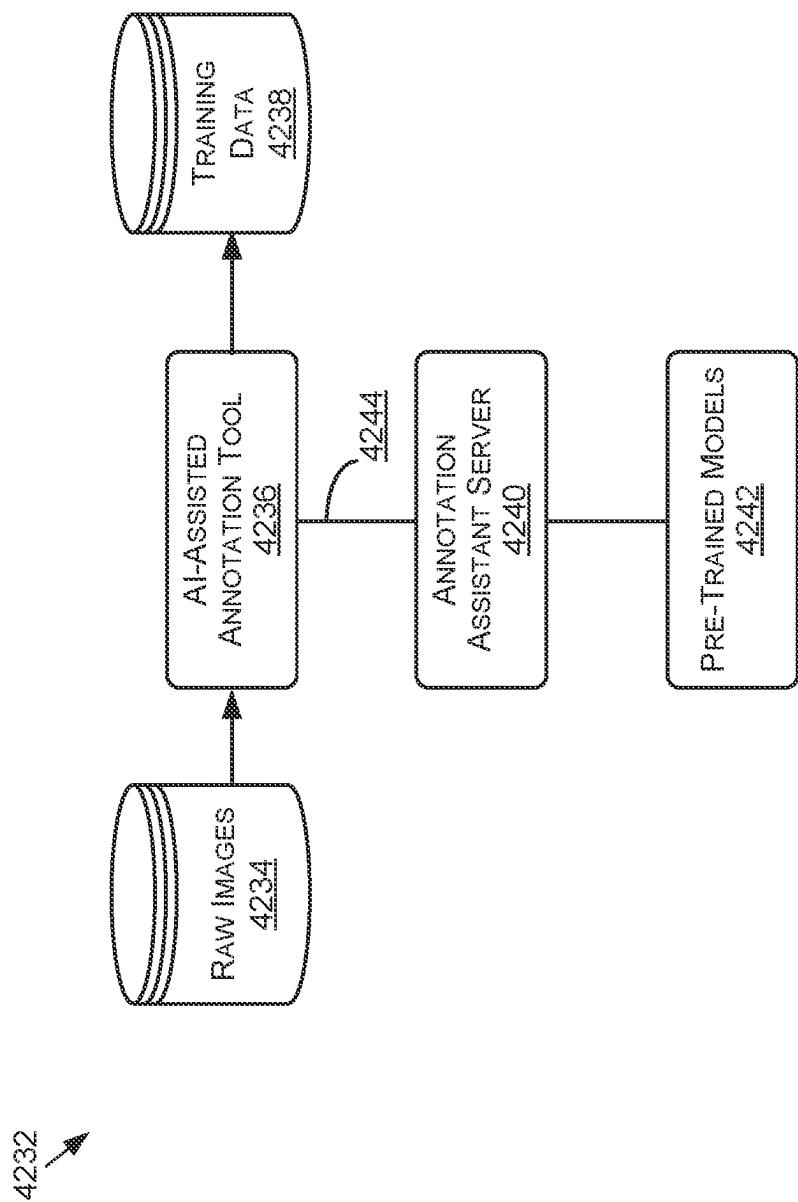


FIG. 42B

**1**
**AUGMENTING MULTIMEDIA STREAMING  
BY ANTICIPATING EVENTS USING  
ARTIFICIAL INTELLIGENCE**
**BACKGROUND**

As video game and other content and uses evolve, there is a corresponding need to adapt the ways in which content is used and made available. For example, videos of gameplay are increasingly streamed by a large number of viewers, and activities such as e-sports are changing the ways in which online gaming is developed and content presented to viewers. In many instances, these rapid changes have resulted in the presentation of content that is less than optimal for its intended purpose, or that requires manual intervention in order to create content of interest. In addition to the increase in resources, such manual intervention often increases the amount of time before such content can be presented to an audience of viewers or gamers.

**BRIEF DESCRIPTION OF DRAWINGS**

FIG. 1 illustrates an example of augmenting broadcasts based at least in part on anticipated in-game events in accordance with an embodiment;

FIG. 2 illustrates an example of a streaming gameplay session in which in-game events are detected in accordance with an embodiment;

FIG. 3 illustrates an example of a game server augmenting a broadcast with dynamically generated assets in response to anticipated in-game events in accordance with an embodiment;

FIG. 4 illustrates an example of an analyzer anticipating in-game events and determining occurrence of the in-game events in accordance with an embodiment;

FIG. 5 illustrates an example of an environment in which in-game events for gameplay sessions are determined based at least in part on user inputs in accordance with an embodiment;

FIG. 6 illustrates an example of an augmented broadcast of a gameplay session including dynamically generated assets in accordance with an embodiment;

FIG. 7 illustrates a process for augmenting broadcasts based at least in part on anticipated in-game events in accordance with an embodiment;

FIG. 8 is an example system diagram for a game streaming system, in accordance with some embodiments of the present disclosure;

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

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

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

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

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

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

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

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

**2**

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

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

5 FIG. 15 illustrates a computer system, according to at least one embodiment;

FIG. 16 illustrates a computer system, according to at least one embodiment;

10 FIG. 17A illustrates a computer system, according to at least one embodiment;

FIG. 17B illustrates a computer system, according to at least one embodiment;

15 FIG. 17C illustrates a computer system, according to at least one embodiment;

FIG. 17D illustrates a computer system, according to at least one embodiment;

FIGS. 17E and 17F illustrate a shared programming model, according to at least one embodiment;

20 FIG. 18 illustrates exemplary integrated circuits and associated graphics processors, according to at least one embodiment;

FIGS. 19A-19B illustrate exemplary integrated circuits and associated graphics processors, according to at least one embodiment;

25 FIGS. 20A-20B illustrate additional exemplary graphics processor logic according to at least one embodiment;

FIG. 21 illustrates a computer system, according to at least one embodiment;

30 FIG. 22A illustrates a parallel processor, according to at least one embodiment;

FIG. 22B illustrates a partition unit, according to at least one embodiment;

35 FIG. 22C illustrates a processing cluster, according to at least one embodiment;

FIG. 22D illustrates a graphics multiprocessor, according to at least one embodiment;

FIG. 23 illustrates a multi-graphics processing unit (GPU) system, according to at least one embodiment;

40 FIG. 24 illustrates a graphics processor, according to at least one embodiment;

FIG. 25 is a block diagram illustrating a processor micro-architecture for a processor, according to at least one embodiment;

45 FIG. 26 illustrates a deep learning application processor, according to at least one embodiment;

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

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

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

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

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

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

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

FIG. 34 illustrates a parallel processing unit (“PPU”), according to at least one embodiment;

65 FIG. 35 illustrates a general processing cluster (“GPC”), according to at least one embodiment;

FIG. 36 illustrates a memory partition unit of a parallel processing unit (“PPU”), according to at least one embodiment;

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

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

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

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

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

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

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

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

#### DETAILED DESCRIPTION

Embodiments of the present disclosure relate to augmenting broadcasts based at least in part on anticipated events (e.g., team win, goal score, flag capture, etc.). Systems and methods are disclosed that utilize data associated with a broadcast (e.g., a gameplay streaming session) to determine and/or predict an occurrence of an event and/or event type within an interval of time in order to augment the broadcast (e.g., to highlight important in-game events and increase audience interest). In various embodiments, gameplay streaming services and other streaming services utilize overlays (e.g., titles, scores, graphics, etc.), effects (e.g., slow-motion, picture-in-picture, animations, camera effects, etc.), and/or audio (e.g., player audio, sound effects, music, etc.) to augment a broadcast of a particular streaming session. As described in greater detail below, a client device initiates a gameplay session with a server, in at least one embodiment. In addition, the server, in various embodiments, broadcasts the gameplay session to one or more additional client devices spectating the gameplay session. In various embodiments, the server captures media of the gameplay session (e.g., audio and video generated during the gameplay session) and augments the media to generate a modified broadcast.

Furthermore, as described in greater detail below, a dynamic asset generator causes particular assets (e.g., overlays, effects, and/or audio) to be included in the broadcast in order to increase viewer engagement, interactivity, and generally augment the broadcast, in accordance with at least one embodiment. In various embodiments, the dynamic asset generator includes one or more assets in the broadcast in response to a soft analyzer predicting an occurrence of an event within an interval of time. In one example, the dynamic asset generator applies a slow-motion effect to a broadcast of a gameplay streaming session in response to the soft analyzer indicating a prediction of a particular in-game event and/or type of in-game event occurring within an

interval of time (e.g., 5 seconds). In various embodiments, the soft analyzer obtains user input data available from user input devices to determine and/or predict an occurrence of an event and/or event type within an interval of time. In one example, the soft analyzer obtains control device inputs (e.g., keystrokes from a keyboard) and determines, based at least in part on the control device inputs, a likelihood of an event occurring. In various embodiments, the soft analyzer continuously generates predictions based at least in part on game data obtained from a client device and/or server.

In addition, in an embodiment, a hard analyzer is used to confirm the occurrence of the event and indicate to the dynamic asset generator the event has occurred. In an example, the soft analyzer generates a prediction of an in-game event (e.g., an in-game gun battle) and indicates a likelihood (e.g., a confidence score) of the in-game gun battle occurring within 3 seconds, the hard analyzer then analyzes game data to confirm the occurrence of the in-game gun battle and indicates the occurrence of the event to the dynamic asset generator which may update a state of the asset included in a broadcast of gameplay. Furthermore, in some examples, the hard analyzer confirms the occurrence of in-game events after they have occurred. In at least one embodiment, media representative of a game session can be analyzed by the hard analyzer. This media may include, for example, an audio and/or video stream of a game session, audio and/or video data captured of players, audio and/or video data captured of the game application, or other data associated with the game. The media can be analyzed to detect or identify the presence of specific occurrences or events in the game in accordance with various embodiments. The hard analyzer, in at least one embodiment, periodically or aperiodically analyzes the media to determine whether the event occurred. In one example, in response to the soft analyzer indicating an event will occur within 5 seconds, the hard analyzer analyzes the media once (or more) per second until 5 seconds have expired and/or occurrence of the event is detected.

In an embodiment, the dynamic asset generator, in response to the hard analyzer indicating the event has occurred, updates the state of the asset included in the broadcast. In one example, updating the state of the asset includes removing the asset from the broadcast. In another example, updating the state of the asset includes updating information (e.g., leaderboard, statistics, etc.) included in an overlay within the broadcast. Furthermore, if at the expiration of the interval of time (e.g., the interval of time the soft analyzer indicated the event will occur within) expires, in various embodiments, the dynamic asset generator updates the state of the asset. For example, if the interval of time elapses without an indication from the hard analyzer that the event occurred, the dynamic asset generator discards the asset and/or causes display of the asset within the broadcast to be terminated.

FIG. 1 illustrates an example of an augmented broadcast 108 to a remote audience 104, where the augmented broadcast 108 is generated based at least in part on anticipated in-game events (also referred to as events for simplicity) in accordance with an embodiment. In various embodiments, the augmented broadcast 108 includes audio and/or video data captured from a gameplay session of a star player 102 executed by a client device (e.g., client device 504(A)-504 (N) as described in greater detail below in connection with FIG. 5). Furthermore, event analysis and special effects 106 are used to generate the augmented broadcast in accordance with at least one embodiment. For example, in the case of online video hosting or streaming, the augmented broadcast

**108** is generated using event data, as described in greater detail below, and the augmented broadcast **108** can then be made available by a third party content provider, such as YouTube®, Twitch®, or Facebook Live® for streaming to the remote audience **104** (e.g., using GeForce Experience® from NVIDIA).

As described in greater detail below, a game server (e.g., the game server **526** described below in connection with FIG. 5) or other computer system analyzes the game broadcast and/or game data to determine events (e.g., durations of relatively high interest within a gameplay session) and then augments the game broadcast with dynamically generated assets to produce the augmented broadcast **108**. In various embodiments, as described in greater detail below with FIGS. 3 and 4, a soft analyzer predicts the occurrence of an event based at least in part on game data and a hard analyzer determines whether the predicted event occurred. In such embodiments, if the hard analyzer determines the event occurred (e.g., based at least in part on processing images from the game broadcast), the game broadcast is augmented using one or more special effects and/or assets to generate the augmented broadcast **108**.

Furthermore, in various embodiments, the soft analyzer identifies and/or determines a duration of events within a gameplay session. For example, the soft analyzer determines that within three seconds a particular event will occur. Furthermore, in such examples, this prediction and/or determination that the particular event will occur is associated with a confidence score. The disclosure also provides approaches for determining and/or predicting, in accordance with an embodiment, whether a duration of the event within the gameplay session would be of sufficient interest for further action (e.g., generating dynamic asset for modification of the game broadcast). In one example, the soft analyzer determines and/or predicts events of relatively high interest within a gameplay session and determines and/or predicts durations of the predicted events.

In embodiments, the star player **102** (e.g., user of the client device) plays a game, user inputs may be captured and used to generate (e.g., periodically during or after the gameplay session) a running user activity measurement (e.g., a metric) collected over an interval of time of the gameplay session. In various embodiments, user activity includes inputs provided by the star player **102** to an input device (e.g., mouse, keyboard, controller), biometric data (e.g., eye movement, heart rate, perspiration), activity of an in-game avatar associated with the star player **102**, or any other inputs that affect the gameplay session. The star player **102** activity measurement, for example, is used to determine and/or identify durations of relatively high user activity (e.g., relative to an amount of user activity during other intervals of the gameplay session) within the gameplay sessions. In some embodiments, durations of relatively high user activity correspond to a frequency of user input events (e.g., keys per second) generated by the user inputs. Once a duration is identified, it may be further analyzed to merge the duration with one or more other durations and/or to determine or predict whether the duration indicates the likelihood of an event and/or particular type of event in accordance with an embodiment. For example, a particular sequence of inputs or a particular frequency of inputs is indicative of a goal being scored in-game.

In some embodiments, the soft analyzer identifies an event based at least in part on an interval of time during which the user activity measurement exceeds a threshold value. For example, a start of the event corresponds to an interval in which the user activity measurement is greater

than the threshold value and/or an end of the event corresponds to an interval in which the user activity measurement is less than the threshold value. The threshold value, in an embodiment, corresponds to a statistic computed from values of the user activity measurement over the gameplay session and/or many gameplay sessions of the user and/or other users. For example, the statistic corresponds to an average or percentile of the user activity measurement over the gameplay session(s). As described in greater detail below in connection with FIG. 5, equation (1) is used, in various embodiments, to predict the occurrence of the event based at least in part on user activity.

Furthermore, in various embodiments, the soft analyzer, by focusing the analysis on durations of the gameplay session that include relatively high user activity, reduces an amount of computing resources used to identify the event, as analysis of video data is not required (although it may be used in some embodiments). Also, in examples where the user activity measurement is monitored during the gameplay session, the events are identified and actions performed (e.g., dynamic generation of assets used to modify the broadcast) during the gameplay session. Thus, for example, the game server or other computer system uses the assets to effectively increase entertainment and/or engagement of the remote audience **104** and/or to begin or complete actions to prepare for completion of the broadcast (e.g., the process of identifying game highlights).

In some embodiments, the hard analyzer performs further analysis on the event, such as by applying image and/or video data of the event to an object detection model or other machine learning model (e.g., deep learning and/or convolutional neural network models). For example, the object detection model is used to identify one or more objects in a scene, and based on the one or more objects, the game server or other computer system determines that the event has occurred during the gameplay session. In other embodiments, the event is identified to a user to assist the user in generating and/or selecting ground truth data (e.g., corresponding to the duration) to train a machine learning model.

As described in the present disclosure, there are various situations or use cases in which the star player **102** performing a task or action is of interest is determined. This can include, for example, when a highlight video or video montage is to be created for interesting events in a game session, highlighting or augmenting events in a live broadcast of the gameplay session, or when gameplay data for a session is to be used for player analysis or coaching, among other such use cases. Furthermore, in at least some embodiments, it is desirable to determine when certain types of events occur within the game session in order to generate certain types of dynamic assets corresponding to the type of events. For example, a slow-motion effect is added to a live broadcast of the game session in response to a particular type of event (e.g., a team victory or goal being scored).

In at least one embodiment, media representative of a game session can be analyzed. This includes, for example, an audio and video stream of the game session, or audio and/or video data captured and stored for the game session, among other such options. The media, in various embodiments, is analyzed by the hard analyzer to detect or identify the presence of specific occurrences of events in the game. For example, this can include any event in a game, such as events related to the appearance or disappearance of an object or character, the death or revival of a character, use of an item, an activation of a switch, a collection of an item, an achievement, a victory condition, completion of an intermediate task, completion of an objective, and the like. In some

embodiments, media data can be analyzed to attempt to determine these occurrences by detecting those specific actions in the audio, video, text, or other such game content.

In various embodiments, the hard analyzer is used to detect the occurrence of events (e.g., events predicted by the soft analyzer) from gameplay data (e.g., audio and/or video data capture from a game session) in accordance with various embodiments. In various examples, the hard analyzer, which can also be implemented using, as, or as part of any of a device, system, service, or process, accepts one or more types of gameplay data as input. The input can include, for example, live gameplay received in a media stream, recorded media stored to an accessible storage medium, or media rendered in real time for presentation on a player device, among other such options. In at least some embodiments, additional game data is obtained as well, including text, metadata, player input (e.g., audio, keystroke, or button press), player biometrics, or other such information that is useable to detect the occurrence of events and/or determining detectors to use. In some embodiments, this data also includes information about the game being played and/or player whose gameplay data is being analyzed.

In one example, the hard analyzer receives video frames (each of which may comprise one or more image frames) of a stream for the game session (e.g., a sampling of frames, such as one frame per 100 ms or every tenth frame). In some embodiments, the hard analyzer receives all frames, but only analyzes a sampling and/or portion thereof. The frames (or other content) to be analyzed or a portion thereof, in various embodiments, are pre-processed using one or more pre-processing algorithms. For example, the one or more pre-processing algorithms include format conversion (e.g., converting from a first video format to a second video format), color correction, grayscaling, color isolating, converting to hue, saturation, value (HSV) color space, upscaling, downscaling, smoothing, noise removal, filtering, stretching, warping, or perspective correction, among other such options, or other algorithms used to process frames or other data so that it can be used by the hard analyzer to detect the occurrence of the event. In various embodiments, a repository or other data store is used to store a set of pre-processing algorithms, and a pre-processing module can select the appropriate algorithm(s) for the content. In some embodiments, the algorithms to be applied are determined based at least in part upon a type of content to be analyzed, or a result of a prior pre-processing step. In one example, game-specific configuration information is used to indicate the types of pre-processing to be performed for a certain game. Various other determination approaches can be used as well within the scope of the various embodiments.

In at least one embodiment, dependent region processing is performed for one or more frames. In such embodiments, during performance of dependent region processing, detection of one object or occurrence of an event triggers additional processing to be performed for one or more other regions of a particular frame. In one example, an icon is detected to appear in a first region of a video frame and the appearance of this icon is indicative of the presence of additional information elsewhere in the frame. In this example, one or more corresponding regions of the frame are then analyzed using one or more detectors associated with that type of additional information. In at least one embodiment, detection of such an object or occurrence triggers a sequence or series of detectors to obtain additional information about a state of the game, whether represented in audio, video, user input, or other such data. In various embodiments, one or more of these additional detectors

were not enabled when the icon is detected but are instead activated or triggered upon detection. In some embodiments, combinations of events are analyzed to determine a particular outcome. For example, an icon might appear on a screen indicating a particular event occurred, but this might be accompanied by another action or display indicating information about the party or player that caused that event or was affected by that event.

In one example, individual frames can have a sequence of 10 pre-processing algorithms applied. This can include, for example, first identifying from the game configuration file one or more regions of the frame to analyze. In this example, the regions are rectangles defined by coordinates or percentages. Various other image or content manipulation techniques can be used in accordance with various embodiments. As mentioned, in at least some games the content (e.g., text) of interest will be displayed against a background of the game. In various embodiments, in order to improve the accuracy of certain detection algorithms (e.g., optical 15 character recognition (OCR) algorithms), thresholding can be used to remove or apply a specific value to background pixels, such that the region once processed appears more like black and white content, particularly for text, which can improve performance of certain OCR algorithms. In this 20 example, the data for the pre-processed regions can then be temporarily stored using a cache or other such location.

In various embodiments, the hard analyzer then accesses the region data from a cache and processes the data using one or more detectors. Furthermore, in some embodiments, 25 the game-specific configuration information specifies the detector(s) to be used, which can also vary by selection or type of region to be analyzed. The detectors can include any of a variety of detector types, as may relate to pattern detection, icon detection, text detection, audio detection, 30 image detection, motion detection, and the like. The hard analyzer, for example, accesses the relevant detectors from a detector repository or other such location, if not already stored in local memory. In various embodiments, a region corresponding to a heads-up display (HUD) can have at least 35 text and icon detection performed as discussed in greater detail below. In various embodiments where additional game data is available, detection can also include user input analysis, such as to detect inputs, or combinations of inputs, to a keyboard, joypad, controller, etc. If the additional data 40 includes sound or webcam video, the detector can also look for patterns in the audio, such as where the star player 102 makes a particular explanation indicative of a type of event, or patterns in the video, where the star player 102 makes a particular action or motion indicative of the occurrence of 45 the event and/or a type of event. Other types of data can be analyzed as well, such as biometric data for the star player 50 102 that may indicate actions or responses indicative of the occurrence of the event and/or types of events. In at least one embodiment, the analysis can be done in near real-time using data streams or after a gameplay session using stored data, among other such options. The types of data available 55 may then depend at least in part upon when the data is analyzed.

The hard analyzer, in one embodiment, processes the 60 frames (or other game content) using the specified detector(s), which generate one or more cues or other such outputs, which can be stored to a local cache or other storage location. The cues, for example, include cues indicative of, or mapped to, the occurrence of the event and/or a type of 65 event a type of event. As an example, a game might indicate a number of skull icons that indicate a number of kills the star player 102 has caused during a current gameplay

session. Therefore, in such an example, a change in the number of skulls indicates or otherwise confirms the occurrence of a kill event. A visual cue in that example includes the skull itself, such as the third skull appearing at a position it was previously absent from.

In at least some embodiments, the one or more events, or types of events, indicated by the determined cue(s) are determined based at least in part on a cue-to-event translation module. In various embodiments, the cue-to-event translation module includes executable code, provided through the game-specific script, that as a result of being executed by the game server, causes the game server to determine the occurrence of the event and/or a type of event from the determined cues. Various detectors, in at least one embodiment, provide different types of outputs in different formats, and the cue-to-event translation module can provide at least some level of standardization so that output can be compared across various detectors. This can be particularly important where multiple detectors may detect cues for the same events, which then need to be correlated as appropriate. These cues, for example, include cues relating to detected text, icons, motions, features, images, sounds, gestures, biometrics, etc. The cue-to-event translation module, in at least some embodiments, includes one or more trained neural networks, chained or otherwise, that can accept the cues for a specific time or interval of gameplay and infer a type of event that occurred with a corresponding confidence value.

In such examples, the translated event data can then be written to an event log or other such location for access. In various embodiments, the log can also store the data in a format that is usable by one or more processes, algorithms, or applications to perform one or more tasks as discussed herein, such as determining dynamic assets to generate, display dynamic assets, discard dynamic assets, or other operations performed in response to the occurrence of the event and/or types of events. In some embodiments, the log includes data for all detected events. In yet other embodiments the log stores data for certain types or numbers of events, or events determined with at least a minimum confidence, among other such options. In at least some embodiments, parameters of the detection, such as a search area, desired cue, and mapping of the changes in a state of the cue to event logs, can be configurable via data objects (e.g., JSON—JavaScript Object Notation).

In at least one embodiment, output of a set of detectors (such as five or six detectors for a given game) will be a match or non-match for an occurrence of an event and/or type of event, with a corresponding confidence value or level of confidence. These cues or other values can then be fed to a process, such as a game-specific script or other executable code (e.g., JavaScript) in the translation module, that can perform additional heuristics per-frame in accordance with an embodiment. These heuristics can help to improve the event match determination, for example, an OCR algorithm is used to determine a match for detecting specific textual content, but heuristics may be applied to see how and when that textual content changed, and by how much, and over what period of time, to determine whether the event actually corresponds to a predicted event obtained from the soft analyzer.

FIG. 2 illustrates an example of a streaming gameplay session 210 in which events within the gameplay session are predicted and/or detected in accordance with an embodiment. Furthermore, in at least the embodiments described in connection with FIG. 2, a renderer 214 modifies the streaming gameplay session 210 based at least in part on the

predicted and/or detected event. The modified streaming gameplay session generated by the renderer 214 can be broadcast as described above and/or stored for use later for a training video, highlight video, montage video, a replay, rebroadcast, or other uses. As referenced above, in various embodiments, the streaming gameplay 210 includes audio and/or video data captured from a gameplay session. Furthermore, a game engine or other executable code (as a result of being executed by one or more processors of a computer system) renders or otherwise generates the gameplay (e.g., audio and/or video of a user playing the game).

In addition, a game engine or other executable code, in various embodiments, streams or otherwise transmits over a network connection the audio and/or video generated as a result of the user playing the game. For example, a client device 204 (e.g., client device 504 described below in connection with FIG. 5) executes a game application, captures gameplay data (e.g., audio, video, user input, biometrics, etc.) and transmits the captured gameplay data to a game server 226. In yet other embodiments, the game server 226 executes the game application and streams gameplay data to the client device 204, the client device 204 then provides user input back to the game server 226 (e.g., games-as-a-service configurations such as NVIDIA™ GeForce NOW™). In various embodiments, the client device 204 also provides input data 238 to the game server 226. As described above, the input data 238 includes any data captured by an input device or other device used to control and/or play the game application. For example, the input data 238 includes mouse and keyboard data, controller data, motion tracking data, biometric data, video data, audio data, or any other data that can be used to play a game.

The game server 226 includes one or more computer systems such as the game servers described below in connection with FIGS. 5 and 8 in accordance with at least one embodiment. Furthermore, the game server 226 includes a stream receiver 212, an input data receiver 216, an event detector 224, and a renderer 214, all of which include executable code that, as a result of being executed by the game server 226, cause the game server 226 to perform various operations as described in the present disclosure. In one example, the stream receiver 212, the input data receiver 216, the event detector 224, and the renderer 214 include applications and/or are integrated into a single application that, as a result of being executed by the game server 226, perform various operations described in the present disclosure. In an embodiment, the stream receiver 212 obtains captured gameplay data and processes the gameplay data for use by the renderer 214. For example, processing the gameplay data by the stream receiver 212 include encoding, decoding, grayscaling, color isolating, converting, upscaling, downscaling, smoothing, noise removal, filtering, stretching, warping, perspective correction, or otherwise modifying the gameplay data for use by the renderer 214.

In various embodiments, the input data receiver 216 obtains input data 238 from the client device 204 and processes the input data 238 for use by the event detector 224. For example, the input data receiver 216 extracts a portion of the obtained input data 238, converts the input data 238, or otherwise enables the event detector 224 to use the obtained input data 238. In at least one embodiment, the event detector 224 includes one or more algorithms that predicts and/or detects the occurrence of events within the gameplay session. Furthermore, in various embodiments, the event detector 224 predicts and/or detects the occurrence of events within the gameplay session based at least in part on the input data (e.g., after processing by the input data

## 11

receiver 216). In one example, the event detector 224 includes a soft analyzer as described in greater detail below in connection with FIG. 3, which predicts the occurrence of an event within a gameplay session using information determined based at least in part on the input data 238. In accordance with the embodiments described in the present disclosure, the event detector 224 includes one or more algorithms suitable for predicting the occurrence of the event and/or determining the occurrence of the event (e.g., predicting an event is likely to occur and detecting the occurrence of the event) such as those described in U.S. Provisional Patent Application No. 62/868,654, filed on Jun. 28, 2019, entitled “UNSUPERVISED CATEGORIZATION OF GAMEPLAY VIDEO USING MACHINE LEARNING MODELS,” U.S. patent application Ser. No. 16/586,506, filed on Sep. 27, 2019, entitled “DETERMINING HIGH-INTEREST DURATIONS OF GAMEPLAY SESSIONS FROM USER INPUTS,” and U.S. patent application Ser. No. 16/669,939, filed on Mar. 30, 2021, and entitled “GAME EVENT RECOGNITION” which are hereby incorporated by reference as if set forth in its entirety herein.

In various embodiments, once the event detector 224 has predicted the event and/or confirmed the occurrence of the event, the event detector 224 indicates to the renderer 214 information associated with the event. For example, the event detector 224 indicates to the renderer 214 a duration of the event, a type of the event, confirmation that the event occurred, an indication that the event is predicted, or other information suitable for the renderer 214 to modify audio and/or video of the gameplay session. As described above, the renderer 214, in an embodiment, broadcasts the gameplay session, as such, the renderer 214, in response to information obtained from the event detector 224, modifies the broadcasts. As described in greater detail below in connection with FIG. 3, the renderer 214 generates assets to modify the broadcast based at least in part on information obtained from the event detector 224. In one example, the event detector 224 indicates a predicted event to the renderer 214, the renderer 214 then causes the broadcast of the gameplay to be modified (e.g., display special effects or overlay). In addition, in such examples, the event detector 224 later indicates that the predicted event occurred, the renderer 214 then causes the broadcast of the gameplay to be modified again (e.g., causes the displayed special effects or overlay to be removed). In other embodiments where the gameplay is not broadcast (e.g., saved for later use), the same operation can be performed.

FIG. 3 illustrates an environment 300 in which a game server 326 augments a broadcast 314 with dynamically generated assets in response to anticipated in-game events in accordance with an embodiment. In an embodiment, a client device 304 streams a game play session with the game server 326 where the game server 326 executes the game application which is streamed to the client device 304 and the client device 304 provides inputs to the game application such as described in connection with FIG. 8. In yet other embodiments, the client device 304 executes the game application and gameplay (e.g., audio and video) which is captured and transmitted to the game server 326. In various embodiments, the client device 304 includes various computing devices described in the present disclosure such as client device 804 described in connection with FIG. 8. Similarly, in various embodiments, the game server 326 includes various server computer systems described in the present disclosure such as game server 802 described in connection with FIG. 8.

In various embodiments, the game server 326 includes various components as illustrated in FIG. 3 such as a game

## 12

capture 312 component, an analyzer 306, including a hard analyzer 308 and a soft analyzer 310, a dynamic asset generator 332, a broadcast stream 314, a static image overlay 316, various asset(s) 318, including effect(s) 320, dynamic image overlay(s) 322, and dynamic audio effect(s) 324. In an embodiment, the game capture 312 component captures renderings of the gameplay session such as audio and/or display data (e.g., as image data capturing the rendered frame of the gameplay session). For example, the game capture 312 component obtains image data rendered by the game application executed by the game service 326. In another example, the game capture 312 component obtains image data from a video stream or other data obtained from the client device 304 executing the game application. Furthermore, the broadcast stream, in an embodiment, includes the audio and video data obtained by the game capture 312 component. In one example, the game capture 312 component obtains audio and video data from the game application executed by the game server 326 and converts the audio and video data into a broadcast stream 314 for transmission to a remote audience as described above.

In addition, in at least one embodiment, the broadcast stream 314 includes the static image overlay 316. In one example, the static image overlay 316 includes a score or other statistic obtained from the gameplay session and presented in the broadcast stream 314. In another example, the static image overlay 316 includes names and images of the players playing the game. In yet other embodiments, the static image overlay 316 is omitted or displayed intermittently (e.g., in response to a goal being scored in the game play session).

In an embodiment, the analyzer 306 includes hardware and/or software (e.g., executable code that, as a result of being executed by the game server 326, causes the game server 326 to perform various operations described in the present disclosure) that predicts an occurrence of an event within an interval of time and confirms whether the predicted event occurred within the interval of time. As illustrated in FIG. 3, the analyzer 306 includes, in an embodiment, the hard analyzer 308 and the soft analyzer 310. In such embodiments, the soft analyzer 310 includes one or more algorithms (e.g., represented as executable code) that predict the occurrence of the event within the interval of time. In turn, in such embodiments, the hard analyzer 308 includes one or more algorithms (e.g., represented as executable code) that detect the occurrence of the event within the interval of time. In one example, the soft analyzer 310 obtains input data from the client device 304 and predicts the occurrence of the event within the interval of time and the hard analyzer 308 obtains image data from the game application (e.g., captured by the game capture 312 component) and determines whether the event has occurred within the interval or time.

In various embodiments, during training of the soft analyzer 310, to determine and/or predict the occurrence of the event in the game within the interval of time, the soft analyzer 310 analyzes user input generated by user inputs to one or more input devices during the gameplay session. To do so, in an embodiment, the soft analyzer 310 computes a user activity score for a duration of relatively high user activity based on a set of the user input events that occur in the duration. In one example, the user activity score is based at least in part on various factors, such as a number of the user input events that occur in the duration (density), one or more action commands that are entered during the duration, a length of the duration, and/or an actuation speed (e.g.,

urgency) of the one or more input devices during the duration. In an embodiment, the soft analyzer 310 uses the user activity score to determine and/or predict whether an in-game event is likely to occur within an interval of time. For example, the soft analyzer 310 computes a statistical value on user activity scores of a plurality of durations of the gameplay session (and/or other gameplay sessions) and an action may be performed based at least in part on comparing the user activity score of the duration to the statistical value (e.g., determining the user activity score is greater than the statistical value or other threshold value). In some embodiments, the statistical value corresponds to a Median Absolute Deviation (MAD) score that is computed for the user activity scores.

In various embodiments, the hard analyzer 308 includes a set of trained neural networks or other algorithms for detecting the occurrence of the event based at least in part on audio and/or video captured from the gameplay session. In one example, the hard analyzer 308 includes various neural networks such as a Convolutional Neural Network (CNN), Radial Basis Function Neural Network, Recurrent Neural Network, Long Short-Term Memory (LSTM), or other neural networks trained to detect an object within image data obtained from the game play session. In yet other embodiments, other detectors are used by the hard analyzer 308 to detect the occurrence of an event. In one example, a detector detects the occurrence of an event based at least in part on audio cues obtained from the gameplay session (e.g., “nice shot,” “goal,” “team victory,” etc.). In yet other embodiments, a detector detects a character displayed within the gameplay session using optical character recognition (OCR) to determine the occurrence of the event.

In at least one embodiment, at least one neural network will be trained per game. In at least one embodiment, a set of neural networks will be trained per game, with different networks being trained to recognize different types of features, such as scenes, actions, or objects. In at least one embodiment, a network can be trained that can be used for inferencing across a variety of games, or at least across games of a specific type or category with at least somewhat similar gameplay. In at least one embodiment, a first model might be trained to recognize features of a type of game like a first-person shooter, while another model might be trained to recognize features of a type of game like a platformer or third person adventure game, as there would be different types of features to detect. For example, the types of features to detect can vary by game or type of game. In various embodiments, training data for the set of neural networks can include video streams including annotations for features of types to be recognized for that game or type of game. In such embodiments, these annotations are performed manually or with modeling assistance.

In various embodiments, the analyzer 306 indicates to the dynamic asset generator 332 that the predicted event will occur within the interval of time and/or that the event occurred. As described in greater detail below in connection with FIG. 4, in an embodiment, the soft analyzer 310 indicates that a particular event and/or type of event is likely (e.g., predicted with a confidence score) to occur. In addition, in some embodiments, the soft analyzer 310 predicts a duration of time the event will occur for and/or interval of time the event will occur within. For example, the soft analyzer 310 predicts a team counterattack will occur within three seconds and last for ten seconds. Once the particular event and/or type of event is predicted, the analyzer 306 and/or soft analyzer 310, transmits an indication of the particular event and/or type of event to the dynamic asset

generator 332 in accordance with at least one embodiment. Similarly, the analyzer 306 and/or soft analyzer 310, in an embodiment, indicates to the hard analyzer 308 the particular event and/or type of event.

5 In this manner, the hard analyzer 308, in various embodiments, begins to perform one or more checks of the captured game data to determine if the particular event and/or type of event has occurred. Returning to the example above, in response to the soft analyzer 310 predicting the team counterattack, a notification is transmitted to the hard analyzer 308 and the dynamic asset generator 332. In this example, the hard analyzer 308 determines if and/or when the event occurs and, if the event is detected, transmits an indication to the dynamic asset generator 332 or causes the indication 10 to be transmitted to the dynamic asset generator 332. Furthermore, in this example, the dynamic asset generator 332 in response to the indication of the particular event and/or type of event generates one or more asset(s) 318 to be included in the broadcast stream 314 to generate the modified broadcast stream 330. Finally, once the hard analyzer 308 indicates the event has occurred and/or the interval of time predicted by the soft analyzer 310 has elapsed, the dynamic asset generator 332, in an embodiment, causes the asset(s) 318 to be removed or otherwise no longer displayed 15 in the modified broadcast stream 330.

In various embodiments, the asset(s) 318 include effect(s) 320, dynamic image overlay(s) 322, and/or dynamic audio effect(s) 324 that can be applied to the modified broadcast stream 330. In general, the asset(s) 318 include any video, 20 audio, or modification of a component of the broadcast stream 314 that can be used to generate the modified broadcast stream 330 in accordance with at least one embodiment. In one example, the effect(s) 320 include slow-motion, color modifications, video and/or audio 25 enchantments or modification, or other special effects that can be applied to a component of the broadcast stream 314. The dynamic image overlay(s) 322, in an embodiment, include various overlays that can be applied to the broadcast stream 314 such as score updates, player names, tokens, avatars, 30 graphics, images, text, or any other overlay. Similarly, dynamic audio effect(s) 324, in an embodiment, include various effects applied to an audio portion of the broadcast stream 314 such as noise cancellation, equalization, compression, volume modifications, filters, or other effects.

45 FIG. 4 illustrates an environment 400 in which a soft analyzer 410 predicts an in-game event and a hard analyzer 408 determines if the event occurred 410 in accordance with an embodiment. In an embodiment, the soft analyzer 410 generates an event anticipation proposal 402. In one example, the event anticipation proposal 402 includes an indication of a type of event predicted, an anticipation start (T=0) 404, and an anticipation end (T=M) 406. In other examples, the event anticipation proposal 402 indicates that an event will occur. In various embodiments, the anticipation start 404 time indicates an interval of time at which an animation and/or effect 418 is generated for a broadcast stream and when the hard analyzer 408 begins to check or otherwise determine if the event occurred 410. For example, at time T=0 (e.g., the anticipation start 404) the soft analyzer 410 predicts an event will occur within 3 second, in response, a dynamic asset generator generates and/or begins to generate the animation and/or effect 418 and the hard analyzer prepare to determine (e.g., loads one or more neural networks, obtains captured game data, etc.) and/or determines if the event occurred 410.

50 As described above, the animation and/or effect 418 includes various modifications to a broadcast stream to

generate a modified broadcast stream such as overlays, audio effects, video effects, or other modifications. Furthermore, in an embodiment, if the hard analyzer 408 detects the event (e.g., detect a particular object within capture game data), the hard analyzer 408 causes the animation and/or effect 418 to be displayed 416. In one example, the hard analyzer 408 determines that a goal is scored (e.g., the predicted event has occurred) and causes the dynamic asset generator to display a goal animation (e.g., the animation and/or effect 418). In another example, the hard analyzer 408 determines that the event has not occurred and/or will not occur prior to the anticipation end 406, and causes the dynamic asset generator to discard 412 the animation and/or effect 418. In yet other embodiments, if the hard analyzer 408 does not detect the event has occurred 410 prior to time T=M (e.g., the anticipation end) the animation and/or effect 418 is discarded 412.

In various embodiments, different actions can be performed at various intervals of time as described in FIG. 4. In one example, the hard analyzer 408 detects if the event occurred 410 at time T=M when the anticipation end 406 occurs (e.g., the interval at which the soft analyzer 410 predicts the event will occur). In other embodiments, the hard analyzer 408 continues to determine if the event occurred 410 after the anticipation end 406 for an interval of time. For example, the soft analyzer 410 predicts an event will occur within 10 second with a confidence level of fifty percent, based at least in part on the confidence level, the hard analyzer 408 continues to determine if the event occurred 410 beyond time T=M (e.g., anticipation end 406). In yet other embodiments, the type of event predicted causes a modification to operations described in FIG. 4. For example, the soft analyzer 410 predicts a player death in a first-person shooter, in response, capture video of the player (e.g., the animation and/or effect 418) is overlaid in the display 416 prior to the anticipation end 406.

With reference to FIG. 5, FIG. 5 illustrates an environment 500 for predicting occurrences or events within gameplay sessions from user inputs, in accordance with some embodiments of the present disclosure. It should be understood that this and other arrangements described herein are set forth only as examples. Other arrangements and elements (e.g., machines, interfaces, functions, orders, groupings of functions, etc.) may be used in addition to or instead of those shown, and some elements may be omitted altogether. For example, video-clip server(s) 516 can be omitted in embodiments where the gameplay sessions are broadcast live. Further, many of the elements described herein are functional entities that may be implemented as discrete or distributed components or in conjunction with other components, and in any suitable combination and location. Various functions described herein as being performed by entities may be carried out by hardware, firmware, software, or any combination thereof. For instance, various functions may be carried out by a processor executing instructions stored in memory.

In various embodiments, the environment 500 of FIG. 5 includes, one or more components instantiated using among other things, client devices 504(A), 504(B), and 504(N) (referred to collectively herein as “client devices 504”), a video-clip server(s) 516, and/or a game server(s) 526. Although the client devices 504(A), 504(B), and 504(N) are illustrated in FIG. 5, this is not intended to be limiting. In any example, there may be any number of the client devices 504. The client devices 504 (and the components and/or

features thereof) may be implemented using one or more computing devices, such as the computing devices described in more detail below.

In at least one embodiment, the components of environment 500 communicate over a network(s) 502. The network(s) 502, for example, include a wide area network (WAN) (e.g., the Internet, a public switched telephone network (PSTN), etc.), a local area network (LAN) (e.g., Wi-Fi, ZigBee, Z-Wave, Bluetooth, Bluetooth Low Energy (BLE), Ethernet, etc.), a low-power wide-area network (LPWAN) (e.g., LoRaWAN, Sigfox, etc.), a global navigation satellite system (GNSS) network (e.g., the Global Positioning System (GPS)), and/or another network type. In an example, the components of the environment 500 communicate with one or more of the other components via one or more of the network(s) 502.

In an embodiment, the client devices 504 include a smart phone, a laptop computer, a tablet computer, a desktop computer, a wearable device, a game console, a virtual reality system (e.g., a headset, a computer, a game console, remote(s), controller(s), and/or other components), a streaming device, (e.g., an NVIDIA SHIELD™), a smart-home device that may include an intelligent personal assistant, and/or another type of device capable of supporting game-play.

Furthermore, the client devices 504, in one example, include a game application 506, a display 508, a communication interface 510, an input device(s) 512, a local storage 536, a game data capturer 538, an actions manager 530, a duration determiner 540, and an interest determiner 542. Although only a few components and/or features of the client device 504 are illustrated in FIG. 5, this is not intended to be limiting. For example, the client devices 504 can include additional and/or alternative components.

In an embodiment, the game application 506 includes a mobile application, a computer application, a console application, and/or another type of application. The game application 506, for example, includes instructions that, as a result of being executed by a processor(s), cause the processor(s) to, without limitation, receive input data representative of user inputs to the one or more input device(s) 512, transmit the input data to the game server(s) 526, retrieve game data from memory or local storage, receive the game data using the communication interface 510 from the game server(s) 526, and cause display of the game on the display 508. In other words, the game application 506, in various embodiments, operates as a facilitator for enabling playing of a game associated with the game application 506 on the client devices 504. In yet other examples, the game application 506 also includes instructions that, as a result of being executed by a processor(s), cause the processor(s) to transmit data to, and receive data from, the video-clip server(s) 516. For example, the game application 506 transmits to the video-clip server(s) 516 video recordings generated when games are played on the client devices 504, and received from the video-clip servers(s) 516 video clips that are associated with events predicted by interest determiner 542.

The game application 506 and/or patches or updates to the game application 506 are downloaded from the game server(s) 526 or downloaded from another server(s), such as a server of a content delivery network (CDN) in accordance with at least one embodiment. For example, the game server(s) 526 is located in a different country or on a different continent, so to reduce the download time, the game application 506 and/or the patches or updates are stored on different servers around the globe. As such, when the client devices 504 are downloading the game application

**506** and/or the patches or updates, the client devices **504** can connect to a more local server that is part of the CDN, for example.

In some examples, the client devices **504** renders the game using the game application **506**, while in other examples, the client devices **504** receive display data (e.g., encoded display data, as described with respect to FIG. 8) and use the display data to display the game on the display **508**. In some examples, a first client device, such as client device **504(A)**, renders the game while a second client device, such as client device **504(B)**, receives the display data and displays the game using the display data. In examples where the display data is received by the client device (e.g., where the client device **504** does not generate the rendering), the environment **500** may be part of a game streaming system, such as the game streaming system **800** of FIG. 8, described in more detail below.

In an embodiment, the client device **504(A)** displays, via the display **508**, a plurality of gameplay sessions over time, such as a gameplay session **514**. The gameplay sessions, such as the gameplay session **514**, for example, includes any number of gameplay sessions participated in by users of the client device **504(A)**. Similarly, users of each of the client devices **504**, in addition to the client device **504(A)**, can participate in any number of gameplay sessions in accordance with at least one embodiment.

The display **508**, in various embodiments, includes various types of displays capable of displaying a game (e.g., a light-emitting diode display (LED), an organic LED display (OLED), a liquid crystal display (LCD), an active matrix OLED display (AMOLED), a quantum dot display (QDD), a plasma display, an LED/LCD display, and/or another type of display). In some examples, the display **508** includes more than one display (e.g., a dual-monitor display for computer gaming, a first display for configuring a game and a virtual reality display for playing the game, etc.). In some examples, the display is a touch-screen display, such as a touch-screen of a smart phone, tablet computer, laptop computer, or the like, where the touch-screen is at least one of the input device(s) **552** of the client device **504**.

In an embodiment, the input device(s) **552** includes various types of devices that are capable of providing inputs associated with a game, such as user inputs to the game and/or used to control the game. For example, the input device(s) include a keyboard, a mouse, a microphone(s), a touch-screen display, a controller(s), a remote(s), a headset (e.g., sensors of a virtual reality headset), and/or other types of input devices.

The communication interface **510**, in various embodiments, includes one or more components and features for communicating across one or more networks, such as the network(s) **502**. In one example, the communication interface **510** communicates via any number of network(s) **502**, described herein. For example, to communicate in the environment **500** of FIG. 5, the client devices **504** uses an Ethernet or Wi-Fi connection through a router to access the Internet in order to communicate with the video-clip server(s) **516**, the game server(s) **526**, and/or with others of the client devices **504**.

In various embodiments, the local storage includes any of a variety of computer-readable media. The computer-readable media, for example, includes any available media that can be accessed by the client device **504(A)**. The computer-readable media includes both volatile and nonvolatile media, and removable and non-removable media in accordance with at least one embodiment. By way of example, and not

limitation, the computer-readable media comprises computer-storage media and communication media.

Additional aspects will now be described, including operations that are carried out in the course of rendering the gameplay session **554** and/or analyzing game-session data (e.g., user input data and/or video data associated with a gameplay session) generated as a result of playing the game during the gameplay session **554** in accordance with an embodiment. For example, the gameplay session **554** is associated with any number of input events **558(A)-558(N)** (referred collectively herein as “input events **558**” or “user input events **558**”), a timeline **548**, time segments **550(A)-550(N)** (referred collectively herein as “time segments **550**”), and durations **552(A)-552(N)** (referred collectively herein as “durations **552**”).

In accordance with embodiments of the disclosure, the duration determiner **540** analyzes the game-session data to identify one or more of the durations **552(A)-552(N)** of relatively high user activity during the gameplay session **514** that indicate the occurrence of an event. For example, a duration of high input activity relative to other durations of the gameplay session indicated that an event will occur within an interval of time. Additionally or alternatively, in an embodiment, the interest determiner **542** analyzes the durations **552(A)-552(N)** to determine and/or predict whether an identified duration is of sufficient interest for further action based on corresponding game-session data. For example, the interest determiner **542** predicts a likelihood that an event will occur and determines based at least in part on the likelihood whether to perform additional actions (e.g., cause a dynamic asset to be generated).

In various embodiments, the game application **506** and/or the game data capturer **538** include instructions that, as a result of being executed, record or otherwise capture at least some of the game-session data (e.g., corresponding to the input events **518** and/or video frames of gameplay) from gameplay sessions **514** and store the recorded game-session data locally on the client device **504** (e.g., in local storage **536**) or transmit the recorded game-session data to the video-clip server **556** or the game server **526** to be stored in the data stores(s) **524** and/or **534**, respectively. For example, the input events **558** and/or video frames of gameplay are used by the soft analyzer and/or hard analyzer to perform various operations described in the present disclosure such as predicting an event and determining whether the event occurred. Game-session data, for example, includes user input data associated with a gameplay session and/or video data associated with a gameplay session. In examples where the client device **504** does not generate the rendering of a game (such as the game streaming system **800** of FIG. 8 described in more herein), the game server(s) **526** records and store the video data or transmits the video data to the video-clip server(s) **556** for storage in the data store(s) **524**.

The game data capturer **538**, in an embodiment, is part of the game application **506** or part of a separate application (e.g., one or more system services, programs, etc.). In one example, the game data capturer **538** is a component of the input device(s) **552** or is executed by some other component of the client devices **504**. Furthermore, in an embodiment, the game data capturer **538** includes instructions that, as a result of being executed by a processor(s), cause the processor(s) to (by example and without limitation) record or log game data, such as input-device usage data, video data, audio data, and/or other data associated with a gameplay session. Examples of input-device usage data includes user input data descriptive or representative of keyboard, mouse, or other input-device usage, such as user input events, and

that is associated with one or more gameplay sessions, such as the gameplay session 514. Examples of information that can be recorded include user input events, which may correspond to keyboard strokes, mouse clicks, mouse movement, microphone inputs, video-camera inputs, and/or inputs to the client devices 504 during the gameplay sessions. Examples of a user input event include a predefined input or combination of inputs that registers as a user input event (e.g., by an operating system of the client device 504(A)), such as a key press (e.g., a key down or a key up), a gesture, a mouse movement, a game command, or a predefined combination and/or pattern of particular key and/or other inputs (which may collectively register as a single input event). For example, a user input event may be identified for each key press provided to an input device.

Examples of the input events for the gameplay session 554 include the input events 558(A)-558(N), which are identified from corresponding input event data that is representative of an input event (also referred to as a “user input event”). For example, in response to an input event, the game data capturer 538 stores metadata descriptive of or otherwise associated with the input event. As an example, the game data capturer 538 stores timestamp information (e.g., a timestamp) that correlates with timing information of the game-session data (e.g., along the timeline 548). Another example of the metadata is a gameplay session identification information (e.g., a gameplay session identifier) that identifies the gameplay session in which the input event was received. A further example of the metadata is an application or game identification information (e.g., an application or game identifier) that identifies the game or application in which the input event was received. A further example is input identification information (e.g., one or more input identifiers) that identifies what input(s) was actuated to trigger the user input event (e.g., a key(s) pressed).

In an embodiment, the timeline 548 is associated with the gameplay session 554 and indicates relative time designations at which the time segments 550 can be sequentially positioned. For illustrative purposes, the time segments 550 are depicted on the timeline 548. In one example, the time segment 550 span a specific duration of time in the timeline 548, and define one or more corresponding durations 552 (A)-552(N). Furthermore, in such examples, the time segment 550 is associated with one or more of the input events 558(A)-558(N). In an embodiment, any number of the input events 558(A)-558(N) includes a timestamp that falls within a corresponding time segment.

The duration determiner 540, in various embodiments, analyzes user input data of gameplay sessions, such as input event data, to identify durations of relatively high user activity during the gameplay sessions. For example, the duration determiner 540 analyzes the input events 558 associated with the gameplay session 554 (and/or other game-session data) to determine the time segments 550 that define the corresponding durations 552. This analysis, in various examples, is performed on real-time game-session data during a gameplay session, or on non-real-time game-session data (e.g., in the local storage 536), such as after completion of the gameplay session. In at least one embodiment, the duration determiner 540 is part of the game application 506 or a separate application (e.g., one or more system services, programs, etc.). In some examples, the duration determiner 540 is part of the same application as the game data capturer 538. Further, like the game data capturer 538, in some embodiments, the duration determiner 540 is at

least partially on the video-clip server(s) 556 and/or the game server(s) 526 in addition to or instead of the client device 504(A).

To identify durations of relatively high user activity, the duration determiner 540 determines durations of relatively higher levels (e.g., frequencies or concentrations) of user activity during a gameplay session based on the input events in accordance with an embodiment. For example, the duration determiner 540 uses the input events 558 of the gameplay session 554 to determine the time segments 550 during the gameplay session 554 that include higher levels of action relative to other time segments in the gameplay session 554, and these time segments 550 define the durations 552 of relatively high user activity. For example, the time segment 550(A) within the gameplay session 554 is identified by the duration determiner 540 as the duration 552(A) of relatively high user activity based at least in part on the time segment 550(A) having a relatively high value(s) of a user activity measurement for the time segment 550(A), which may correspond to high Keys Per Second (KPSs), a high percentage of action-key selections per second, and/or other frequency based input-device metrics. In at least one embodiment, to determine durations of relatively high user activity, the duration determiner 540 computes the user activity measurement and compares the user activity measurement to a threshold value. In one example, the duration determiner 540 computes the user activity measurement for a gameplay session (e.g., the gameplay session 554) from the input events (e.g., the input events 558) of the gameplay session. To do so, in such examples, the duration determiner 540 analyzes the input events using metadata associated with the input events, such as the timestamp information. For example, the timestamp information is used to compute the frequency at which the input events are generated, the speed at which inputs are actuated, and whether and/or how many action commands that are predefined for a game of the gameplay session were actuated.

By way of example, an action command refers to a predefined input actuation (e.g., an input event) such as a key or combination of input actuations (e.g., an input event(s)) such as keys for a game that when present results in an objective being met in the game (e.g., left mouse button click initiates weapon deployment or changes weapons, right mouse button click initiates shield deployment, left trigger initiates vehicle drifting, keys a and b together with a right trigger may initiates building a structure, etc.). Action commands, for example, can be predefined for a particular game and/or games. Thus, one game or application can have a different set of action commands than another game or application. The input identification information is also be used in embodiments where the frequency is based on predefined input events or types of input events being generated (e.g., predefined inputs and/or action commands).

In various embodiments, the user activity measurement is a running user activity measurement having values computed periodically (e.g., every second) or iteratively and/or may be computed as needed. The duration determiner 540, for example, defines a time segment of a gameplay session based at least in part on a time(s) that the user activity measurement determines a value(s) of the user activity measurement exceeds the threshold value (e.g., the user activity measurement threshold). In an embodiment, a start time of a duration is based at least in part on a time at which the duration determiner 540 determines the user activity measurement has exceeded the threshold value such that it is greater than the threshold value. Additionally or alternatively, a time of a duration, in at least one embodiment, is

based at least in part on a time at which the duration determiner 540 determines the user activity measurement has exceeded the threshold value such that it is less than the threshold value.

In some examples, the time at which the user activity measurement exceeds the threshold value such that it is greater than the threshold value is used as the start time of the time segment 550(A). In an embodiment, additionally or alternatively, the time at which the user activity measurement exceeds the threshold value such that it is less than the threshold value is used as the end time of the time segment 550(A). In examples, a start time and/or an end time of a duration can be different than a time at which the duration determiner 540 determines the user activity measurement has exceeded the threshold value. For example, the duration determiner 540 uses the time to define the start time and derive the end time from the start time (e.g., by adding a minute or other amount of time to the start time). Similarly, the duration determiner 540 uses the time to define the end time and derive the start time from the end time (e.g., by subtracting a minute or other amount of time from the end time) in accordance with an embodiment. As further examples, the duration determiner 540 uses the time to define the start time and/or the end time as a duration of time that occurs before and/or after the identified time. For example, where the time corresponds to when the user activity measurement exceeds the threshold value, an amount of time may be added before and/or after that time to define the time segment.

In at least one embodiment, the threshold value is hard-coded or may be computed. For example, the duration determiner 540 computes a statistic from values of the user activity measurement over the gameplay session 554 and/or many gameplay sessions of the user and/or other users. For example, the statistic corresponds to an average or percentile of the user activity measurement over at least a portion of a gameplay session(s). Furthermore, in such examples, the statistics are used as the threshold value or otherwise define the threshold value. The threshold value, in an embodiment, is user-specific and based at least in part on user interactions with gameplay session and/or gameplay session specifically based at least in part on the gameplay session being analyzed. In further embodiments, the determined statistics are global and based at least in part on user interactions of other users in gameplay sessions of the game. In examples where the duration determiner 540 determines the threshold value during the gameplay session (e.g., in real-time), the statistics are based part least in part on at least a portion of the gameplay session leading up to a current time (e.g., the time associated with the user activity measurement). In yet other examples where the duration determiner 540 determines the threshold value after the gameplay session, the statistics are based at least in part on at least any portion of the gameplay session before and/or after the time associated with the user activity measurement being compared to the threshold value. In any example, the statistics (and threshold value) are updated based at least in part on the time being compared to the threshold value or the same statistic (or threshold value) is used for various comparisons.

In at least one embodiment, when predicting events, the duration determiner 540 distinguishes between input events that correspond to gameplay activity and input events that correspond to non-gameplay activity. Input events that correspond to gameplay activity, for example, include input events that control or effectuate gameplay within a gameplay session. An example of an input event that corresponds to gameplay activity is an action command. Further examples

include an input actuation or combination of input actuations that effectuate directional movement (e.g., steering, exploring, etc.), weapons deployment (e.g., firing a weapon, reloading a weapon, switching weapons, etc.), speed adjustments (e.g., running, walking, etc.), body positioning (e.g., standing, crouching, etc.), and the like.

In contrast, examples of input events that correspond to non-gameplay activity include input events that do not control or effectuate gameplay within a gameplay session. In 10 one example, input events that correspond to non-gameplay activity include input events that control or effectuate activity that accompanies or facilitates gameplay activity, such as communication between gamers (e.g., teammates, etc.) during a gameplay session, setting game options such as resolution, volume level, etc., navigating a menu to initiate gameplay, etc. In another example, input events that correspond to non-gameplay activity include those that initiate a communication mode between users and/or players (e.g., enables voice recordings, keyboard chat, etc.), define content of a communication (e.g., typing of a message), terminate the communication mode, and/or transmit the communication to at least one other user and/or player.

In various embodiments, different approaches are used by the duration determiner 540 to identify input events that correspond to non-gameplay activity. In some examples, one or more input events are predefined as non-gameplay activity commands (e.g., in the corresponding metadata) to indicate that corresponding non-gameplay activity. Non-gameplay activity commands, for example, are game-specific and/or predefined for a particular game or games. In 25 some examples, one or more non-gameplay activity commands are predefined as initiation commands that initiate input events that correspond to non-gameplay activity and the duration determiner 540 considers subsequent input events (e.g., of a particular type such as message defining or 30 menu navigation input events) as non-gameplay activity (and/or perform further analysis to determine the input events correspond to non-gameplay activity based on identifying the command).

Also in some examples, one or more non-gameplay activity commands are predefined as termination commands that terminate input events that correspond to non-gameplay activity and the duration determiner 540 considers previous input events (e.g., of a particular type such as message defining or menu navigation input events) as non-gameplay activity (and/or perform further analysis to determine the input events correspond to non-gameplay activity based at 45 least in part on identifying the command). For example, the duration determiner 540 determines input events between an initiation command and a termination command corresponding to non-gameplay activity. As indicated, one or more input events that correspond to gameplay activity can still be present between the commands (e.g., a user may use a mouse to control gameplay while typing a message on a 50 keyboard), but the commands may identify particular types of input events as corresponding to non-gameplay activity in accordance with at least one embodiment.

In at least one embodiment, the duration determiner 540 assigns a value (e.g., weight) to input events that correspond 55 to non-gameplay activity differently than (e.g., less than) input events that correspond to gameplay activity in computing the user activity measurement. As another example, the duration determiner 540 discards input events that correspond to non-gameplay activity when computing the user activity measurement. In various embodiments, these approaches make it less likely that the duration determiner 540 identifies durations in which user activity is largely

attributable to non-gameplay activity. In some embodiments, once the duration determiner 540 identifies a duration of relatively high user activity, the duration determiner 540 compares the duration to a minimum and/or maximum length duration threshold to determine whether to discard the duration or maintain the duration for additional analysis by, for example, the interest determiner 542. For example, the duration determiner 540 discards an identified duration of relatively high user activity if the duration falls below the minimum length duration (e.g., 50 seconds or less). Similarly, in another example, the duration determiner 540 discards or truncates the identified duration of relatively high user activity if the duration exceeds the maximum length duration (e.g., greater than one minute).

In embodiments, the minimum and maximum length durations may be hardcoded, configurable, dynamically and/or automatically determined by the duration determiner 540. Also, in some examples, the duration determiner 540 merges a duration with at least one other duration based at least in part on determining one or more of those durations fall below the minimum length duration. Another factor the duration determiner 540 uses to merge durations is the proximity of the durations in the gameplay session in at least one embodiment. For example, durations are merged into a single duration based at least in part on the duration determiner 540 determining the durations are within a threshold proximity of one another. The duration determiner 540, in various embodiments, adds time to a duration (merged or otherwise) to reach at least the minimum length duration and/or truncate a duration (merged or otherwise) to stay under at least the maximum length duration.

In an embodiment, the interest determiner 542 determines and/or predicts whether an identified duration (e.g., identified by the duration determiner 540) within a gameplay session is indicative of and/or predictive of an event based at least in part on user input events generated by user inputs to one or more input devices during a gameplay session. As with the duration determiner 540, the interest determiner 542, in an embodiment, is part of the game application 506 or part of a separate application (e.g., one or more system services, programs, etc.). In some examples, interest determiner 542 is part of the same application as the game data capturer 538. Furthermore, in yet other examples, the interest determiner 542 is at least partially instantiated or otherwise executed on the video-clip server(s) 516 and/or the game server(s) 526 in addition to or instead of being instantiated or otherwise executed on the client device 504(A).

In various embodiments, to determine whether a duration indicates and/or predicts an event, the interest determiner 542 computes a user interest score. For example, the user interest score for a duration is based at least in part on a set of the user input events of a gameplay session that occur during the identified duration. For example, for the duration 552(A), the user interest score is computed from the input events 518 that have a timestamp within the duration 552(A) and/or the time segment 550(A) (e.g., between the start time and end time of the duration). In an embodiment, the interest determiner 542 computes the user interest score based on various factors, such as a number of the user input events that occur during the duration, one or more action commands that are entered during the duration, a length of the duration, a number or presence of input events that correspond to non-gameplay activity during the duration, and/or an actuation speed of the one or more input devices during the duration.

In at least one embodiment, the interest determiner 542 assigns a value (e.g., weight) within a duration input event that corresponds to non-gameplay activity differently than (e.g., less than) input events that correspond to gameplay activity in computing the user interest score for the duration. As another example, the interest determiner 542 discards input events that correspond to non-gameplay activity when computing the user interest score or otherwise account for the presence and/or amount of those input events.

10 In an embodiment, the duration determiner 540 identifies a duration to have relatively high user activity. For example, once a duration is identified, the interest determiner 542 determines whether the identified duration is predictive of an event based at least in part on calculating the user interest score for the identified duration by applying an interest-level algorithm to user input data associated with the identified duration. An example of such an algorithm is a High Energy Area Technique ("HEAT") algorithm, which can be expressed with the following equation (1):

$$\text{HEAT} = \log(k * (A_{\text{factor}} * D)^2 * K_c * S_{\text{max}}) \quad (1)$$

where,

$A_{\text{factor}}$ =High constant (e.g., greater than 5), if an action command is present in the duration and is otherwise=5,  
 25  $D$ =A length of the duration,  
 $K_c$ =number of keys and/or input events in the duration,  
 $S_{\text{max}}$ =Max speed of actuation among keys and/or input events in the duration, and  
 $K$ =a constant.

30 Any of the above examples of factors for an interest-level algorithm may be used without the others and/or in a different formula or equation in accordance with at least one embodiment. Furthermore, in some examples, rather than being set to a high constant when an action command is present, the  $A_{\text{factor}}$  is based at least in part on (e.g., increase based at least in part on) the number of action commands in the duration. A user interest score, for example, is computed for identified duration within the gameplay session. In various embodiments, based at least in part on the user interest scores (e.g., values of HEAT) for identified durations, the interest determiner 542 determines and/or predicts which of those durations, if any, correlate with an event within the game. For example, using the user interest scores, the interest determiner 542 compares a threshold value to user interest scores to determine whether a duration(s) is of sufficiently high interest (e.g., is greater than the threshold value) to predict an event within the gameplay session with occur within an interval of time. In some embodiments, the threshold value is a statistical value computed from the user interest scores of the identified durations of the gameplay session. In further examples, the statistical value additionally or instead, is computed from user interest scores of durations from one or more other gameplay session of the user, other users, and/or the same game.

55 As an example, the statistical value corresponds to a Median Absolute Deviation (MAD) score that is computed from the user interest scores. In any example, the statistical value corresponds to a minimum user interest score required for an identified duration to be predictive of an event. In an embodiment, the interest determiner 542 compares the user interest score of a duration to the determined statistical value to determine whether the user interest score is predictive of an event occurring within an interval of time. For example, if the user interest score for the identified duration exceeds the statistical value, the identified duration is deemed as a duration predictive of the event occurring within the interval of time. Similarly, in another example, if the user interest

score for an identified duration falls below the statistical value, the identified duration is discarded as not predictive of the event. MAD is one example of a statistical value which can be computed, but other statistics can be used, such as a standard deviation, an average, etc. in accordance with at least one embodiment. In various examples, the statistical value corresponds to a robust statistic, such as a robust measure of statistical dispersion of the user interest scores (e.g., MAD).

In various embodiments, in addition to or instead of using the statistical value to predict or determine whether a duration is predictive of the event, the interest determiner 542 utilizes other factors, such as whether the duration includes at least one action command and/or whether the user interest score is greater than a baseline threshold value. For example, the interest determiner 542 determines the duration is of sufficiently high predicted interest when its user interest score exceeds the statistical value. However, in other examples, when the user interest score does not exceed the statistical value, the interest determiner 542 can still determine the duration is predictive of the event based at least in part on the other factors.

In an embodiment, the action(s) manager 530 enables the various actions that may be performed during or after a gameplay session based at least in part on the interest determiner 542 determining and/or predicting the duration is predictive of the event. For example, the action manager 530 enables or cause presentation of an indication of the event on a display, such as the display 508, as further described with respect to FIG. 3. In some embodiments, the indications include transmitting information indicating the predicted event, a type of event, the duration of the event, an interval of time at within which the event will occur, and other information useable by the game server(s) 526 to modify a broadcast stream as described above in connection with FIG. 3. These are just some non-limiting examples of the indications which may be presented for predicted event(s).

The video-clip server(s) 516, in an embodiment, include one or more servers for storing, trimming, sharing, or otherwise editing video-clips associated with events recorded during gameplay session 514. Although only a few components and/or features of the video-clip server(s) 516 are illustrated in FIG. 5, this is not intended to be limiting. For example, the video-clip server(s) 556 may include additional or alternative components, such as those described below with respect to the computing device described in the present disclosure.

The video-clip server(s) 516, in various embodiments, is separate or distinct from the game server(s) 526; however, this is not intended to be limiting. Further, the video-clip server(s) 516, in some embodiments, is not included in the environment 500, but may be used to facilitate one or more of the actions performed by the action manager 530. In some examples, the video-clip server(s) 516 is implemented using the same or similar servers to the game server(s) 526 (e.g., running as a task on the game server(s) 526). In some examples, the video-clip server(s) 516 is operated or hosted by a first entity (e.g., a first company) and the game server(s) 526 is operated or hosted by a second entity (e.g., a second, different company). For example, the second entity is a game developer or publisher, and the first entity and the second entity share data such that the first entity can identify events that correspond to the identified durations using data received from the second entity and/or the client device 504(A). In other examples, the video-clip server(s) 516 and the game server(s) 526 are operated or hosted by the same entity. In further examples, the environment 500 is imple-

mented completely on a client device 504 and/or one or more of the components and/or functionality thereof shown as being included in a server may be at least partially implemented on the client device 504.

5 The video-clip server(s) 516, for example, include a communications interface 522 and a data store(s) 524. In an embodiment, one or more video clips are generated based on the predicted events within the gameplay sessions, which are identified and/or selected using the duration determiner 540 and/or the interest determiner 542. For example, the video-clip server(s) 516 receives the video clips based at least in part on the predicted events from the game server(s) 526 and/or the client device(s) 504. In an embodiment, the communication interface 522 includes one or more components and features for communicating across one or more networks, such as the network(s) 502.

10 The game server(s) 526, in an embodiment, includes one or more servers (e.g., dedicated game servers) for storing, hosting, managing, and, in some examples, rendering a game. 20 In some examples, first game server(s) 526 is used to create, update, and modify a game (e.g., the program code of the game), and second game server(s) 526 is used to host the game (e.g., as dedicated game servers). Although only a few components and/or features of the game server(s) 526 are illustrated in FIG. 5, this is not intended to be limiting. For example, the game server(s) 526 may include additional or alternative components, such as those described below with respect to the computing devices described in the present disclosure.

25 In an embodiment, the game server(s) 526 includes one or more Application Programming Interfaces (APIs) to enable gameplay by the client device(s) 504 and/or to enable communication of information (e.g., user input data, etc.) with the video-clip server(s) 516. For example, the game server(s) 526 includes one or more game APIs that interface with the game applications 506 of the client devices 504 to enable gameplay by the client devices 504. As another example, the game server(s) 526 includes one or more gameplay session APIs that interface with the duration determiner 540 to pass data representative of durations that indicate and/or predict events to the interest determiner 542 for further analysis. As a further example, the game server(s) 526 includes one or more APIs that interface with the duration determiner 540 to receive the identified durations to determine and/or predict the event within the gameplay session. Although different APIs are described herein, the APIs may be part of a single API, two or more of the APIs may be combined, different APIs may be included other than those described as examples herein, or a combination thereof.

30 In an embodiment, the game server(s) 526 includes the game engine 528. The game engine 528, for example, includes the functionality of the game that enables the game to be played by one or more users over a network, such as network(s) 502. For example, the gameplay session 554 is played using the game engine 528. In an embodiment, the game engine 528 includes a rendering engine, an audio engine, a physics engine, an animation engine, an artificial intelligence engine, a networking engine, a streaming engine, a memory management engine, and/or other components or features. In an example, the game engine 528 is used to generate some or all of the user input data during a gameplay session.

35 The communication interface 532 of the game server(s) 526, in various embodiments, includes one or more components and features for communicating across one or more networks, such as the network(s) 502. For example, the

communication interface 532 communicates via any number of network(s) 502, described herein. For example, to communicate in the environment 500 of FIG. 5, the game server(s) 526 communicates over a LAN with other game server(s) and/or over the Internet with other game server(s) 526, video-clip server(s) 556, and/or the client devices 504.

With reference to FIG. 6, FIG. 6 is an example of a modified broadcast stream 600 of a gameplay session. As described above, in various embodiments, a predicted event triggers special effects 602 to be applied to data captured from a gameplay session where the predicted event is determined based at least in part on the user interest score for the duration exceeding the threshold value as described above in FIG. 5. The user interface is shown as being presented during broadcast of a gameplay session, but could be presented at any suitable time and may or may not be part of the game and/or an application used to present the game. The special effects 602 shown includes various elements, overlays, and effects as described in the present disclosure that correspond to a particular predicted event.

As an example, special effects 602 is shown applying a slow-motion effect to the broadcast of the gameplay session in response to the predicted event indicating that “player 1” will kill “player 2” within an interval of time. For example, as a user (e.g., gamer) plays a game, user input data is captured by the game data capturer and used to identify durations of relatively high user activity. In such examples, of the durations of high user activity, the interest determiner determines at least one duration that is predictive of the event occurring within the interval of time. As a result, a dynamic asset generator receives an indication of the event and generates the special effects 602 in accordance with an embodiment.

As described above, a hard analyzer determines the predicted event occurs and causes the dynamic asset generator to display the special effects 602. For example, if a user is attempting to complete a puzzle-based mission, the soft analyzer determines and/or predicts the user will complete the puzzle within an interval of time based at least in part on a corresponding duration of user activity and then transmits an indication to the dynamic asset generator. In this example, the dynamic asset generator then generates the special effects 602 (e.g., an image overlay with the text “congratulations” and fireworks), and once the event occurs and is confirmed by the hard analyzer, the dynamic asset generator causes the special effects 602 to be displayed.

The special effects 602 and the modified broadcast 600 of FIG. 6 are merely examples presented during or after a gameplay session. As described herein, and by way of non-limiting example, other operations performed include a highlight reel or game summarization (e.g., video clips corresponding to the durations) presented during or after a gameplay session, as well as any other operations to save, edit, and/or share a video clip that corresponds to predicted events (e.g., in a video editor application). In some embodiments, a game server generates data representative of predicted events within the gameplay session (e.g., video frames extracted from video data of the gameplay session and/or timestamps of the video frames that correspond to the event) that correspond to the durations and that are identified from video data of the gameplay session. In at least one embodiment, the data includes identifiers (e.g., metadata) of locations of the video clips within the gameplay session. Additionally or alternatively, in at least one embodiment the data includes video data, such as a concatenated sequence of the video clips, which may or may not include intervening video clips.

FIG. 7 shows a process 700 for displaying asset(s) within a gameplay broadcast or other gameplay video accordance with at least one embodiment. The process 700, in an embodiment, is performed by a game engine, game server, client device, video clip server, or other service and/or servers as described above. Now referring to FIG. 7, the blocks of processes 700, described in the present disclosure, in various embodiments, include a computing process that is performed using any combination of hardware, firmware, and/or software. The various functions, processes, and/or operations described in the present disclosure may be carried out by a processor executing instructions stored in memory. The processes, for example, are embodied as computer usable instructions stored on computer storage media. In addition, in some embodiments, the processes are provided by a standalone application, a service or hosted service (standalone or in combination with another hosted service), or a plug-in to another product, to name a few. In addition, the processes 700 are described, by way of example, with respect to the computer systems of FIGS. 3 and 5. However, these processes may additionally or alternatively be executed by any one system, or any combination of systems, including, but not limited to, those described in the present disclosure. In addition, various blocks of the processes 700 may be performed in different and/or alternative orders, performed in serial or parallel, or omitted entirely.

Returning to FIG. 7, the system executing the process 700, at block B702 obtains gameplay stream and event data. As described above, in various embodiments, a client device executes a game application which renders the game and transmits the gameplay stream to the game server. In addition, in such embodiments, the client device captures event data such as user input data and other data generated at least in part on the operation of the user to effectuate an operation of the game application or otherwise control the game. In yet other embodiments, the game server executes the game application, captures gameplay from the game application, and obtains at least a portion of the event data from the client device (e.g., user input data).

In an embodiment, in block B704, the system executing the process 700, predicts an in-game event based at least in part on the event data. As described above in connection with FIG. 5, in an embodiment, events are predicted based at least in part on durations of high user activity relative to other durations of gameplay. For example, a soft analyzer calculates user interest level using equation (1) as described above, if the user interest level is within a value of a threshold the soft analyzer predicts an event will occur within an interval of time. In other examples, additional algorithms and/or alternative algorithms are used to predict the event will occur within the interval of time.

In an embodiment, in block B706, the system executing the process 700, generates asset(s) for the gameplay stream based at least in part on the predicted in-game event. For example, as described above, a dynamic asset generator generates the asset(s) in order to modify the gameplay stream. In an embodiment, the asset(s) generated are determined based at least in part on the type of event predicted. Furthermore, in various embodiments, predicting the event allows additional processing and/or computational time to generate the asset(s) such that there is little to no delay in displaying the asset(s) in response to detecting the event occurred. Furthermore, in various embodiments, in block B708, the system executing the process 700, determines if the event occurred. In one example, a hard analyzer detects the event occurs based at least in part on gameplay data. In an embodiment, the hard analyzer utilizes one or more

trained neural networks to detect an object within the gameplay data (e.g., images and/or video captured from the game).

In an embodiment, if in block B708, the event is detected (e.g., the event occurred), the system executing the process 700, continues to block B710 and displays the asset(s) within the game broadcast. For example, as described above in connection with FIG. 3, the game server applies the asset(s) to the game broadcast to generate a modified game broadcast. However, if in block B708, the system executing the process 700 determines the event did not occur and/or the event is not detected within an interval of time (e.g., the interval of time the soft analyzer predicted the event will occur within), the process 700 continues to block B712 and the asset(s) are discarded. For example, as a result of the hard analyzer not detecting the event before the expiration of the interval of time, the game server deletes the asset(s) generated by the dynamic asset generator.

Now referring to FIG. 8, FIG. 8 illustrates an environment 800 in which a game streaming system streams a game to a client device 804, in accordance with some embodiments of the present disclosure. In an embodiment, the game server(s) 802 includes similar components, features, and/or functionality to the game server(s) 526 of FIG. 5, the client device(s) 804 includes similar components, features, and/or functionality to the client devices 504 of FIG. 5, and the network(s) 806 include similar components, features, and/or functionality to the network(s) 502 of FIG. 5.

In at least one embodiment, the game server(s) 802 hosts a cloud computing platform used by the client device(s) 804. For example, the gameplay sessions of a game(s) presented using the game application of the client device(s) 804 are facilitated by the cloud computing platform. In various embodiments, the cloud computing platform generates the rendering of the gameplay session (e.g., the video data thereof) that is presented on the display 824. The cloud computing platform, for example, dynamically provisions and provides wholly and/or partially virtualized computing environments. As an example, the cloud computing platform executes games hosted on the cloud computing platform on one or more virtual machines.

In an embodiment, for a gameplay session, the client device(s) 804 receives input data in response to inputs to the input device(s), transmits the input data to the game server(s) 802, receives encoded display data from the game server(s) 802, and displays the display data on the display 824. As such, the more computationally intense computing and processing is offloaded to the game server(s) 802 (e.g., rendering of the gameplay session is executed by the GPU(s) of the game server(s) 802) in accordance with at least one embodiment. In other words, the gameplay session is streamed to the client device(s) 804 from the game server(s) 802, thereby reducing the requirements of the client device(s) 804 for graphics processing and rendering.

For example, with respect to an instantiation of a gameplay session, a client device 804 is displaying a frame of the gameplay session on the display 824 based at least in part on receiving the display data from the game server(s) 802. The client device 804, in various embodiments, receives an input to one of the input device(s) and generates input data in response. In such embodiments, the client device 804 then transmits the input data to the game server(s) 802 via the communication interface 820 and over the network(s) 806 (e.g., the Internet), and the game server(s) 802 receives the input data via the communication interface 818. In an embodiment, the CPU(s) receives the input data, processes the input data, and transmits data to the GPU(s) that causes

the GPU(s) to generate a rendering of the gameplay session. For example, the input data is representative of a movement of a character of the user in a game, firing a weapon, reloading, passing a ball, turning a vehicle, etc. The rendering component 812, in an embodiment, renders the gameplay session (e.g., representative of the result of the input data) and the render capture component 814 captures the rendering of the gameplay session as display data (e.g., as image data capturing the rendered frame of the gameplay session). In addition, in various embodiments, the encoder 816 then encodes the display data to generate encoded display data and the encoded display data is transmitted to the client device 804 over the network(s) 806 via the communication interface 818. In such embodiments, the client device 804 receives the encoded display data via the communication interface 820 and the decoder 822 decodes the encoded display data to generate the display data. For example, the client device 804 then displays the display data via the display 824 after it has been decoded.

Further, at least a portion of the game data capturer 538 of FIG. 5 may reside on a client device 804 and is used to provide the input data and/or input event data to the game server(s) 802 in accordance with at least one embodiment. In one example, the CPU(s) 808 receives the input data, processes the input data, and/or uses the soft analyzer and/or the hard analyzer to predict the occurrence of the event and/or confirm the event occurred within the gameplay session. In at least one embodiment, the client device 804 processes the input data, and/or uses the soft analyzer (which can include the duration determiner and/or the interest determiner) to identify and/or predict events, event durations, event types, and/or intervals of time before events occur.

#### Inference and Training Logic

FIG. 9A illustrates inference and/or training logic 915 used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided below in conjunction with FIGS. 9A and/or 9B.

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

In at least one embodiment, any portion of code and/or data storage 901 may be internal or external to one or more processors or other hardware logic devices or circuits. In at

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

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

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

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

In at least one embodiment, inference and/or training logic 915 may include, without limitation, one or more arithmetic logic unit(s) (“ALU(s)”) 910, including integer and/or floating point units, to perform logical and/or math-

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

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

In at least one embodiment, activation storage 920 may be cache memory, DRAM, SRAM, non-volatile memory (e.g., flash memory), or other storage. In at least one embodiment, activation storage 920 may be completely or partially within or external to one or more processors or other logical circuits. In at least one embodiment, a choice of whether activation storage 920 is internal or external to a processor, for example, or comprising DRAM, SRAM, flash memory or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

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

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

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

#### Neural Network Training and Deployment

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

least one embodiment, training may be performed in either a supervised, partially supervised, or unsupervised manner. In at least one embodiment, untrained neural network 1006 is trained using supervised learning, wherein training dataset 1002 includes an input paired with a desired output for an input, or where training dataset 1002 includes input having a known output and an output of neural network 1006 is manually graded. In at least one embodiment, untrained neural network 1006 is trained in a supervised manner and processes inputs from training dataset 1002 and compares resulting outputs against a set of expected or desired outputs. In at least one embodiment, errors are then propagated back through untrained neural network 1006. In at least one embodiment, training framework 1004 adjusts weights that control untrained neural network 1006. In at least one embodiment, training framework 1004 includes tools to monitor how well untrained neural network 1006 is converging towards a model, such as trained neural network 1008, suitable to generating correct answers, such as in result 1014, based on input data such as a new dataset 1012. In at least one embodiment, training framework 1004 trains untrained neural network 1006 repeatedly while adjusting weights to refine an output of untrained neural network 1006 using a loss function and adjustment algorithm, such as stochastic gradient descent. In at least one embodiment, training framework 1004 trains untrained neural network 1006 until untrained neural network 1006 achieves a desired accuracy. In at least one embodiment, trained neural network 1008 can then be deployed to implement any number of machine learning operations.

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

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

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

In at least one embodiment, OpenVINO is a toolkit for facilitating development of applications, specifically neural network applications, for various tasks and operations, such as human vision emulation, speech recognition, natural language processing, recommendation systems, and/or

variations thereof. In at least one embodiment, OpenVINO supports neural networks such as convolutional neural networks (CNNs), recurrent and/or attention-based neural networks, and/or various other neural network models. In at least one embodiment, OpenVINO supports various software libraries such as OpenCV, OpenCL, and/or variations thereof.

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

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

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

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

devices (e.g., a first set of layers on a first device, such as a GPU, and a second set of layers on a second device, such as a CPU).

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

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

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

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

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

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

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

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

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

In at least one embodiment, any of configuration manager 1124, resource manager 1126, and resource orchestrator 1112 may implement any number and type of self-modifying actions based on any amount and type of data acquired in any technically feasible fashion. In at least one embodiment, self-modifying actions may relieve a data center operator of data center 1100 from making possibly bad configuration decisions and possibly avoiding underutilized and/or poor performing portions of a data center.

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

described above with respect to data center 1100 by using weight parameters calculated through one or more training techniques described herein.

In at least one embodiment, data center may use CPUs, application-specific integrated circuits (ASICs), GPUs, FPGAs, or other hardware to perform training and/or inferencing using above-described resources. Moreover, one or more software and/or hardware resources described above may be configured as a service to allow users to train or 10 performing inferencing of information, such as image recognition, speech recognition, or other artificial intelligence services.

Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or 15 more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 11 for inferencing or predicting operations based, at least in part, 20 on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations 25 associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

#### 30 Autonomous Vehicle

FIG. 12A illustrates an example of an autonomous vehicle 1200, according to at least one embodiment. In at least one embodiment, autonomous vehicle 1200 (alternatively referred to herein as “vehicle 1200”) may be, without limitation, a passenger vehicle, such as a car, a truck, a bus, and/or another type of vehicle that accommodates one or more passengers. In at least one embodiment, vehicle 1200 may be a semi-tractor-trailer truck used for hauling cargo. In at least one embodiment, vehicle 1200 may be an airplane, robotic vehicle, or other kind of vehicle.

Autonomous vehicles may be described in terms of automation levels, defined by National Highway Traffic Safety Administration (“NHTSA”), a division of US Department of Transportation, and Society of Automotive Engineers (“SAE”) “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles” (e.g., Standard No. J3016-201806, published on Jun. 15, 2018, Standard No. J3016-201609, published on Sep. 30, 2016, and previous and future versions of this standard). In 45 at least one embodiment, vehicle 1200 may be capable of functionality in accordance with one or more of Level 1 through Level 5 of autonomous driving levels. For example, in at least one embodiment, vehicle 1200 may be capable of conditional automation (Level 3), high automation (Level 50 4), and/or full automation (Level 5), depending on embodiment.

In at least one embodiment, vehicle 1200 may include, without limitation, components such as a chassis, a vehicle body, wheels (e.g., 2, 4, 6, 8, 18, etc.), tires, axles, and other 55 components of a vehicle. In at least one embodiment, vehicle 1200 may include, without limitation, a propulsion system 1250, such as an internal combustion engine, hybrid electric power plant, an all-electric engine, and/or another propulsion system type. In at least one embodiment, propulsion system 1250 may be connected to a drive train of vehicle 1200, which may include, without limitation, a transmission, to enable propulsion of vehicle 1200. In at

least one embodiment, propulsion system 1250 may be controlled in response to receiving signals from a throttle/accelerator(s) 1252.

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

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

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

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

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

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

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

41

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.

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

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

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

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

In at least one embodiment, any number of stereo camera(s) 1268 may also be included in a front-facing configuration. In at least one embodiment, one or more of stereo camera(s) 1268 may include an integrated control unit comprising a scalable processing unit, which may provide a programmable logic (“FPGA”) and a multi-core micro-

42

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

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

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

Inference and/or training logic 915 are used to perform 45 inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 12B for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

In various embodiments, inference and/or training logic 55 are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 12C is a block diagram illustrating an example system architecture for autonomous vehicle 1200 of FIG. 12A, according to at least one embodiment. In at least one embodiment, each of components, features, and systems of vehicle 1200 in FIG. 12C is illustrated as being connected via a bus 1202. In at least one embodiment, bus 1202 may include, without limitation, a CAN data interface (alternatively

tively referred to herein as a “CAN bus”). In at least one embodiment, a CAN may be a network inside vehicle **1200** used to aid in control of various features and functionality of vehicle **1200**, such as actuation of brakes, acceleration, braking, steering, windshield wipers, etc. In at least one embodiment, bus **1202** may be configured to have dozens or even hundreds of nodes, each with its own unique identifier (e.g., a CAN ID). In at least one embodiment, bus **1202** may be read to find steering wheel angle, ground speed, engine revolutions per minute (“RPMs”), button positions, and/or other vehicle status indicators. In at least one embodiment, bus **1202** may be a CAN bus that is ASIL B compliant.

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

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

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

In at least one embodiment, CPU(s) **1206** may include a CPU cluster or CPU complex (alternatively referred to herein as a “CCPLEX”). In at least one embodiment, CPU(s) **1206** may include multiple cores and/or level two (“L2”) caches. For instance, in at least one embodiment, CPU(s) **1206** may include eight cores in a coherent multi-processor configuration. In at least one embodiment, CPU(s) **1206** may include four dual-core clusters where each cluster has a dedicated L2 cache (e.g., a 2 megabyte (MB) L2 cache). In at least one embodiment, CPU(s) **1206** (e.g., CCPLEX) may

be configured to support simultaneous cluster operations enabling any combination of clusters of CPU(s) **1206** to be active at any given time.

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

In at least one embodiment, GPU(s) **1208** may include an integrated GPU (alternatively referred to herein as an “iGPU”). In at least one embodiment, GPU(s) **1208** may be programmable and may be efficient for parallel workloads. In at least one embodiment, GPU(s) **1208** may use an enhanced tensor instruction set. In at least one embodiment, GPU(s) **1208** may include one or more streaming microprocessors, where each streaming microprocessor may include a level one (“L1”) cache (e.g., an L1 cache with at least 96 KB storage capacity), and two or more streaming microprocessors may share an L2 cache (e.g., an L2 cache with a 512 KB storage capacity). In at least one embodiment, GPU(s) **1208** may include at least eight streaming microprocessors. In at least one embodiment, GPU(s) **1208** may use compute application programming interface(s) (API(s)). In at least one embodiment, GPU(s) **1208** may use one or more parallel computing platforms and/or programming models (e.g., NVIDIA’s CUDA model).

In at least one embodiment, one or more of GPU(s) **1208** may be power-optimized for best performance in automotive and embedded use cases. For example, in at least one embodiment, GPU(s) **1208** could be fabricated on Fin field-effect transistor (“FinFET”) circuitry. In at least one embodiment, each streaming microprocessor may incorporate a number of mixed-precision processing cores partitioned into multiple blocks. For example, and without limitation, 64 PF32 cores and 32 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.

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

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

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

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

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

In at least one embodiment, accelerator(s) **1214** (e.g., hardware acceleration cluster) may include one or more

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

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

In at least one embodiment, accelerator(s) **1214** may include programmable vision accelerator (“PVA”), which may alternatively be referred to herein as a computer vision accelerator. In at least one embodiment, PVA may be designed and configured to accelerate computer vision algorithms for advanced driver assistance system (“ADAS”) **1238**, autonomous driving, augmented reality (“AR”) applications, and/or virtual reality (“VR”) applications. In at least one embodiment, PVA may provide a balance between performance and flexibility. For example, in at least one embodiment, each PVA may include, for example and without limitation, any number of reduced instruction set computer (“RISC”) cores, direct memory access (“DMA”), and/or any number of vector processors.

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

In at least one embodiment, DMA may enable components of PVA to access system memory independently of CPU(s) **1206**. In at least one embodiment, DMA may support any number of features used to provide optimization

to a PVA including, but not limited to, supporting multi-dimensional addressing and/or circular addressing. In at least one embodiment, DMA may support up to six or more dimensions of addressing, which may include, without limitation, block width, block height, block depth, horizontal block stepping, vertical block stepping, and/or depth stepping.

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

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

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

In at least one embodiment, a computer vision network on-chip may include an interface that determines, before transmission of any control signal/address/data, that both a PVA and a DLA provide ready and valid signals. In at least one embodiment, an interface may provide for separate

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

In at least one embodiment, one or more of SoC(s) 1204 may include a real-time ray-tracing hardware accelerator. In at least one embodiment, real-time ray-tracing hardware accelerator may be used to quickly and efficiently determine positions and extents of objects (e.g., within a world model), to generate real-time visualization simulations, for RADAR signal interpretation, for sound propagation synthesis and/or analysis, for simulation of SONAR systems, for general wave propagation simulation, for comparison to LIDAR data for purposes of localization and/or other functions, and/or for other uses.

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

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

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

In at least one embodiment, a DLA may be used to run any type of network to enhance control and driving safety, including for example and without limitation, a neural network that outputs a measure of confidence for each object detection. In at least one embodiment, confidence may be represented or interpreted as a probability, or as providing a relative “weight” of each detection compared to other detections. In at least one embodiment, a confidence measure enables a system to make further decisions regarding which detections should be considered as true positive detections rather than false positive detections. In at least one embodiment, a system may set a threshold value for confidence and consider only detections exceeding threshold value as true positive detections. In an embodiment in which an automatic emergency braking (“AEB”) system is used, false positive detections would cause vehicle to automatically perform emergency braking, which is obviously undesirable. In at

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

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

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

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

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

In at least one embodiment, processor(s) 1210 may further include a safety cluster engine that includes, without limitation, a dedicated processor subsystem to handle safety management for automotive applications. In at least one embodiment, a safety cluster engine may include, without limitation, two or more processor cores, a tightly coupled

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

In at least one embodiment, processor(s) 1210 may include a video image compositor that may be a processing block (e.g., implemented on a microprocessor) that implements video post-processing functions needed by a video playback application to produce a final image for a player window. In at least one embodiment, a video image compositor may perform lens distortion correction on wide-view camera(s) 1270, surround camera(s) 1274, and/or on in-cabin monitoring camera sensor(s). In at least one embodiment, in-cabin monitoring camera sensor(s) are preferably monitored by a neural network running on another instance of SoC 1204, configured to identify in cabin events and respond accordingly. In at least one embodiment, an in-cabin system may perform, without limitation, lip reading to activate cellular service and place a phone call, dictate emails, change a vehicle's destination, activate or change a vehicle's infotainment system and settings, or provide voice-activated web surfing. In at least one embodiment, certain functions are available to a driver when a vehicle is operating in an autonomous mode and are disabled otherwise.

In at least one embodiment, a video image compositor may include enhanced temporal noise reduction for both spatial and temporal noise reduction. For example, in at least one embodiment, where motion occurs in a video, noise reduction weights spatial information appropriately, decreasing weights of information provided by adjacent frames. In at least one embodiment, where an image or portion of an image does not include motion, temporal noise reduction performed by video image compositor may use information from a previous image to reduce noise in a current image.

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

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

In at least one embodiment, one or more Soc of SoC(s) 1204 may further include a broad range of peripheral

51

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

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

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

Embodiments described herein allow for multiple neural networks to be performed simultaneously and/or sequentially, and for results to be combined together to enable Level 3-5 autonomous driving functionality. For example, in at least one embodiment, a CNN executing on a DLA or a discrete GPU (e.g., GPU(s) 1220) may include text and word recognition, allowing reading and understanding of traffic signs, including signs for which a neural network has not been specifically trained. In at least one embodiment, a DLA may further include a neural network that is able to identify, interpret, and provide semantic understanding of a sign, and to pass that semantic understanding to path planning modules running on a CPU Complex.

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

52

least one embodiment, a flashing light may be identified by operating a third deployed neural network over multiple frames, informing a vehicle’s path-planning software of a presence (or an absence) of flashing lights. In at least one embodiment, all three neural networks may run simultaneously, such as within a DLA and/or on GPU(s) 1208.

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

In at least one embodiment, a CNN for emergency vehicle detection and identification may use data from microphones 1296 to detect and identify emergency vehicle sirens. In at least one embodiment, SoC(s) 1204 use a CNN for classifying environmental and urban sounds, as well as classifying visual data. In at least one embodiment, a CNN running on a DLA is trained to identify a relative closing speed of an emergency vehicle (e.g., by using a Doppler effect). In at least one embodiment, a CNN may also be trained to identify emergency vehicles specific to a local area in which a vehicle is operating, as identified by GNSS sensor(s) 1258. In at least one embodiment, when operating in Europe, a CNN will seek to detect European sirens, and when in North America, a CNN will seek to identify only North American sirens. In at least one embodiment, once an emergency vehicle is detected, a control program may be used to execute an emergency vehicle safety routine, slowing a vehicle, pulling over to a side of a road, parking a vehicle, and/or idling a vehicle, with assistance of ultrasonic sensor(s) 1262, until emergency vehicles pass.

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

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

In at least one embodiment, vehicle 1200 may further include network interface 1224 which may include, without limitation, wireless antenna(s) 1226 (e.g., one or more wireless antennas for different communication protocols, such as a cellular antenna, a Bluetooth antenna, etc.). In at least one embodiment, network interface 1224 may be used to enable wireless connectivity to Internet cloud services (e.g., with server(s) and/or other network devices), with other vehicles, and/or with computing devices (e.g., client devices of passengers). In at least one embodiment, to communicate with other vehicles, a direct link may be established between vehicle 120 and another vehicle and/or an indirect link may be established (e.g., across networks

and over the Internet). In at least one embodiment, direct links may be provided using a vehicle-to-vehicle communication link. In at least one embodiment, a vehicle-to-vehicle communication link may provide vehicle 1200 information about vehicles in proximity to vehicle 1200 (e.g., vehicles in front of, on a side of, and/or behind vehicle 1200). In at least one embodiment, such aforementioned functionality may be part of a cooperative adaptive cruise control functionality of vehicle 1200.

In at least one embodiment, network interface 1224 may include an SoC that provides modulation and demodulation functionality and enables controller(s) 1236 to communicate over wireless networks. In at least one embodiment, network interface 1224 may include a radio frequency front-end for up-conversion from baseband to radio frequency, and down conversion from radio frequency to baseband. In at least one embodiment, frequency conversions may be performed in any technically feasible fashion. For example, frequency conversions could be performed through well-known processes, and/or using super-heterodyne processes. In at least one embodiment, radio frequency front end functionality may be provided by a separate chip. In at least one embodiment, network interfaces may include wireless functionality for communicating over LTE, WCDMA, UMTS, GSM, CDMA2000, Bluetooth, Bluetooth LE, Wi-Fi, Z-Wave, Zig-Bee, LoRaWAN, and/or other wireless protocols.

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

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

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

In at least one embodiment, RADAR sensor(s) 1260 may include different configurations, such as long-range with narrow field of view, short-range with wide field of view, short-range side coverage, etc. In at least one embodiment, long-range RADAR may be used for adaptive cruise control functionality. In at least one embodiment, long-range RADAR systems may provide a broad field of view realized by two or more independent scans, such as within a 250 m

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

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

In at least one embodiment, vehicle 1200 may further include ultrasonic sensor(s) 1262. In at least one embodiment, ultrasonic sensor(s) 1262, which may be positioned at a front, a back, and/or side location of vehicle 1200, may be used for parking assist and/or to create and update an occupancy grid. In at least one embodiment, a wide variety of ultrasonic sensor(s) 1262 may be used, and different ultrasonic sensor(s) 1262 may be used for different ranges of detection (e.g., 2.5 m, 4 m). In at least one embodiment, ultrasonic sensor(s) 1262 may operate at functional safety levels of ASIL B.

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

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

mounted LIDAR sensor(s) 1264 may be configured for a horizontal field of view between 45 degrees and 135 degrees.

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

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

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

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

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

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

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

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

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

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

In at least one embodiment, an FCW system is designed to alert a driver to a hazard, so that such driver may take corrective action. In at least one embodiment, an FCW

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

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

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

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

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

In at least one embodiment, conventional ADAS systems may be prone to false positive results which may be annoying and distracting to a driver, but typically are not catastrophic, because conventional ADAS systems alert a driver and allow that driver to decide whether a safety condition truly exists and act accordingly. In at least one embodiment, vehicle 1200 itself decides, in case of conflicting results, whether to heed result from a primary computer or a

secondary computer (e.g., a first controller or a second controller of controllers 1236). For example, in at least one embodiment, ADAS system 1238 may be a backup and/or secondary computer for providing perception information to a backup computer rationality module. In at least one embodiment, a backup computer rationality monitor may run redundant diverse software on hardware components to detect faults in perception and dynamic driving tasks. In at least one embodiment, outputs from ADAS system 1238 may be provided to a supervisory MCU. In at least one embodiment, if outputs from a primary computer and outputs from a secondary computer conflict, a supervisory MCU determines how to reconcile conflict to ensure safe operation.

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

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

In at least one embodiment, ADAS system 1238 may include a secondary computer that performs ADAS functionality using traditional rules of computer vision. In at least one embodiment, that secondary computer may use classic computer vision rules (if-then), and presence of a neural network(s) in a supervisory MCU may improve reliability, safety and performance. For example, in at least one embodiment, diverse implementation and intentional non-identity makes an overall system more fault-tolerant, especially to faults caused by software (or software-hardware interface) functionality. For example, in at least one embodiment, if there is a software bug or error in software running on a primary computer, and non-identical software code running on a secondary computer provides a consistent overall result, then a supervisory MCU may have greater confidence that an overall result is correct, and a bug in software or hardware on that primary computer is not causing a material error.

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

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

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

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

examples, information may be displayed and/or shared among infotainment SoC 1230 and instrument cluster 1232. In at least one embodiment, instrument cluster 1232 may be included as part of infotainment SoC 1230, or vice versa.

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

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

In at least one embodiment, server(s) 1278 may be used to train machine learning models (e.g., neural networks) based at least in part on training data. In at least one embodiment, training data may be generated by vehicles, and/or may be generated in a simulation (e.g., using a game engine). In at least one embodiment, any amount of training data is tagged (e.g., where associated neural network benefits from supervised learning) and/or undergoes other pre-processing. In at least one embodiment, any amount of training data is not tagged and/or pre-processed (e.g., where associated neural network does not require supervised learning). In at least one embodiment, once machine learning models are trained, machine learning models may be used by vehicles (e.g., transmitted to vehicles over network(s) 1290), and/or machine learning models may be used by server(s) 1278 to remotely monitor vehicles.

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

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

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

#### Computer Systems

FIG. 13 is a block diagram illustrating an exemplary computer system, which may be a system with interconnected devices and components, a system-on-a-chip (SOC) or some combination thereof formed with a processor that may include execution units to execute an instruction, according to at least one embodiment. In at least one embodiment, a computer system 1300 may include, without limitation, a component, such as a processor 1302 to employ execution units including logic to perform algorithms for process data, in accordance with present disclosure, such as in embodiment described herein. In at least one embodiment,

computer system 1300 may include processors, such as PENTIUM® Processor family, Xeon™, Itanium®, XScale™ and/or StrongARM™, Intel® Core™, or Intel® Nervana™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and like) may also be used. In at least one embodiment, computer system 1300 may execute a version of WINDOWS operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems (UNIX and Linux, for example), embedded software, and/or graphical user interfaces, may also be used.

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

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

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

In at least one embodiment, execution unit 1308, including, without limitation, logic to perform integer and floating point operations, also resides in processor 1302. In at least one embodiment, processor 1302 may also include a microcode ("ucode") read only memory ("ROM") that stores microcode for certain macro instructions. In at least one embodiment, execution unit 1308 may include logic to handle a packed instruction set 1309. In at least one embodiment, by including packed instruction set 1309 in an instruction set of a general-purpose processor, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using

packed data in processor **1302**. In at least one embodiment, many multimedia applications may be accelerated and executed more efficiently by using a full width of a processor's data bus for performing operations on packed data, which may eliminate a need to transfer smaller units of data across that processor's data bus to perform one or more operations one data element at a time.

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

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

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

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

Inference and/or training logic **915** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or train-

ing logic **915** are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic **915** may be used in system FIG. 13 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

In at least one embodiment, electronic device **1400** may include, without limitation, processor **1410** communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor **1410** is coupled using a bus or interface, such as a 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 Universal Serial Bus ("USB") (versions 1, 2, 3, etc.), or a Universal Asynchronous Receiver/Transmitter ("UART") bus. In at least one embodiment, FIG. 14 illustrates a system, which includes interconnected hardware devices or "chips", whereas in other embodiments, FIG. 14 may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. 14 may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of FIG. 14 are interconnected using compute express link (CXL) interconnects.

In at least one embodiment, FIG. 14 may include a display **1424**, a touch screen **1425**, a touch pad **1430**, a Near Field Communications unit ("NFC") **1445**, a sensor hub **1440**, a thermal sensor **1446**, an Express Chipset ("EC") **1435**, a Trusted Platform Module ("TPM") **1438**, BIOS/firmware/flash memory ("BIOS, FW Flash") **1422**, a DSP **1460**, a drive **1420** such as a Solid State Disk ("SSD") or a Hard Disk Drive ("HDD"), a wireless local area network unit ("WLAN") **1450**, a Bluetooth unit **1452**, a Wireless Wide Area Network unit ("WWAN") **1456**, a Global Positioning System (GPS) unit **1455**, a camera ("USB 3.0 camera") **1454** such as a USB 3.0 camera, and/or a Low Power Double 55 Data Rate ("LPDDR") memory unit ("LPDDR3") **1415** implemented in, for example, an LPDDR3 standard. These components may each be implemented in any suitable manner.

In at least one embodiment, other components may be communicatively coupled to processor **1410** through components described herein. In at least one embodiment, an accelerometer **1441**, an ambient light sensor ("ALS") **1442**, a compass **1443**, and a gyroscope **1444** may be communicatively coupled to sensor hub **1440**. In at least one embodiment, a thermal sensor **1439**, a fan **1437**, a keyboard **1436**, and touch pad **1430** may be communicatively coupled to EC **1435**. In at least one embodiment, speakers **1463**, head-

phones 1464, and a microphone (“mic”) 1465 may be communicatively coupled to an audio unit (“audio codec and class D amp”) 1462, which may in turn be communicatively coupled to DSP 1460. In at least one embodiment, audio unit 1462 may include, for example and without limitation, an audio coder/decoder (“codec”) and a class D amplifier. In at least one embodiment, a SIM card (“SIM”) 1457 may be communicatively coupled to WWAN unit 1456. In at least one embodiment, components such as WLAN unit 1450 and Bluetooth unit 1452, as well as WWAN unit 1456 may be implemented in a Next Generation Form Factor (“NGFF”).

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

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

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

Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in system FIG. 15 for

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

5 In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to 10 perform various operations such as generating assets and modify broadcasts and/or streams.

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

15 In at least one embodiment, USB stick 1620 includes, without limitation, a processing unit 1630, a USB interface 1640, and USB interface logic 1650. In at least one embodiment, processing unit 1630 may be any instruction execution system, apparatus, or device capable of executing instructions. In at least one embodiment, processing unit 1630 may include, without limitation, any number and type of processing cores (not shown). In at least one embodiment, processing unit 1630 comprises an application specific integrated circuit (“ASIC”) that is optimized to perform any amount and type of operations associated with machine learning. For instance, in at least one embodiment, processing unit 1630 is a tensor processing unit (“TPC”) that is optimized to perform machine learning inference operations. 20 In at least one embodiment, processing unit 1630 is a vision processing unit (“VPU”) that is optimized to perform machine vision and machine learning inference operations.

25 In at least one embodiment, USB interface 1640 may be any type of USB connector or USB socket. For instance, in at least one embodiment, USB interface 1640 is a USB 3.0 Type-C socket for data and power. In at least one embodiment, USB interface 1640 is a USB 3.0 Type-A connector. In at least one embodiment, USB interface logic 1650 may include any amount and type of logic that enables processing unit 1630 to interface with devices (e.g., computer 1610) via 30 USB connector 1640.

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

40 In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to 45 perform various operations such as generating assets and modify broadcasts and/or streams.

45 FIG. 17A illustrates an exemplary architecture in which a plurality of GPUs 1710(1)-1710(N) is communicatively coupled to a plurality of multi-core processors 1705(1)-1705(M) over high-speed links 1740(1)-1740(N) (e.g., buses,

point-to-point interconnects, etc.). In at least one embodiment, high-speed links 1740(1)-1740(N) support a communication throughput of 4 GB/s, 30 GB/s, 80 GB/s or higher. In at least one embodiment, various interconnect protocols may be used including, but not limited to, PCIe 4.0 or 5.0 and NVLink 2.0. In various figures, “N” and “M” represent positive integers, values of which may be different from figure to figure.

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

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

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

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

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

illustrated. In at least one embodiment, caches 1762A-1762D may comprise Level 1 (L1) and Level 2 (L2) caches. In addition, one or more shared caches 1756 may be included in caches 1762A-1762D and shared by sets of cores 1760A-1760D. For example, one embodiment of processor 1707 includes 24 cores, each with its own L1 cache, twelve shared L2 caches, and twelve shared L3 caches. In this embodiment, one or more L2 and L3 caches are shared by two adjacent cores. In at least one embodiment, processor 1707 and graphics acceleration module 1746 connect with system memory 1714, which may include processor memories 1701(1)-1701(M) of FIG. 17A.

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

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

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

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

accesses of cache lines on processor caches 1762A-1762D, 1756 and receiving updates from cache 1738).

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

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

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

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

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

In at least one embodiment, to reduce data traffic over high-speed link 1740, biasing techniques can be used to ensure that data stored in graphics memories 1733(1)-1733(M) is data that will be used most frequently by graphics

processing engines 1731(1)-1731(N) and preferably not used by cores 1760A-1760D (at least not frequently). Similarly, in at least one embodiment, a biasing mechanism attempts to keep data needed by cores (and preferably not graphics processing engines 1731(1)-1731(N)) within caches 1762A-1762D, 1756 and system memory 1714.

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

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

In at least one embodiment, graphics processing engines 1731(1)-1731(N), may be shared by multiple VM/application partitions. In at least one embodiment, shared models 35 may use a system hypervisor to virtualize graphics processing engines 1731(1)-1731(N) to allow access by each operating system. In at least one embodiment, for single-partition systems without a hypervisor, graphics processing engines 1731(1)-1731(N) are owned by an operating system. In at least one embodiment, an operating system can virtualize graphics processing engines 1731(1)-1731(N) to provide access to each process or application.

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

FIG. 17D illustrates an exemplary accelerator integration slice 1790. In at least one embodiment, a “slice” comprises a specified portion of processing resources of accelerator integration circuit 1736. In at least one embodiment, an application is effective address space 1782 within system memory 1714 stores process elements 1783. In at least one embodiment, process elements 1783 are stored in response to GPU invocations 1781 from applications 1780 executed on processor 1707. In at least one embodiment, a process element 1783 contains process state for corresponding application 1780. In at least one embodiment, a work descriptor

(WD) **1784** contained in process element **1783** can be a single job requested by an application or may contain a pointer to a queue of jobs. In at least one embodiment, WD **1784** is a pointer to a job request queue in an application's effective address space **1782**.

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

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

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

In at least one embodiment, registers **1745** are duplicated for each graphics processing engine **1731(1)-1731(N)** and/or graphics acceleration module **1746** and may be initialized by a hypervisor or an operating system. In at least one embodiment, each of these duplicated registers may be included in an accelerator integration slice **1790**. Exemplary registers that may be initialized by a hypervisor are shown in Table 1.

TABLE 1

| Hypervisor Initialized Registers |   |
|----------------------------------|---|
| Register #                       | Description   |
| 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  |

Exemplary registers that may be initialized by an operating system are shown in Table 2.

TABLE 2

| Operating System Initialized Registers |   |
|--|---|
| Register #                             | Description   |
| 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   |

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

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

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

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

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

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

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

TABLE 3

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

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

TABLE 4

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

TABLE 4-continued

| Process Element Information |   |  |
|-----------------------------|---|--|
| Element #                   | Description   |  |
| 11                          | A real address (RA) hypervisor accelerator utilization record pointer |  |
| 12                          | Storage Descriptor Register (SDR)                                     |  |

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

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

In at least one embodiment, bias/coherence management circuitry 1794A-1794E within one or more of MMUs 1739A-1739E ensures cache coherence between caches of one or more host processors (e.g., 1705) and GPUs 1710 and implements biasing techniques indicating physical memories in which certain types of data should be stored. In at least one embodiment, while multiple instances of bias/coherence management circuitry 1794A-1794E are illustrated in FIG. 17F, bias/coherence circuitry may be implemented within an MMU of one or more host processors 1705 and/or within accelerator integration circuit 1736.

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

In at least one embodiment, selection of GPU bias and host processor bias is driven by a bias tracker data structure.

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

In at least one embodiment, a bias table entry associated with each access to a GPU attached memory 1720 is accessed prior to actual access to a GPU memory, causing following operations. In at least one embodiment, local requests from a GPU 1710 that find their page in GPU bias are forwarded directly to a corresponding GPU memory 1720. In at least one embodiment, local requests from a GPU that find their page in host bias are forwarded to processor 1705 (e.g., over a high-speed link as described herein). In at least one embodiment, requests from processor 1705 that find a requested page in host processor bias complete a request like a normal memory read. Alternatively, requests directed to a GPU-biased page may be forwarded to a GPU 1710. In at least one embodiment, a GPU may then transition a page to a host processor bias if it is not currently using a page. In at least one embodiment, a bias state of a page can be changed either by a software-based mechanism, a hardware-assisted software-based mechanism, or, for a limited set of cases, a purely hardware-based mechanism.

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

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

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

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

FIG. 18 is a block diagram illustrating an exemplary system on a chip integrated circuit 1800 that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, integrated circuit 1800 includes one or more application processor(s) 1805 (e.g., CPUs), at least one graphics processor 1810, and may additionally include an image processor 1815 and/or a

video processor 1820, any of which may be a modular IP core. In at least one embodiment, integrated circuit 1800 includes peripheral or bus logic including a USB controller 1825, a UART controller 1830, an SPI/SDIO controller 1835, and an I<sup>2</sup>S/I<sup>2</sup>C controller 1840. In at least one embodiment, integrated circuit 1800 can include a display device 1845 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1850 and a mobile industry processor interface (MIPI) display interface 1855. 10 In at least one embodiment, storage may be provided by a flash memory subsystem 1860 including flash memory and a flash memory controller. In at least one embodiment, a memory interface may be provided via a memory controller 1865 for access to SDRAM or SRAM memory devices. In 15 at least one embodiment, some integrated circuits additionally include an embedded security engine 1870.

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIGS. 19A-19B illustrate exemplary integrated circuits 35 and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included in at least one embodiment, including additional graphics processors/cores, peripheral interface controllers, or general-purpose processor cores.

FIGS. 19A-19B are block diagrams illustrating exemplary graphics processors for use within an SoC, according to embodiments described herein. FIG. 19A illustrates an exemplary graphics processor 1910 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. FIG. 19B illustrates an additional exemplary graphics processor 1940 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, graphics processor 1910 of FIG. 19A is a low power graphics processor core. In at least one embodiment, graphics processor 1940 of FIG. 19B is a higher performance graphics processor core. In at 45 least one embodiment, each of graphics processors 1910, 1940 can be variants of graphics processor 1810 of FIG. 18.

In at least one embodiment, graphics processor 1910 includes a vertex processor 1905 and one or more fragment processor(s) 1915A-1915N (e.g., 1915A, 1915B, 1915C, 1915D, through 1915N-1, and 1915N). In at least one embodiment, graphics processor 1910 can execute different shader programs via separate logic, such that vertex processor 1905 is optimized to execute operations for vertex shader programs, while one or more fragment processor(s) 1915A-1915N execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. In at least one embodiment, vertex processor 1905 performs a vertex processing

stage of a 3D graphics pipeline and generates primitives and vertex data. In at least one embodiment, fragment processor(s) 1915A-1915N use primitive and vertex data generated by vertex processor 1905 to produce a framebuffer that is displayed on a display device. In at least one embodiment, fragment processor(s) 1915A-1915N are optimized to execute fragment shader programs as provided for in an OpenGL API, which may be used to perform similar operations as a pixel shader program as provided for in a Direct 3D API.

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

In at least one embodiment, graphics processor 1940 includes one or more shader core(s) 1955A-1955N (e.g., 1955A, 1955B, 1955C, 1955D, 1955E, 1955F, through 1955N-1, and 1955N) as shown in FIG. 19B, which provides for a unified shader core architecture in which a single core or type or core can execute all types of programmable shader code, including shader program code to implement vertex shaders, fragment shaders, and/or compute shaders. In at least one embodiment, a number of shader cores can vary. In at least one embodiment, graphics processor 1940 includes an inter-core task manager 1945, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores 1955A-1955N and a tiling unit 1958 to accelerate tiling operations for tile-based rendering, in which rendering operations for a scene are subdivided in image space, for example to exploit local spatial coherence within a scene or to optimize use of internal caches.

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

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

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

least one embodiment, GPGPU 2030 can be linked directly to other instances of GPGPU 2030 to create a multi-GPU cluster to improve training speed for deep neural networks. In at least one embodiment, GPGPU 2030 includes a host interface 2032 to enable a connection with a host processor. In at least one embodiment, host interface 2032 is a PCI Express interface. In at least one embodiment, host interface 2032 can be a vendor-specific communications interface or communications fabric. In at least one embodiment, GPGPU 2030 receives commands from a host processor and uses a global scheduler 2034 to distribute execution threads associated with those commands to a set of compute clusters 2036A-2036H. In at least one embodiment, compute clusters 2036A-2036H share a cache memory 2038. In at least one embodiment, cache memory 2038 can serve as a higher-level cache for cache memories within compute clusters 2036A-2036H.

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

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

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

In at least one embodiment, GPGPU 2030 can be configured to train neural networks. In at least one embodiment, GPGPU 2030 can be used within an inferencing platform. In at least one embodiment, in which GPGPU 2030 is used for inferencing, GPGPU 2030 may include fewer compute clusters 2036A-2036H relative to when GPGPU 2030 is used for training a neural network. In at least one embodiment, memory technology associated with memory 2044A-

2044B may differ between inferencing and training configurations, with higher bandwidth memory technologies devoted to training configurations. In at least one embodiment, an inferencing configuration of GPGPU 2030 can support inferencing specific instructions. For example, in at least one embodiment, an inferencing configuration can provide support for one or more 8-bit integer dot product instructions, which may be used during inferencing operations for deployed neural networks.

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

In at least one embodiment, processing subsystem 2101 includes one or more parallel processor(s) 2112 coupled to memory hub 2105 via a bus or other communication link 2113. In at least one embodiment, communication link 2113 may use one of any number of standards-based communication link technologies or protocols, such as, but not limited to PCI Express, or may be a vendor-specific communications interface or communications fabric. In at least one embodiment, one or more parallel processor(s) 2112 form a computationally focused parallel or vector processing system that can include a large number of processing cores and/or processing clusters, such as a many-integrated core (MIC) processor. In at least one embodiment, some or all of parallel processor(s) 2112 form a graphics processing subsystem that can output pixels to one of one or more display device(s) 2110A coupled via I/O Hub 2107. In at least one embodiment, parallel processor(s) 2112 can also include a display controller and display interface (not shown) to enable a direct connection to one or more display device(s) 2110B.

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

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

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

#### Processors

FIG. 22A illustrates a parallel processor 2200 according to at least one embodiment. In at least one embodiment, various components of parallel processor 2200 may be implemented using one or more integrated circuit devices, such as programmable processors, application specific inte-

grated circuits (ASICs), or field programmable gate arrays (FPGA). In at least one embodiment, illustrated parallel processor 2200 is a variant of one or more parallel processor (s) 2112 shown in FIG. 21 according to an exemplary embodiment.

In at least one embodiment, parallel processor 2200 includes a parallel processing unit 2202. In at least one embodiment, parallel processing unit 2202 includes an I/O unit 2204 that enables communication with other devices, including other instances of parallel processing unit 2202. In at least one embodiment, I/O unit 2204 may be directly connected to other devices. In at least one embodiment, I/O unit 2204 connects with other devices via use of a hub or switch interface, such as a memory hub 2205. In at least one embodiment, connections between memory hub 2205 and I/O unit 2204 form a communication link 2213. In at least one embodiment, I/O unit 2204 connects with a host interface 2206 and a memory crossbar 2216, where host interface 2206 receives commands directed to performing processing operations and memory crossbar 2216 receives commands directed to performing memory operations.

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

In at least one embodiment, processing cluster array 2212 can include up to "N" processing clusters (e.g., cluster 2214A, cluster 2214B, through cluster 2214N), where "N" represents a positive integer (which may be a different integer "N" than used in other figures). In at least one embodiment, each cluster 2214A-2214N of processing cluster array 2212 can execute a large number of concurrent threads. In at least one embodiment, scheduler 2210 can allocate work to clusters 2214A-2214N of processing cluster array 2212 using various scheduling and/or work distribution algorithms, which may vary depending on workload arising for each type of program or computation. In at least one embodiment, scheduling can be handled dynamically by scheduler 2210, or can be assisted in part by compiler logic during compilation of program logic configured for execution by processing cluster array 2212. In at least one embodiment, different clusters 2214A-2214N of processing cluster array 2212 can be allocated for processing different types of programs or for performing different types of computations.

In at least one embodiment, processing cluster array 2212 can be configured to perform various types of parallel processing operations. In at least one embodiment, process-

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

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

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

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

In at least one embodiment, each of one or more instances of parallel processing unit 2202 can couple with a parallel processor memory 2222. In at least one embodiment, parallel processor memory 2222 can be accessed via memory crossbar 2216, which can receive memory requests from processing cluster array 2212 as well as I/O unit 2204. In at least one embodiment, memory crossbar 2216 can access parallel processor memory 2222 via a memory interface 2218. In at least one embodiment, memory interface 2218 can include multiple partition units (e.g., partition unit

2220A, partition unit 2220B, through partition unit 2220N) that can each couple to a portion (e.g., memory unit) of parallel processor memory 2222. In at least one embodiment, a number of partition units 2220A-2220N is configured to be equal to a number of memory units, such that a first partition unit 2220A has a corresponding first memory unit 2224A, a second partition unit 2220B has a corresponding memory unit 2224B, and an N-th partition unit 2220N has a corresponding N-th memory unit 2224N. In at least one embodiment, a number of partition units 2220A-2220N may not be equal to a number of memory units.

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

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

In at least one embodiment, multiple instances of parallel processing unit 2202 can be provided on a single add-in card, or multiple add-in cards can be interconnected. In at least one embodiment, different instances of parallel processing unit 2202 can be configured to interoperate even if different instances have different numbers of processing cores, different amounts of local parallel processor memory, and/or other configuration differences. For example, in at least one embodiment, some instances of parallel processing unit 2202 can include higher precision floating point units relative to other instances. In at least one embodiment, systems incorporating one or more instances of parallel processing unit 2202 or parallel processor 2200 can be implemented in a variety of configurations and form factors, including but not limited to desktop, laptop, or handheld personal computers, servers, workstations, game consoles, and/or embedded systems.

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

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

In at least one embodiment, ROP 2226 is included within each processing cluster (e.g., cluster 2214A-2214N of FIG. 22A) instead of within partition unit 2220. In at least one embodiment, read and write requests for pixel data are transmitted over memory crossbar 2216 instead of pixel fragment data. In at least one embodiment, processed graphics data may be displayed on a display device, such as one of one or more display device(s) 2110 of FIG. 21, routed for further processing by processor(s) 2102, or routed for further processing by one of processing entities within parallel processor 2200 of FIG. 22A.

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

In at least one embodiment, operation of processing cluster 2214 can be controlled via a pipeline manager 2232 that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager 2232 receives instructions from scheduler 2210 of FIG. 22A and manages execution of those instructions via a graphics multiprocessor

2234 and/or a texture unit 2236. In at least one embodiment, graphics multiprocessor 2234 is an exemplary instance of a SIMT parallel processor. However, in at least one embodiment, various types of SIMT parallel processors of differing architectures may be included within processing cluster 2214. In at least one embodiment, one or more instances of graphics multiprocessor 2234 can be included within a processing cluster 2214. In at least one embodiment, graphics multiprocessor 2234 can process data and a data crossbar 2240 can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager 2232 can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data crossbar 2240.

In at least one embodiment, each graphics multiprocessor 2234 within processing cluster 2214 can include an identical set of functional execution logic (e.g., arithmetic logic units, load-store units, etc.). In at least one embodiment, functional execution logic can be configured in a pipelined manner in which new instructions can be issued before previous instructions are complete. In at least one embodiment, functional execution logic supports a variety of operations including integer and floating point arithmetic, comparison operations, Boolean operations, bit-shifting, and computation of various algebraic functions. In at least one embodiment, same functional-unit hardware can be leveraged to perform different operations and any combination of functional units may be present.

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

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

least one embodiment, processing cluster 2214 includes multiple instances of graphics multiprocessor 2234 and can share common instructions and data, which may be stored in L1 cache 2248.

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

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

Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment, inference and/or training logic 915 may be used in graphics processing cluster 2214 for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 22D shows a graphics multiprocessor 2234 according to at least one embodiment. In at least one embodiment, graphics multiprocessor 2234 couples with pipeline manager 2232 of processing cluster 2214. In at least one embodiment, graphics multiprocessor 2234 has an execution pipeline including but not limited to an instruction cache 2252,

an instruction unit 2254, an address mapping unit 2256, a register file 2258, one or more general purpose graphics processing unit (GPGPU) cores 2262, and one or more load/store units 2266. In at least one embodiment, GPGPU cores 2262 and load/store units 2266 are coupled with cache memory 2272 and shared memory 2270 via a memory and cache interconnect 2268.

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

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

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

In at least one embodiment, GPGPU cores 2262 include SIMD logic capable of performing a single instruction on multiple sets of data. In at least one embodiment, GPGPU cores 2262 can physically execute SIMD4, SIMD8, and SIMD16 instructions and logically execute SIMD1, SIMD2, and SIMD32 instructions. In at least one embodiment, SIMD instructions for GPGPU cores can be generated at compile time by a shader compiler or automatically generated when executing programs written and compiled for single program multiple data (SPMD) or SIMT architectures. In at least one embodiment, multiple threads of a program configured for an SIMT execution model can be executed via a single SIMD instruction. For example, in at least one embodiment, eight SIMT threads that perform same or similar operations can be executed in parallel via a single SIMD8 logic unit.

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

In at least one embodiment, a parallel processor or GPGPU as described herein is communicatively coupled to host/processor cores to accelerate graphics operations, machine-learning operations, pattern analysis operations, and various general purpose GPU (GPGPU) functions. In at least one embodiment, a GPU may be communicatively coupled to host processor/cores over a bus or other interconnect (e.g., a high-speed interconnect such as PCIe or NVLink). In at least one embodiment, a GPU may be integrated on a package or chip as cores and communicatively coupled to cores over an internal processor bus/interconnect internal to a package or chip. In at least one embodiment, regardless a manner in which a GPU is connected, processor cores may allocate work to such GPU in a form of sequences of commands/instructions contained in a work descriptor. In at least one embodiment, that GPU then uses dedicated circuitry/logic for efficiently processing these commands/instructions.

Inference and/or training logic **915** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **915** are provided herein in conjunction with FIGS. **9A** and/or **9B**. In at least one embodiment, inference and/or training logic **915** may be used in graphics multiprocessor **2234** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. **23** illustrates a multi-GPU computing system **2300**, according to at least one embodiment. In at least one embodiment, multi-GPU computing system **2300** can include a processor **2302** coupled to multiple general purpose graphics processing units (GPGPUs) **2306A-D** via a host interface switch **2304**. In at least one embodiment, host interface switch **2304** is a PCI express switch device that couples processor **2302** to a PCI express bus over which processor **2302** can communicate with GPGPUs **2306A-D**.

In at least one embodiment, GPGPUs **2306A-D** can interconnect via a set of high-speed point-to-point GPU-to-GPU links **2316**. In at least one embodiment, GPU-to-GPU links **2316** connect to each of GPGPUs **2306A-D** via a dedicated GPU link. In at least one embodiment, P2P GPU links **2316** enable direct communication between each of GPGPUs **2306A-D** without requiring communication over host interface bus **2304** to which processor **2302** is connected. In at least one embodiment, with GPU-to-GPU traffic directed to P2P GPU links **2316**, host interface bus **2304** remains available for system memory access or to communicate with other instances of multi-GPU computing system **2300**, for example, via one or more network devices. While in at least one embodiment GPGPUs **2306A-D** connect to processor **2302** via host interface switch **2304**, in at least one embodiment processor **2302** includes direct support for P2P GPU links **2316** and can connect directly to GPGPUs **2306A-D**.

Inference and/or training logic **915** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **915** are provided herein in conjunction with FIGS. **9A** and/or **9B**. In at least one embodiment, inference and/or training logic **915** may be used in multi-GPU computing system **2300** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

In various embodiments, inference and/or training logic **915** are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

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

In at least one embodiment, graphics processor 2400 includes scalable thread execution resources featuring graphics cores 2480A-2480N (which can be modular and are sometimes referred to as core slices), each having multiple sub-cores 2450A-50N, 2460A-2460N (sometimes referred to as core sub-slices). In at least one embodiment, graphics processor 2400 can have any number of graphics cores 2480A. In at least one embodiment, graphics processor 2400 includes a graphics core 2480A having at least a first sub-core 2450A and a second sub-core 2460A. In at least one embodiment, graphics processor 2400 is a low power processor with a single sub-core (e.g., 2450A). In at least one embodiment, graphics processor 2400 includes multiple graphics cores 2480A-2480N, each including a set of first sub-cores 2450A-2450N and a set of second sub-cores 2460A-2460N. In at least one embodiment, each sub-core in first sub-cores 2450A-2450N includes at least a first set of execution units 2452A-2452N and media/texture samplers 2454A-2454N. In at least one embodiment, each sub-core in second sub-cores 2460A-2460N includes at least a second set of execution units 2462A-2462N and samplers 2464A-2464N. In at least one embodiment, each sub-core 2450A-2450N, 2460A-2460N shares a set of shared resources 2470A-2470N. In at least one embodiment, shared resources include shared cache memory and pixel operation logic.

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

In at least one embodiment, processor 2500 includes an in-order front end (“front end”) 2501 to fetch instructions to be executed and prepare instructions to be used later in a processor pipeline. In at least one embodiment, front end

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

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

In at least one embodiment, out-of-order execution engine 40 (“out of order engine”) 2503 may prepare instructions for execution. In at least one embodiment, out-of-order execution logic has a number of buffers to smooth out and re-order flow of instructions to optimize performance as they go down a pipeline and get scheduled for execution. In at least one embodiment, out-of-order execution engine 2503 includes, without limitation, an allocator/register renamer 2540, a memory uop queue 2542, an integer/floating point uop queue 2544, a memory scheduler 2546, a fast scheduler 2502, a slow/general floating point scheduler (“slow/general FP scheduler”) 2504, and a simple floating point scheduler (“simple FP scheduler”) 2506. In at least one embodiment, fast schedule 2502, slow/general floating point scheduler 2504, and simple floating point scheduler 2506 are also collectively referred to herein as “uop schedulers 2502, 2504, 2506.” In at least one embodiment, allocator/register renamer 2540 allocates machine buffers and resources that each uop needs in order to execute. In at least one embodiment, allocator/register renamer 2540 renames logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer 2540 also allocates an entry for each uop in one of two uop queues, memory uop queue 2542 for memory operations and integer/floating point uop queue 2544 for non-memory operations, in front of memory scheduler 2546 and up schedulers 2502, 2504, 2506. In at least one embodiment, uop schedulers 2502, 2504, 2506, determine when a uop is ready to execute based on readiness of their dependent input register operand sources and availability of

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

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

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

In at least one embodiment, execution units **2512**, **2514**, **2516**, **2518**, **2520**, **2522**, **2524** may execute instructions. In at least one embodiment, register networks **2508**, **2510** store integer and floating point data operand values that micro-instructions need to execute. In at least one embodiment, processor **2500** may include, without limitation, any number and combination of execution units **2512**, **2514**, **2516**, **2518**, **2520**, **2522**, **2524**. In at least one embodiment, floating point ALU **2522** and floating point move unit **2524**, may execute floating point, MMX, SIMD, AVX and SSE, or other operations, including specialized machine learning instructions. In at least one embodiment, floating point ALU **2522** may include, without limitation, a 64-bit by 64-bit floating point divider to execute divide, square root, and remainder micro ops. In at least one embodiment, instructions involving a floating point value may be handled with floating point hardware. In at least one embodiment, ALU operations may be passed to fast ALUs **2516**, **2518**. In at least one embodiment, fast ALUs **2516**, **2518** may execute fast operations

with an effective latency of half a clock cycle. In at least one embodiment, most complex integer operations go to slow ALU **2520** as slow ALU **2520** may include, without limitation, integer execution hardware for long-latency type of operations, such as a multiplier, shifts, flag logic, and branch processing. In at least one embodiment, memory load/store operations may be executed by AGUs **2512**, **2514**. In at least one embodiment, fast ALU **2516**, fast ALU **2518**, and slow ALU **2520** may perform integer operations on 64-bit data 10 operands. In at least one embodiment, fast ALU **2516**, fast ALU **2518**, and slow ALU **2520** may be implemented to support a variety of data bit sizes including sixteen, thirty-two, 128, 256, etc. In at least one embodiment, floating point ALU **2522** and floating point move unit **2524** may be 15 implemented to support a range of operands having bits of various widths, such as 128-bit wide packed data operands in conjunction with SIMD and multimedia instructions.

In at least one embodiment, uop schedulers **2502**, **2504**, **2506** dispatch dependent operations before a parent load has 20 finished executing. In at least one embodiment, as uops may be speculatively scheduled and executed in processor **2500**, processor **2500** may also include logic to handle memory misses. In at least one embodiment, if a data load misses in a data cache, there may be dependent operations in flight in 25 a pipeline that have left a scheduler with temporarily incorrect data. In at least one embodiment, a replay mechanism tracks and re-executes instructions that use incorrect data. In at least one embodiment, dependent operations might need to be replayed and independent ones may be allowed to complete. In at least one embodiment, schedulers and a replay mechanism of at least one embodiment of a processor 30 may also be designed to catch instruction sequences for text string comparison operations.

In at least one embodiment, “registers” may refer to 35 on-board processor storage locations that may be used as part of instructions to identify operands. In at least one embodiment, registers may be those that may be usable from outside of a processor (from a programmer’s perspective). In at least one embodiment, registers might not be limited to a particular type of circuit. Rather, in at least one embodiment, a register may store data, provide data, and perform functions described herein. In at least one embodiment, registers described herein may be implemented by circuitry within a processor using any number of different techniques, such as 40 dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. In at least one embodiment, integer registers store 32-bit integer data. A register file of at least one embodiment also contains eight 45 multimedia SIMD registers for packed data.

Inference and/or training logic **915** are used to perform 50 inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **915** are provided herein in conjunction with FIGS. 55 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic **915** may be incorporated into execution block **2511** and other memory or registers shown or not shown. For example, in at least one embodiment, training and/or inferencing techniques described herein may 50 use one or more of ALUs illustrated in execution block **2511**. Moreover, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of execution block **2511** to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

In various embodiments, inference and/or training logic 60 are used to perform inferencing and/or training operations

associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 26 illustrates a deep learning application processor 2600, according to at least one embodiment. In at least one embodiment, deep learning application processor 2600 uses instructions that, if executed by deep learning application processor 2600, cause deep learning application processor 2600 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, deep learning application processor 2600 is an application-specific integrated circuit (ASIC). In at least one embodiment, application processor 2600 performs matrix multiply operations either “hard-wired” into hardware as a result of performing one or more instructions or both. In at least one embodiment, deep learning application processor 2600 includes, without limitation, processing clusters 2610 (1)-2610(12), Inter-Chip Links (“ICLs”) 2620(1)-2620(12), Inter-Chip Controllers (“ICCs”) 2630(1)-2630(2), high-bandwidth memory second generation (“HBM2”) 2640(1)-2640(4), memory controllers (“Mem Ctrlrs”) 2642(1)-2642(4), high bandwidth memory physical layer (“HBM PHY”) 2644(1)-2644(4), a management-controller central processing unit (“management-controller CPU”) 2650, a Serial Peripheral Interface, Inter-Integrated Circuit, and General Purpose Input/Output block (“SPI, I<sup>2</sup>C, GPIO”) 2660, a peripheral component interconnect express controller and direct memory access block (“PCIe Controller and DMA”) 2670, and a sixteen-lane peripheral component interconnect express port (“PCI Express ×16”) 2680.

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

In at least one embodiment, HBM2s 2640 provide a total of 32 Gigabytes (GB) of memory. In at least one embodiment, HBM2 2640(i) is associated with both memory controller 2642(i) and HBM PHY 2644(i) where “i” is an arbitrary integer. In at least one embodiment, any number of HBM2s 2640 may provide any type and total amount of high bandwidth memory and may be associated with any number (including zero) and type of memory controllers 2642 and HBM PHYs 2644. In at least one embodiment, SPI, I<sup>2</sup>C, GPIO 2660, PCIe Controller and DMA 2670, and/or PCIe 2680 may be replaced with any number and type of blocks that enable any number and type of communication standards in any technically feasible fashion.

Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS.

9A and/or 9B. In at least one embodiment, deep learning application processor is used to train a machine learning model, such as a neural network, to predict or infer information provided to deep learning application processor 2600. In at least one embodiment, deep learning application processor 2600 is used to infer or predict information based on a trained machine learning model (e.g., neural network) that has been trained by another processor or system or by deep learning application processor 2600. In at least one embodiment, processor 2600 may be used to perform one or more neural network use cases described herein.

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 27 is a block diagram of a neuromorphic processor 2700, according to at least one embodiment. In at least one embodiment, neuromorphic processor 2700 may receive one or more inputs from sources external to neuromorphic processor 2700. In at least one embodiment, these inputs may be transmitted to one or more neurons 2702 within neuromorphic processor 2700. In at least one embodiment, neurons 2702 and components thereof may be implemented using circuitry or logic, including one or more arithmetic logic units (ALUs). In at least one embodiment, neuromorphic processor 2700 may include, without limitation, thousands or millions of instances of neurons 2702, but any suitable number of neurons 2702 may be used. In at least one embodiment, each instance of neuron 2702 may include a neuron input 2704 and a neuron output 2706. In at least one embodiment, neurons 2702 may generate outputs that may be transmitted to inputs of other instances of neurons 2702. For example, in at least one embodiment, neuron inputs 2704 and neuron outputs 2706 may be interconnected via synapses 2708.

In at least one embodiment, neurons 2702 and synapses 2708 may be interconnected such that neuromorphic processor 2700 operates to process or analyze information received by neuromorphic processor 2700. In at least one embodiment, neurons 2702 may transmit an output pulse (or “fire” or “spike”) when inputs received through neuron input 2704 exceed a threshold. In at least one embodiment, neurons 2702 may sum or integrate signals received at neuron inputs 2704. For example, in at least one embodiment, neurons 2702 may be implemented as leaky integrate-and-fire neurons, wherein if a sum (referred to as a “membrane potential”) exceeds a threshold value, neuron 2702 may generate an output (or “fire”) using a transfer function such as a sigmoid or threshold function. In at least one embodiment, a leaky integrate-and-fire neuron may sum signals received at neuron inputs 2704 into a membrane potential and may also apply a decay factor (or leak) to reduce a membrane potential. In at least one embodiment, a leaky integrate-and-fire neuron may fire if multiple input signals are received at neuron inputs 2704 rapidly enough to exceed a threshold value (i.e., before a membrane potential decays too low to fire). In at least one embodiment, neurons 2702 may be implemented using circuits or logic that receive inputs, integrate inputs into a membrane potential, and decay a membrane potential. In at least one embodiment, inputs may be averaged, or any other suitable transfer function may be used. Furthermore, in at least one embodiment, neurons 2702 may include, without limitation, comparator circuits or logic that generate an output spike at

neuron output 2706 when result of applying a transfer function to neuron input 2704 exceeds a threshold. In at least one embodiment, once neuron 2702 fires, it may disregard previously received input information by, for example, resetting a membrane potential to 0 or another suitable default value. In at least one embodiment, once membrane potential is reset to 0, neuron 2702 may resume normal operation after a suitable period of time (or refractory period).

In at least one embodiment, neurons 2702 may be interconnected through synapses 2708. In at least one embodiment, synapses 2708 may operate to transmit signals from an output of a first neuron 2702 to an input of a second neuron 2702. In at least one embodiment, neurons 2702 may transmit information over more than one instance of synapse 2708. In at least one embodiment, one or more instances of neuron output 2706 may be connected, via an instance of synapse 2708, to an instance of neuron input 2704 in same neuron 2702. In at least one embodiment, an instance of neuron 2702 generating an output to be transmitted over an instance of synapse 2708 may be referred to as a “pre-synaptic neuron” with respect to that instance of synapse 2708. In at least one embodiment, an instance of neuron 2702 receiving an input transmitted over an instance of synapse 2708 may be referred to as a “post-synaptic neuron” with respect to that instance of synapse 2708. Because an instance of neuron 2702 may receive inputs from one or more instances of synapse 2708, and may also transmit outputs over one or more instances of synapse 2708, a single instance of neuron 2702 may therefore be both a “pre-synaptic neuron” and “post-synaptic neuron,” with respect to various instances of synapses 2708, in at least one embodiment.

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

In at least one embodiment, neuromorphic processor 2700 may include, without limitation, a reconfigurable interconnect architecture or dedicated hard-wired interconnects to connect synapse 2708 to neurons 2702. In at least one embodiment, neuromorphic processor 2700 may include, without limitation, circuitry or logic that allows synapses to be allocated to different neurons 2702 as needed based on

neural network topology and neuron fan-in/out. For example, in at least one embodiment, synapses 2708 may be connected to neurons 2702 using an interconnect fabric, such as network-on-chip, or with dedicated connections. In at least one embodiment, synapse interconnections and components thereof may be implemented using circuitry or logic.

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

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

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

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

integer registers, floating point registers, status registers, and an instruction pointer register). In at least one embodiment, register file 2806 may include general-purpose registers or other registers.

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

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

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

interface bus 2810. In at least one embodiment, audio controller 2846 is a multi-channel high definition audio controller. In at least one embodiment, system 2800 includes an optional legacy I/O controller 2840 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to system 2800. In at least one embodiment, platform controller hub 2830 can also connect to one or more Universal Serial Bus (USB) controllers 2842 connect input devices, such as keyboard and mouse 2843 combinations, a camera 2844, or other USB input devices.

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

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

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

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

In at least one embodiment, processor 2900 may also include a set of one or more bus controller units 2916 and a system agent core 2910. In at least one embodiment, bus controller units 2916 manage a set of peripheral buses, such as one or more PCI or PCI express busses. In at least one

## 101

embodiment, system agent core 2910 provides management functionality for various processor components. In at least one embodiment, system agent core 2910 includes one or more integrated memory controllers 2914 to manage access to various external memory devices (not shown).

In at least one embodiment, one or more of processor cores 2902A-2902N include support for simultaneous multi-threading. In at least one embodiment, system agent core 2910 includes components for coordinating and operating cores 2902A-2902N during multi-threaded processing. In at least one embodiment, system agent core 2910 may additionally include a power control unit (PCU), which includes logic and components to regulate one or more power states of processor cores 2902A-2902N and graphics processor 2908.

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

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

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

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

Inference and/or training logic 915 are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic 915 are provided herein in conjunction with FIGS. 9A and/or 9B. In at least one embodiment portions or all of inference and/or training logic 915 may be incorporated into graphics processor 2908. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in

## 102

a 3D pipeline, graphics core(s) 2902, shared function logic, or other logic in FIG. 29. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. 9A or 9B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of processor 2900 to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

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

In at least one embodiment, graphics processor 3000 also includes a display controller 3002 to drive display output data to a display device 3020. In at least one embodiment, display controller 3002 includes hardware for one or more overlay planes for display device 3020 and composition of multiple layers of video or user interface elements. In at least one embodiment, display device 3020 can be an internal or external display device. In at least one embodiment, display device 3020 is a head mounted display device, such as a virtual reality (VR) display device or an augmented reality (AR) display device. In at least one embodiment, graphics processor 3000 includes a video codec engine 3006 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

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

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

In at least one embodiment, media pipeline 3016 includes fixed function or programmable logic units to perform one or more specialized media operations, such as video decode

103

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

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

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

In at least one embodiment, GPE **3110** is coupled to or includes a command streamer **3103**, which provides a command stream to a 3D pipeline **3112** and/or media pipeline **3116**. In at least one embodiment, command streamer **3103** is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In at least one embodiment, command streamer **3103** receives commands from memory and sends commands to 3D pipeline **3112** and/or media pipeline **3116**. In at least one embodiment, commands are instructions, primitives, or micro-operations fetched from a ring buffer, which

104

stores commands for 3D pipeline **3112** and media pipeline **3116**. In at least one embodiment, a ring buffer can additionally include batch command buffers storing batches of multiple commands. In at least one embodiment, commands for 3D pipeline **3112** can also include references to data stored in memory, such as, but not limited to, vertex and geometry data for 3D pipeline **3112** and/or image data and memory objects for media pipeline **3116**. In at least one embodiment, 3D pipeline **3112** and media pipeline **3116** process commands and data by performing operations or by dispatching one or more execution threads to a graphics core array **3114**. In at least one embodiment, graphics core array **3114** includes one or more blocks of graphics cores (e.g., graphics core(s) **3115A**, graphics core(s) **3115B**), each block including one or more graphics cores. In at least one embodiment, each graphics core includes a set of graphics execution resources that includes general-purpose and graphics specific execution logic to perform graphics and compute operations, as well as fixed function texture processing and/or machine learning and artificial intelligence acceleration logic, including inference and/or training logic **915** in FIG. **9A** and FIG. **9B**.

In at least one embodiment, 3D pipeline **3112** includes fixed function and programmable logic to process one or more shader programs, such as vertex shaders, geometry shaders, pixel shaders, fragment shaders, compute shaders, or other shader programs, by processing instructions and dispatching execution threads to graphics core array **3114**. In at least one embodiment, graphics core array **3114** provides a unified block of execution resources for use in processing shader programs. In at least one embodiment, a multi-purpose execution logic (e.g., execution units) within graphics core(s) **3115A-3115B** of graphic core array **3114** includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

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

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

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

In at least one embodiment, graphics core array **3114** is coupled to shared function logic **3120** that includes multiple resources that are shared between graphics cores in graphics core array **3114**. In at least one embodiment, shared functions performed by shared function logic **3120** are embodied in hardware logic units that provide specialized supplemental functionality to graphics core array **3114**. In at least one

## 105

embodiment, shared function logic 3120 includes but is not limited to a sampler unit 3121, a math unit 3122, and inter-thread communication (ITC) logic 3123. In at least one embodiment, one or more cache(s) 3125 are included in, or coupled to, shared function logic 3120.

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 32 is a block diagram of hardware logic of a graphics processor core 3200, according to at least one embodiment described herein. In at least one embodiment, graphics processor core 3200 is included within a graphics core array. In at least one embodiment, graphics processor core 3200, sometimes referred to as a core slice, can be one or multiple graphics cores within a modular graphics processor. In at least one embodiment, graphics processor core 3200 is exemplary of one graphics core slice, and a graphics processor as described herein may include multiple graphics core slices based on target power and performance envelopes. In at least one embodiment, each graphics core 3200 can include a fixed function block 3230 coupled with multiple sub-cores 3201A-3201F, also referred to as sub-slices, that include modular blocks of general-purpose and fixed function logic.

## 106

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

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

15 In at least one embodiment, SoC interface 3237 enables graphics core 3200 to communicate with general-purpose application processor cores (e.g., CPUs) and/or other components within an SoC, including memory hierarchy elements such as a shared last level cache memory, system RAM, and/or embedded on-chip or on-package DRAM. In at least one embodiment, SoC interface 3237 can also enable communication with fixed function devices within an SoC, such as camera imaging pipelines, and enables use of and/or implements global memory atomics that may be shared between graphics core 3200 and CPUs within an SoC. In at least one embodiment, graphics SoC interface 3237 can also implement power management controls for graphics processor core 3200 and enable an interface between a clock domain of graphics processor core 3200 and other clock domains within an SoC. In at least one embodiment, SoC interface 3237 enables receipt of command buffers from a command streamer and global thread dispatcher that are configured to provide commands and instructions to each of 20 one or more graphics cores within a graphics processor. In at least one embodiment, commands and instructions can be dispatched to media pipeline 3239, when media operations are to be performed, or a geometry and fixed function pipeline (e.g., geometry and fixed function pipeline 3236, and/or a geometry and fixed function pipeline 3214) when graphics processing operations are to be performed.

25 In at least one embodiment, graphics microcontroller 3238 can be configured to perform various scheduling and management tasks for graphics core 3200. In at least one embodiment, graphics microcontroller 3238 can perform graphics and/or compute workload scheduling on various graphics parallel engines within execution unit (EU) arrays 3202A-3202F, 3204A-3204F within sub-cores 3201A-3201F. In at least one embodiment, host software executing 30 on a CPU core of an SoC including graphics core 3200 can submit workloads to one of multiple graphic processor paths, which invokes a scheduling operation on an appropriate graphics engine. In at least one embodiment, scheduling operations include determining which workload to run next, submitting a workload to a command streamer, preempting existing workloads running on an engine, monitoring progress of a workload, and notifying host software

when a workload is complete. In at least one embodiment, graphics microcontroller **3238** can also facilitate low-power or idle states for graphics core **3200**, providing graphics core **3200** with an ability to save and restore registers within graphics core **3200** across low-power state transitions independently from an operating system and/or graphics driver software on a system.

In at least one embodiment, graphics core **3200** may have greater than or fewer than illustrated sub-cores **3201A-3201F**, up to N modular sub-cores. For each set of N sub-cores, in at least one embodiment, graphics core **3200** can also include shared function logic **3210**, shared and/or cache memory **3212**, geometry/fixed function pipeline **3214**, as well as additional fixed function logic **3216** to accelerate various graphics and compute processing operations. In at least one embodiment, shared function logic **3210** can include logic units (e.g., sampler, math, and/or inter-thread communication logic) that can be shared by each N sub-cores within graphics core **3200**. In at least one embodiment, shared and/or cache memory **3212** can be a last-level cache for N sub-cores **3201A-3201F** within graphics core **3200** and can also serve as shared memory that is accessible by multiple sub-cores. In at least one embodiment, geometry/fixed function pipeline **3214** can be included instead of geometry/fixed function pipeline **3236** within fixed function block **3230** and can include similar logic units.

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

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

In at least one embodiment, within each graphics sub-core **3201A-3201F** includes a set of execution resources that may be used to perform graphics, media, and compute operations in response to requests by graphics pipeline, media pipeline, or shader programs. In at least one embodiment, graphics

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

Inference and/or training logic **915** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **915** are provided herein in conjunction with FIGS. **9A** and/or **9B**. In at least one embodiment, portions or all of inference and/or training logic **915** may be incorporated into graphics processor **3200**. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in a 3D pipeline, graphics microcontroller **3238**, geometry and fixed function pipeline **3214** and **3236**, or other logic in FIG. **32**. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. **9A** or **9B**. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor **3200** to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

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

As illustrated in FIG. **33A**, in at least one embodiment, thread execution logic **3300** includes a shader processor **3302**, a thread dispatcher **3304**, an instruction cache **3306**, a scalable execution unit array including a plurality of execution units **3307A-3307N** and **3308A-3308N**, a sampler **3310**, a data cache **3312**, and a data port **3314**. In at least one embodiment, a scalable execution unit array can dynamically scale by enabling or disabling one or more execution units (e.g., any of execution unit **3308A-N** or **3307A-N**) based on computational requirements of a workload, for

109

example. In at least one embodiment, scalable execution units are interconnected via an interconnect fabric that links to each execution unit. In at least one embodiment, thread execution logic 3300 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 3306, data port 3314, sampler 3310, and execution units 3307 or 3308. In at least one embodiment, each execution unit (e.g., 3307A) is a stand-alone programmable general-purpose computational unit that is capable of executing multiple simultaneous hardware threads while processing multiple data elements in parallel for each thread. In at least one embodiment, array of execution units 3307 and/or 3308 is scalable to include any number individual execution units.

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

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

In at least one embodiment, each execution unit in execution units 3307 and/or 3308 operates on arrays of data elements. In at least one embodiment, a number of data elements is an “execution size,” or number of channels for an instruction. In at least one embodiment, an execution

110

channel is a logical unit of execution for data element access, masking, and flow control within instructions. In at least one embodiment, a number of channels may be independent of a number of physical arithmetic logic units (ALUs) or floating point units (FPUs) for a particular graphics processor. In at least one embodiment, execution units 3307 and/or 3308 support integer and floating-point data types.

In at least one embodiment, an execution unit instruction set includes SIMD instructions. In at least one embodiment, various data elements can be stored as a packed data type in a register and execution unit will process various elements based on data size of elements. For example, in at least one embodiment, when operating on a 256-bit wide vector, 256 bits of a vector are stored in a register and an execution unit operates on a vector as four separate 64-bit packed data elements (Quad-Word (QW) size data elements), eight separate 32-bit packed data elements (Double Word (DW) size data elements), sixteen separate 16-bit packed data elements (Word (W) size data elements), or thirty-two separate 8-bit data elements (byte (B) size data elements). However, in at least one embodiment, different vector widths and register sizes are possible.

In at least one embodiment, one or more execution units 25 can be combined into a fused execution unit 3309A-3309N having thread control logic (3311A-3311N) that is common to fused EUs such as execution unit 3307A fused with execution unit 3308A into fused execution unit 3309A. In at least one embodiment, multiple EUs can be fused into an EU 30 group. In at least one embodiment, each EU in a fused EU group can be configured to execute a separate SIMD hardware thread, with a number of EUs in a fused EU group possibly varying according to various embodiments. In at least one embodiment, various SIMD widths can be performed per-EU, including but not limited to SIMD8, SIMD16, and SIMD32. In at least one embodiment, each fused graphics execution unit 3309A-3309N includes at least two execution units. For example, in at least one embodiment, fused execution unit 3309A includes a first EU 40 3307A, second EU 3308A, and thread control logic 3311A that is common to first EU 3307A and second EU 3308A. In at least one embodiment, thread control logic 3311A controls threads executed on fused graphics execution unit 3309A, allowing each EU within fused execution units 45 3309A-3309N to execute using a common instruction pointer register.

In at least one embodiment, one or more internal instruction caches (e.g., 3306) are included in thread execution logic 3300 to cache thread instructions for execution units. 50 In at least one embodiment, one or more data caches (e.g., 3312) are included to cache thread data during thread execution. In at least one embodiment, sampler 3310 is included to provide texture sampling for 3D operations and media sampling for media operations. In at least one embodiment, sampler 3310 includes specialized texture or media sampling functionality to process texture or media data during sampling process before providing sampled data to an execution unit.

During execution, in at least one embodiment, graphics 60 and media pipelines send thread initiation requests to thread execution logic 3300 via thread spawning and dispatch logic. In at least one embodiment, once a group of geometric objects has been processed and rasterized into pixel data, pixel processor logic (e.g., pixel shader logic, fragment shader logic, etc.) within shader processor 3302 is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth

111

buffers, stencil buffers, etc.). In at least one embodiment, a pixel shader or a fragment shader calculates values of various vertex attributes that are to be interpolated across a rasterized object. In at least one embodiment, pixel processor logic within shader processor 3302 then executes an application programming interface (API)-supplied pixel or fragment shader program. In at least one embodiment, to execute a shader program, shader processor 3302 dispatches threads to an execution unit (e.g., 3308A) via thread dispatcher 3304. In at least one embodiment, shader processor 3302 uses texture sampling logic in sampler 3310 to access texture data in texture maps stored in memory. In at least one embodiment, arithmetic operations on texture data and input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

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

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

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

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

112

ing modes can permit registers to be addressed together to build effectively wider registers or to represent strided rectangular block data structures.

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

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

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

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

FIG. 34 illustrates a parallel processing unit (“PPU”) 3400, according to at least one embodiment. In at least one embodiment, PPU 3400 is configured with machine-readable code that, if executed by PPU 3400, causes PPU 3400 to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, PPU 3400 is a multi-threaded processor that is implemented on one or more integrated circuit devices and that utilizes multithreading as a latency-hiding technique designed to process computer-readable instructions (also referred to as machine-readable instructions or simply instructions) on multiple threads in parallel. In at least one embodiment, a thread refers to a thread of execution and is an instantiation of a set of instructions configured to be executed by PPU 3400. In at least one embodiment, PPU 3400 is a graphics processing unit (“GPU”) configured to implement a graphics rendering pipeline for processing three-dimensional (“3D”) graphics data in order to generate two-dimensional (“2D”) image data for display on a display device such as a liquid crystal display (“LCD”) device. In at least one embodiment, PPU 3400 is utilized to perform computations such as linear

algebra operations and machine-learning operations. FIG. 34 illustrates an example parallel processor for illustrative purposes only and should be construed as a non-limiting example of processor architectures contemplated within scope of this disclosure and that any suitable processor may be employed to supplement and/or substitute for same.

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

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

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

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

In at least one embodiment, I/O unit 3406 decodes packets received via system bus 3402. In at least one embodiment, at least some packets represent commands configured to

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

In at least one embodiment, a program executed by host processor encodes a command stream in a buffer that provides workloads to PPU 3400 for processing. In at least one embodiment, a workload comprises instructions and data to be processed by those instructions. In at least one embodiment, a buffer is a region in a memory that is accessible (e.g., read/write) by both a host processor and PPU 3400 a host interface unit may be configured to access that buffer in a system memory connected to system bus 3402 via memory requests transmitted over system bus 3402 by I/O unit 3406. In at least one embodiment, a host processor writes a command stream to a buffer and then transmits a pointer to a start of a command stream to PPU 3400 such that front-end unit 3410 receives pointers to one or more command streams and manages one or more command streams, reading commands from command streams and forwarding commands to various units of PPU 3400.

In at least one embodiment, front-end unit 3410 is coupled to scheduler unit 3412 that configures various GPCs 3418 to process tasks defined by one or more command streams. In at least one embodiment, scheduler unit 3412 is configured to track state information related to various tasks managed by scheduler unit 3412 where state information may indicate which of GPCs 3418 a task is assigned to, whether task is active or inactive, a priority level associated with task, and so forth. In at least one embodiment, scheduler unit 3412 manages execution of a plurality of tasks on one or more of GPCs 3418.

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

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

one embodiment, one or more other units of PPU 3400 may also be connected to XBar 3420 via hub 3416.

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

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 35 illustrates a general processing cluster (“GPC”) 3500, according to at least one embodiment. In at least one embodiment, GPC 3500 is GPC 3418 of FIG. 34. In at least one embodiment, each GPC 3500 includes, without limita-

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

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

In at least one embodiment, preROP unit 3504 is configured, in at least one embodiment, to route data generated by raster engine 3508 and DPCs 3506 to a Raster Operations (“ROP”) unit in partition unit 3422, described in more detail above in conjunction with FIG. 34. In at least one embodiment, preROP unit 3504 is configured to perform optimizations for color blending, organize pixel data, perform address translations, and more. In at least one embodiment, raster engine 3508 includes, without limitation, a number of fixed function hardware units configured to perform various raster operations, in at least one embodiment, and raster engine 3508 includes, without limitation, a setup engine, a coarse raster engine, a culling engine, a clipping engine, a fine raster engine, a tile coalescing engine, and any suitable combination thereof. In at least one embodiment, setup engine receives transformed vertices and generates plane equations associated with geometric primitive defined by vertices; plane equations are transmitted to a coarse raster engine to generate coverage information (e.g., an x, y coverage mask for a tile) for primitive; output of a coarse raster engine is transmitted to a culling engine where fragments associated with a primitive that fail a z-test are culled, and transmitted to a clipping engine where fragments lying outside a viewing frustum are clipped. In at least one embodiment, fragments that survive clipping and culling are passed to a fine raster engine to generate attributes for pixel fragments based on plane equations generated by a setup engine. In at least one embodiment, an output of raster engine 3508 comprises fragments to be processed by any suitable entity, such as by a fragment shader implemented within DPC 3506.

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

butes associated with a vertex from memory; in contrast, packets associated with a shader program may be transmitted to SM 3514.

In at least one embodiment, SM 3514 comprises, without limitation, a programmable streaming processor that is configured to process tasks represented by a number of threads. In at least one embodiment, SM 3514 is multi-threaded and configured to execute a plurality of threads (e.g., 32 threads) from a particular group of threads concurrently and implements a Single-Instruction, Multiple-Data (“SIMD”) architecture where each thread in a group of threads (e.g., a warp) is configured to process a different set of data based on same set of instructions. In at least one embodiment, all threads in group of threads execute a common set of instructions. In at least one embodiment, SM 3514 implements a Single-Instruction, Multiple Thread (“SIMT”) architecture wherein each thread in a group of threads is configured to process a different set of data based on that common set of instructions, but where individual threads in a group of threads are allowed to diverge during execution. In at least one embodiment, a program counter, call stack, and execution state is maintained for each warp, enabling concurrency between warps and serial execution within warps when threads within a warp diverge. In another embodiment, a program counter, call stack, and execution state is maintained for each individual thread, enabling equal concurrency between all threads, within and between warps. In at least one embodiment, execution state is maintained for each individual thread and threads executing common instructions may be converged and executed in parallel for better efficiency. At least one embodiment of SM 3514 is described in more detail herein.

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

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

In various embodiments, inference and/or training logic are used to perform inferencing and/or training operations associated determining an event will occur and/or detecting the occurrence of the event. In addition, the computing resources described, in various embodiments, is used to perform various operations such as generating assets and modify broadcasts and/or streams.

FIG. 36 illustrates a memory partition unit 3600 of a parallel processing unit (“PPU”), in accordance with at least one embodiment. In at least one embodiment, memory partition unit 3600 includes, without limitation, a Raster Operations (“ROP”) unit 3602, a level two (“L2”) cache

3604, a memory interface 3606, and any suitable combination thereof. In at least one embodiment, memory interface 3606 is coupled to memory. In at least one embodiment, memory interface 3606 may implement 32, 64, 128, 1024-bit data buses, or like, for high-speed data transfer. In at least one embodiment, PPU incorporates U memory interfaces 3606 where U is a positive integer, with one memory interface 3606 per pair of partition units 3600, where each pair of partition units 3600 is connected to a corresponding memory device. For example, in at least one embodiment, PPU may be connected to up to Y memory devices, such as high bandwidth memory stacks or graphics double-data-rate, version 5, synchronous dynamic random access memory (“GDDR5 SDRAM”).

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

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

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

Data from memory 3404 of FIG. 34 or other system memory is fetched by memory partition unit 3600 and stored in L2 cache 3604, which is located on-chip and is shared between various GPCs, in accordance with at least one embodiment. Each memory partition unit 3600, in at least one embodiment, includes, without limitation, at least a portion of L2 cache associated with a corresponding memory device. In at least one embodiment, lower level caches are implemented in various units within GPCs. In at least one embodiment, each of SMs 3514 in FIG. 35 may

119

implement a Level 1 (“L1”) cache wherein that L1 cache is private memory that is dedicated to a particular SM **3514** and data from L2 cache **3604** is fetched and stored in each L1 cache for processing in functional units of SMs **3514**. In at least one embodiment, L2 cache **3604** is coupled to memory interface **3606** and XBar **3420** shown in FIG. **34**.

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

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

In at least one embodiment, a work distribution unit dispatches tasks for execution on general processing clusters (“GPCs”) of parallel processing units (“PPUs”) and each task is allocated to a particular Data Processing Cluster (“DPC”) within a GPC and, if a task is associated with a shader program, that task is allocated to one of SMs **3700**. In at least one embodiment, scheduler unit **3704** receives tasks from a work distribution unit and manages instruction scheduling for one or more thread blocks assigned to SM **3700**. In at least one embodiment, scheduler unit **3704** schedules thread blocks for execution as warps of parallel threads, wherein each thread block is allocated at least one warp. In at least one embodiment, each warp executes threads. In at least one embodiment, scheduler unit **3704** manages a plurality of different thread blocks, allocating warps to different thread blocks and then dispatching instructions from plurality of different cooperative groups to various functional units (e.g., processing cores **3710**, SFUs **3712**, and LSUs **3714**) during each clock cycle.

In at least one embodiment, Cooperative Groups may refer to a programming model for organizing groups of communicating threads that allows developers to express granularity at which threads are communicating, enabling expression of richer, more efficient parallel decompositions. In at least one embodiment, cooperative launch APIs support synchronization amongst thread blocks for execution of parallel algorithms. In at least one embodiment, applications of conventional programming models provide a single, simple construct for synchronizing cooperating threads: a barrier across all threads of a thread block (e.g., `syncthreads()` function). However, in at least one embodiment, programmers may define groups of threads at smaller than thread

120

block granularities and synchronize within defined groups to enable greater performance, design flexibility, and software reuse in form of collective group-wide function interfaces. In at least one embodiment, Cooperative Groups enables programmers to define groups of threads explicitly at sub-block (i.e., as small as a single thread) and multi-block granularities, and to perform collective operations such as synchronization on threads in a cooperative group. In at least one embodiment, that programming model supports clean composition across software boundaries, so that libraries and utility functions can synchronize safely within their local context without having to make assumptions about convergence. In at least one embodiment, Cooperative Groups primitives enable new patterns of cooperative parallelism, including, without limitation, producer-consumer parallelism, opportunistic parallelism, and global synchronization across an entire grid of thread blocks.

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

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

Tensor cores are configured to perform matrix operations in accordance with at least one embodiment. In at least one embodiment, one or more tensor cores are included in processing cores **3710**. In at least one embodiment, tensor cores are configured to perform deep learning matrix arithmetic, such as convolution operations for neural network training and inferencing. In at least one embodiment, each tensor core operates on a 4×4 matrix and performs a matrix multiply and accumulate operation, D=AxB+C, where A, B, C, and D are 4×4 matrices.

In at least one embodiment, matrix multiply inputs A and B are 16-bit floating point matrices and accumulation matrices C and D are 16-bit floating point or 32-bit floating point matrices. In at least one embodiment, tensor cores operate on 16-bit floating point input data with 32-bit floating point accumulation. In at least one embodiment, 16-bit floating

## 121

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  $4 \times 4 \times 4$  matrix multiply. Tensor cores are used to perform much larger two-dimensional or higher dimensional matrix operations, built up from these smaller elements, in at least one embodiment. In at least one embodiment, an API, such as a CUDA 9 C++ API, exposes specialized matrix load, matrix multiply and accumulate, and matrix store operations to efficiently use tensor cores from a CUDA-C++ program. In at least one embodiment, at a CUDA level, a warp-level interface assumes  $16 \times 16$  size matrices spanning all 32 threads of warp.

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

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

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

Combining data cache and shared memory functionality into a single memory block provides improved performance for both types of memory accesses, in at least one embodiment. In at least one embodiment, capacity is used or is usable as a cache by programs that do not use shared memory, such as if shared memory is configured to use half of a capacity, and texture and load/store operations can use remaining capacity. Integration within shared memory/L1 cache 3718 enables shared memory/L1 cache 3718 to function as a high-throughput conduit for streaming data while simultaneously providing high-bandwidth and low-latency access to frequently reused data, in accordance with at least

## 122

one embodiment. In at least one embodiment, when configured for general purpose parallel computation, a simpler configuration can be used compared with graphics processing. In at least one embodiment, fixed function graphics processing units are bypassed, creating a much simpler programming model. In a general purpose parallel computation configuration, a work distribution unit assigns and distributes blocks of threads directly to DPCs, in at least one embodiment. In at least one embodiment, threads in a block execute a common program, using a unique thread ID in calculation to ensure each thread generates unique results, using SM 3700 to execute program and perform calculations, shared memory/L1 cache 3718 to communicate between threads, and LSU 3714 to read and write global memory through shared memory/L1 cache 3718 and memory partition unit. In at least one embodiment, when configured for general purpose parallel computation, SM 3700 writes commands that scheduler unit 3704 can use to launch new work on DPCs.

In at least one embodiment, a PPU is included in or coupled to a desktop computer, a laptop computer, a tablet computer, servers, supercomputers, a smart-phone (e.g., a wireless, hand-held device), personal digital assistant ("PDA"), a digital camera, a vehicle, a head mounted display, a hand-held electronic device, and more. In at least one embodiment, a PPU is embodied on a single semiconductor substrate. In at least one embodiment, a PPU is included in a system-on-a-chip ("SoC") along with one or more other devices such as additional PPUs, memory, a reduced instruction set computer ("RISC") CPU, a memory management unit ("MMU"), a digital-to-analog converter ("DAC"), and like.

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

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

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

123

ogy, meteorology, intelligent area or object tracking and monitoring, sensor data processing (e.g., RADAR, SONAR, LIDAR, etc.), and/or genomics and gene sequencing.

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

In at least one embodiment, process 3800 may be executed within a training system 3804 and/or a deployment system 3806. In at least one embodiment, training system 3804 may be used to perform training, deployment, and implementation of machine learning models (e.g., neural networks, object detection algorithms, computer vision algorithms, etc.) for use in deployment system 3806. In at least one embodiment, deployment system 3806 may be configured to offload processing and compute resources among a distributed computing environment to reduce infrastructure requirements at facility 3802. In at least one embodiment, deployment system 3806 may provide a streamlined platform for selecting, customizing, and implementing virtual instruments for use with imaging devices (e.g., MRI, CT Scan, X-Ray, Ultrasound, etc.) or sequencing devices at facility 3802. In at least one embodiment, virtual instruments may include software-defined applications for performing one or more processing operations with respect to imaging data generated by imaging devices, sequencing devices, radiology devices, and/or other device types. In at least one embodiment, one or more applications in a pipeline may use or call upon services (e.g., inference, visualization, compute, AI, etc.) of deployment system 3806 during execution of applications.

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

In at least one embodiment, a model registry 3824 may be backed by object storage that may support versioning and object metadata. In at least one embodiment, object storage may be accessible through, for example, a cloud storage (e.g., a cloud 3926 of FIG. 39) compatible application programming interface (API) from within a cloud platform. In at least one embodiment, machine learning models within model registry 3824 may be uploaded, listed, modified, or deleted by developers or partners of a system interacting with an API. In at least one embodiment, an API may provide access to methods that allow users with appropriate

124

credentials to associate models with applications, such that models may be executed as part of execution of containerized instantiations of applications.

In at least one embodiment, a training pipeline 3904 (FIG. 39) may include a scenario where facility 3802 is training their own machine learning model, or has an existing machine learning model that needs to be optimized or updated. In at least one embodiment, imaging data 3808 generated by imaging device(s), sequencing devices, and/or other device types may be received. In at least one embodiment, once imaging data 3808 is received, AI-assisted annotation 3810 may be used to aid in generating annotations corresponding to imaging data 3808 to be used as ground truth data for a machine learning model. In at least one embodiment, AI-assisted annotation 3810 may include one or more machine learning models (e.g., convolutional neural networks (CNNs)) that may be trained to generate annotations corresponding to certain types of imaging data 3808 (e.g., from certain devices) and/or certain types of anomalies in imaging data 3808. In at least one embodiment, AI-assisted annotations 3810 may then be used directly, or may be adjusted or fine-tuned using an annotation tool (e.g., by a researcher, a clinician, a doctor, a scientist, etc.) to generate ground truth data. In at least one embodiment, in some examples, labeled clinic data 3812 (e.g., annotations provided by a clinician, doctor, scientist, technician, etc.) may be used as ground truth data for training a machine learning model. In at least one embodiment, AI-assisted annotations 3810, labeled clinic data 3812, or a combination thereof may be used as ground truth data for training a machine learning model. In at least one embodiment, a trained machine learning model may be referred to as an output model 3816, and may be used by deployment system 3806, as described herein.

In at least one embodiment, training pipeline 3904 (FIG. 39) may include a scenario where facility 3802 needs a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system 3806, but facility 3802 may not currently have such a machine learning model (or may not have a model that is optimized, efficient, or effective for such purposes). In at least one embodiment, an existing machine learning model may be selected from model registry 3824. In at least one embodiment, model registry 3824 may include machine learning models trained to perform a variety of different inference tasks on imaging data. In at least one embodiment, machine learning models in model registry 3824 may have been trained on imaging data from different facilities than facility 3802 (e.g., facilities remotely located). In at least one embodiment, machine learning models may have been trained on imaging data from one location, two locations, or any number of locations. In at least one embodiment, when being trained on imaging data from a specific location, training may take place at that location, or at least in a manner that protects confidentiality of imaging data or restricts imaging data from being transferred off-premises (e.g., to comply with HIPAA regulations, privacy regulations, etc.). In at least one embodiment, once a model is trained—or partially trained—at one location, a machine learning model may be added to model registry 3824. In at least one embodiment, a machine learning model may then be retrained, or updated, at any number of other facilities, and a retrained or updated model may be made available in model registry 3824. In at least one embodiment, a machine learning model may then be selected from model registry 3824—and referred to as output model 3816—and may be

125

used in deployment system 3806 to perform one or more processing tasks for one or more applications of a deployment system.

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

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

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

126

display at facility 3802). In at least one embodiment, a combination of containers within software 3818 (e.g., that make up a pipeline) may be referred to as a virtual instrument (as described in more detail herein), and a virtual instrument may leverage services 3820 and hardware 3822 to execute some or all processing tasks of applications instantiated in containers.

In at least one embodiment, a data processing pipeline may receive input data (e.g., imaging data 3808) in a DICOM, RIS, CIS, REST compliant, RPC, raw, and/or other format in response to an inference request (e.g., a request from a user of deployment system 3806, such as a clinician, a doctor, a radiologist, etc.). In at least one embodiment, input data may be representative of one or more images, video, and/or other data representations generated by one or more imaging devices, sequencing devices, radiology devices, genomics devices, and/or other device types. In at least one embodiment, data may undergo pre-processing as part of data processing pipeline to prepare data for processing by one or more applications. In at least one embodiment, post-processing may be performed on an output of one or more inferencing tasks or other processing tasks of a pipeline to prepare an output data for a next application and/or to prepare output data for transmission and/or use by a user (e.g., as a response to an inference request). In at least one embodiment, inferencing tasks may be performed by one or more machine learning models, such as trained or deployed neural networks, which may include output models 3816 of training system 3804.

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

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

provider, etc.) to perform one or more processing tasks with respect to data at a facility (e.g., a second facility) of a user.

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

In at least one embodiment, to aid in processing or execution of applications or containers in pipelines, services 3820 may be leveraged. In at least one embodiment, services 3820 may include compute services, artificial intelligence (AI) services, visualization services, and/or other service types. In at least one embodiment, services 3820 may provide functionality that is common to one or more applications in software 3818, so functionality may be abstracted to a service that may be called upon or leveraged by applications. In at least one embodiment, functionality provided by services 3820 may run dynamically and more efficiently, while also scaling well by allowing applications to process data in parallel (e.g., using a parallel computing platform 3930 (FIG. 39)). In at least one embodiment, rather than each application that shares a same functionality offered by a service 3820 being required to have a respective instance of service 3820, service 3820 may be shared between and among various applications. In at least one embodiment, services may include an inference server or engine that may be used for executing detection or segmentation tasks, as non-limiting examples. In at least one embodiment, a model training service may be included that may provide machine learning model training and/or retraining capabilities. In at least one embodiment, a data augmentation service may further be included that may provide GPU accelerated data (e.g., DICOM, RIS, CIS, REST compliant, RPC, raw, etc.) extraction, resizing, scaling, and/or other augmentation. In at least one embodiment, a visualization service may be used that may add image rendering effects—such as ray-tracing, rasterization, denoising, sharpening, etc.—to add realism to two-dimensional (2D) and/or three-dimensional (3D) models. In at least one

embodiment, virtual instrument services may be included that provide for beam-forming, segmentation, inferencing, imaging, and/or support for other applications within pipelines of virtual instruments.

- 5     In at least one embodiment, where a service 3820 includes an AI service (e.g., an inference service), one or more machine learning models associated with an application for anomaly detection (e.g., tumors, growth abnormalities, scarring, etc.) may be executed by calling upon (e.g., as an API call) an inference service (e.g., an inference server) to execute machine learning model(s), or processing thereof, as part of application execution. In at least one embodiment, where another application includes one or more machine learning models for segmentation tasks, an application may 10 call upon an inference service to execute machine learning models for performing one or more of processing operations associated with segmentation tasks. In at least one embodiment, software 3818 implementing advanced processing and inferencing pipeline that includes segmentation application 15 and anomaly detection application may be streamlined because each application may call upon a same inference 20 service to perform one or more inferencing tasks.

In at least one embodiment, hardware 3822 may include GPUs, CPUs, graphics cards, an AI/deep learning system 25 (e.g., an AI supercomputer, such as NVIDIA's DGX supercomputer system), a cloud platform, or a combination thereof. In at least one embodiment, different types of hardware 3822 may be used to provide efficient, purpose-built support for software 3818 and services 3820 in deployment system 3806. In at least one embodiment, use of GPU 30 processing may be implemented for processing locally (e.g., at facility 3802), within an AI/deep learning system, in a cloud system, and/or in other processing components of deployment system 3806 to improve efficiency, accuracy, 35 and efficacy of image processing, image reconstruction, segmentation, MRI exams, stroke or heart attack detection (e.g., in real-time), image quality in rendering, etc. In at least one embodiment, a facility may include imaging devices, genomics devices, sequencing devices, and/or other device 40 types on-premises that may leverage GPUs to generate imaging data representative of a subject's anatomy.

In at least one embodiment, software 3818 and/or services 3820 may be optimized for GPU processing with respect to deep learning, machine learning, and/or high-performance 45 computing, as non-limiting examples. In at least one embodiment, at least some of computing environment of deployment system 3806 and/or training system 3804 may be executed in a datacenter one or more supercomputers or high performance computing systems, with GPU optimized 50 software (e.g., hardware and software combination of NVIDIA's DGX system). In at least one embodiment, datacenters may be compliant with provisions of HIPAA, such that receipt, processing, and transmission of imaging data and/or other patient data is securely handled with 55 respect to privacy of patient data. In at least one embodiment, hardware 3822 may include any number of GPUs that may be called upon to perform processing of data in parallel, as described herein. In at least one embodiment, cloud platform may further include GPU processing for GPU- 60 optimized execution of deep learning tasks, machine learning tasks, or other computing tasks. In at least one embodiment, cloud platform (e.g., NVIDIA's NGC) may be executed using an AI/deep learning supercomputer(s) and/or GPU-optimized software (e.g., as provided on NVIDIA's 65 DGX systems) as a hardware abstraction and scaling platform. In at least one embodiment, cloud platform may integrate an application container clustering system or

129

orchestration system (e.g., KUBERNETES) on multiple GPUs to enable seamless scaling and load balancing.

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

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

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

In at least one embodiment, training system 3804 may execute training pipelines 3904, similar to those described herein with respect to FIG. 38. In at least one embodiment, where one or more machine learning models are to be used in deployment pipelines 3910 by deployment system 3806, training pipelines 3904 may be used to train or retrain one or more (e.g., pre-trained) models, and/or implement one or more of pre-trained models 3906 (e.g., without a need for retraining or updating). In at least one embodiment, as a result of training pipelines 3904, output model(s) 3816 may be generated. In at least one embodiment, training pipelines 3904 may include any number of processing steps, such as but not limited to imaging data (or other input data) conversion or adaption (e.g., using DICOM adapter 3902A to convert DICOM images to another format suitable for processing by respective machine learning models, such as Neuroimaging Informatics Technology Initiative (NIfTI) format), AI-assisted annotation 3810, labeling or annotating of imaging data 3808 to generate labeled clinic data 3812, model selection from a model registry, model training 3814, training, retraining, or updating models, and/or other processing steps. In at least one embodiment, for different

130

machine learning models used by deployment system 3806, different training pipelines 3904 may be used. In at least one embodiment, training pipeline 3904 similar to a first example described with respect to FIG. 38 may be used for a first machine learning model, training pipeline 3904 similar to a second example described with respect to FIG. 38 may be used for a second machine learning model, and training pipeline 3904 similar to a third example described with respect to FIG. 38 may be used for a third machine learning model. In at least one embodiment, any combination of tasks within training system 3804 may be used depending on what is required for each respective machine learning model. In at least one embodiment, one or more of machine learning models may already be trained and ready for deployment so machine learning models may not undergo any processing by training system 3804, and may be implemented by deployment system 3806.

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

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

## 131

or another data type adapter such as RIS, CIS, REST compliant, RPC, raw, etc.) to perform operations, such as training machine learning models, deploying machine learning models, image processing, inferencing, and/or other operations.

In at least one embodiment, a software layer may be implemented as a secure, encrypted, and/or authenticated API through which applications or containers may be invoked (e.g., called) from an external environment(s) (e.g., facility 3802). In at least one embodiment, applications may then call or execute one or more services 3820 for performing compute, AI, or visualization tasks associated with respective applications, and software 3818 and/or services 3820 may leverage hardware 3822 to perform processing tasks in an effective and efficient manner.

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

In at least one embodiment, applications available for deployment pipelines 3910 may include any application that may be used for performing processing tasks on imaging data or other data from devices. In at least one embodiment, different applications may be responsible for image enhancement, segmentation, reconstruction, anomaly detection, object detection, feature detection, treatment planning, dosimetry, beam planning (or other radiation treatment procedures), and/or other analysis, image processing, or inferencing tasks. In at least one embodiment, deployment system 3806 may define constructs for each of applications, such that users of deployment system 3806 (e.g., medical facilities, labs, clinics, etc.) may understand constructs and adapt applications for implementation within their respective facility. In at least one embodiment, an application for image reconstruction may be selected for inclusion in deployment pipeline 3910, but data type generated by an imaging device may be different from a data type used within an application. In at least one embodiment, DICOM adapter 3902B (and/or a DICOM reader) or another data type adapter or reader (e.g., RIS, CIS, REST compliant, RPC, raw, etc.) may be used within deployment pipeline 3910 to convert data to a form useable by an application within deployment system 3806. In at least one embodiment, access to DICOM, RIS, CIS, REST compliant, RPC, raw, and/or other data type libraries may be accumulated and pre-processed, including decoding, extracting, and/or performing any convolutions, color corrections, sharpness, gamma, and/or other augmentations to data. In at least one embodiment, DICOM, RIS, CIS, REST compliant, RPC, and/or raw data may be unordered and a pre-pass may be executed to organize or sort collected data. In at least one embodiment, because various applications may share com-

## 132

mon image operations, in some embodiments, a data augmentation library (e.g., as one of services 3820) may be used to accelerate these operations. In at least one embodiment, to avoid bottlenecks of conventional processing approaches 5 that rely on CPU processing, parallel computing platform 3930 may be used for GPU acceleration of these processing tasks.

In at least one embodiment, an image reconstruction application may include a processing task that includes use 10 of a machine learning model. In at least one embodiment, a user may desire to use their own machine learning model, or to select a machine learning model from model registry 3824. In at least one embodiment, a user may implement their own machine learning model or select a machine 15 learning model for inclusion in an application for performing a processing task. In at least one embodiment, applications may be selectable and customizable, and by defining constructs of applications, deployment and implementation of applications for a particular user are presented as a more seamless user experience. In at least one embodiment, by leveraging other features of system 3900—such as services 3820 and hardware 3822—deployment pipelines 3910 may be even more user friendly, provide for easier integration, and produce more accurate, efficient, and timely results.

In at least one embodiment, deployment system 3806 may 20 include a user interface 3914 (e.g., a graphical user interface, a web interface, etc.) that may be used to select applications for inclusion in deployment pipeline(s) 3910, arrange applications, modify or change applications or parameters or 25 constructs thereof, use and interact with deployment pipeline(s) 3910 during set-up and/or deployment, and/or to otherwise interact with deployment system 3806. In at least one embodiment, although not illustrated with respect to training system 3804, user interface 3914 (or a different user 30 interface) may be used for selecting models for use in deployment system 3806, for selecting models for training, or retraining, in training system 3804, and/or for otherwise interacting with training system 3804.

In at least one embodiment, pipeline manager 3912 may 35 be used, in addition to an application orchestration system 3928, to manage interaction between applications or containers of deployment pipeline(s) 3910 and services 3820 and/or hardware 3822. In at least one embodiment, pipeline manager 3912 may be configured to facilitate interactions 40 from application to application, from application to service 3820, and/or from application or service to hardware 3822. In at least one embodiment, although illustrated as included in software 3818, this is not intended to be limiting, and in some examples (e.g., as illustrated in FIG. 40) pipeline manager 3912 may be included in services 3820. In at least one embodiment, application orchestration system 3928 (e.g., Kubernetes, DOCKER, etc.) may include a container 45 orchestration system that may group applications into containers as logical units for coordination, management, scaling, and deployment. In at least one embodiment, by associating applications from deployment pipeline(s) 3910 (e.g., a reconstruction application, a segmentation application, etc.) with individual containers, each application may 50 execute in a self-contained environment (e.g., at a kernel level) to increase speed and efficiency.

In at least one embodiment, each application and/or container (or image thereof) may be individually developed, modified, and deployed (e.g., a first user or developer may 55 develop, modify, and deploy a first application and a second user or developer may develop, modify, and deploy a second application separate from a first user or developer), which may allow for focus on, and attention to, a task of a single

133

application and/or container(s) without being hindered by tasks of another application(s) or container(s). In at least one embodiment, communication, and cooperation between different containers or applications may be aided by pipeline manager **3912** and application orchestration system **3928**. In at least one embodiment, so long as an expected input and/or output of each container or application is known by a system (e.g., based on constructs of applications or containers), application orchestration system **3928** and/or pipeline manager **3912** may facilitate communication among and between, and sharing of resources among and between, each of applications or containers. In at least one embodiment, because one or more of applications or containers in deployment pipeline(s) **3910** may share same services and resources, application orchestration system **3928** may orchestrate, load balance, and determine sharing of services or resources between and among various applications or containers. In at least one embodiment, a scheduler may be used to track resource requirements of applications or containers, current usage or planned usage of these resources, and resource availability. In at least one embodiment, a scheduler may thus allocate resources to different applications and distribute resources between and among applications in view of requirements and availability of a system. In some examples, a scheduler (and/or other component of application orchestration system **3928**) may determine resource availability and distribution based on constraints imposed on a system (e.g., user constraints), such as quality of service (QoS), urgency of need for data outputs (e.g., to determine whether to execute real-time processing or delayed processing), etc.

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

134

times, etc.). In at least one embodiment, as data is used to generate new data as a result of processing, this information of a new location of data may be stored and shared between various applications. In at least one embodiment, location of data and a location of updated or modified data may be part of a definition of how a payload is understood within containers.

In at least one embodiment, AI services **3918** may be leveraged to perform inferencing services for executing machine learning model(s) associated with applications (e.g., tasked with performing one or more processing tasks of an application). In at least one embodiment, AI services **3918** may leverage AI system **3924** to execute machine learning model(s) (e.g., neural networks, such as CNNs) for segmentation, reconstruction, object detection, feature detection, classification, and/or other inferencing tasks. In at least one embodiment, applications of deployment pipeline(s) **3910** may use one or more of output models **3816** from training system **3804** and/or other models of applications to perform inference on imaging data (e.g., DICOM data, RIS data, CIS data, REST compliant data, RPC data, raw data, etc.). In at least one embodiment, two or more examples of inferencing using application orchestration system **3928** (e.g., a scheduler) may be available. In at least one embodiment, a first category may include a high priority/low latency path that may achieve higher service level agreements, such as for performing inference on urgent requests during an emergency, or for a radiologist during diagnosis. In at least one embodiment, a second category may include a standard priority path that may be used for requests that may be non-urgent or where analysis may be performed at a later time. In at least one embodiment, application orchestration system **3928** may distribute resources (e.g., services **3820** and/or hardware **3822**) based on priority paths for different inferencing tasks of AI services **3918**.

In at least one embodiment, shared storage may be mounted to AI services **3918** within system **3900**. In at least one embodiment, shared storage may operate as a cache (or other storage device type) and may be used to process inference requests from applications. In at least one embodiment, when an inference request is submitted, a request may be received by a set of API instances of deployment system **3806**, and one or more instances may be selected (e.g., for best fit, for load balancing, etc.) to process a request. In at least one embodiment, to process a request, a request may be entered into a database, a machine learning model may be located from model registry **3824** if not already in a cache, a validation step may ensure appropriate machine learning model is loaded into a cache (e.g., shared storage), and/or a copy of a model may be saved to a cache. In at least one embodiment, a scheduler (e.g., of pipeline manager **3912**) may be used to launch an application that is referenced in a request if an application is not already running or if there are not enough instances of an application. In at least one embodiment, if an inference server is not already launched to execute a model, an inference server may be launched. In at least one embodiment, any number of inference servers may be launched per model. In at least one embodiment, in a pull model, in which inference servers are clustered, models may be cached whenever load balancing is advantageous. In at least one embodiment, inference servers may be statically loaded in corresponding, distributed servers.

In at least one embodiment, inferencing may be performed using an inference server that runs in a container. In at least one embodiment, an instance of an inference server may be associated with a model (and optionally a plurality of versions of a model). In at least one embodiment, if an

## 135

instance of an inference server does not exist when a request to perform inference on a model is received, a new instance may be loaded. In at least one embodiment, when starting an inference server, a model may be passed to an inference server such that a same container may be used to serve different models so long as inference server is running as a different instance.

In at least one embodiment, during application execution, an inference request for a given application may be received, and a container (e.g., hosting an instance of an inference server) may be loaded (if not already), and a start procedure may be called. In at least one embodiment, pre-processing logic in a container may load, decode, and/or perform any additional pre-processing on incoming data (e.g., using a CPU(s) and/or GPU(s)). In at least one embodiment, once data is prepared for inference, a container may perform inference as necessary on data. In at least one embodiment, this may include a single inference call on one image (e.g., a hand X-ray), or may require inference on hundreds of images (e.g., a chest CT). In at least one embodiment, an application may summarize results before completing, which may include, without limitation, a single confidence score, pixel level-segmentation, voxel-level segmentation, generating a visualization, or generating text to summarize findings. In at least one embodiment, different models or applications may be assigned different priorities. For example, some models may have a real-time (TAT less than one minute) priority while others may have lower priority (e.g., TAT less than 10 minutes). In at least one embodiment, model execution times may be measured from requesting institution or entity and may include partner network traversal time, as well as execution on an inference service.

In at least one embodiment, transfer of requests between services 3820 and inference applications may be hidden behind a software development kit (SDK), and robust transport may be provided through a queue. In at least one embodiment, a request will be placed in a queue via an API for an individual application/tenant ID combination and an SDK will pull a request from a queue and give a request to an application. In at least one embodiment, a name of a queue may be provided in an environment from where an SDK will pick it up. In at least one embodiment, asynchronous communication through a queue may be useful as it may allow any instance of an application to pick up work as it becomes available. In at least one embodiment, results may be transferred back through a queue, to ensure no data is lost. In at least one embodiment, queues may also provide an ability to segment work, as highest priority work may go to a queue with most instances of an application connected to it, while lowest priority work may go to a queue with a single instance connected to it that processes tasks in an order received. In at least one embodiment, an application may run on a GPU-accelerated instance generated in cloud 3926, and an inference service may perform inferencing on a GPU.

In at least one embodiment, visualization services 3920 may be leveraged to generate visualizations for viewing outputs of applications and/or deployment pipeline(s) 3910. In at least one embodiment, GPUs 3922 may be leveraged by visualization services 3920 to generate visualizations. In at least one embodiment, rendering effects, such as ray-tracing, may be implemented by visualization services 3920 to generate higher quality visualizations. In at least one embodiment, visualizations may include, without limitation, 2D image renderings, 3D volume renderings, 3D volume reconstruction, 2D tomographic slices, virtual reality displays, augmented reality displays, etc. In at least one

## 136

embodiment, virtualized environments may be used to generate a virtual interactive display or environment (e.g., a virtual environment) for interaction by users of a system (e.g., doctors, nurses, radiologists, etc.). In at least one embodiment, visualization services 3920 may include an internal visualizer, cinematics, and/or other rendering or image processing capabilities or functionality (e.g., ray tracing, rasterization, internal optics, etc.).

In at least one embodiment, hardware 3822 may include 10 GPUs 3922, AI system 3924, cloud 3926, and/or any other hardware used for executing training system 3804 and/or deployment system 3806. In at least one embodiment, GPUs 3922 (e.g., NVIDIA's TESLA and/or QUADRO GPUs) may include any number of GPUs that may be used for 15 executing processing tasks of compute services 3916, AI services 3918, visualization services 3920, other services, and/or any of features or functionality of software 3818. For example, with respect to AI services 3918, GPUs 3922 may 20 be used to perform pre-processing on imaging data (or other data types used by machine learning models), post-processing on outputs of machine learning models, and/or to perform inferencing (e.g., to execute machine learning models). In at least one embodiment, cloud 3926, AI system 3924, and/or other components of system 3900 may use GPUs 3922. In at least one embodiment, cloud 3926 may include 25 a GPU-optimized platform for deep learning tasks. In at least one embodiment, AI system 3924 may use GPUs, and cloud 3926—or at least a portion tasked with deep learning or inferencing—may be executed using one or more AI systems 3924. As such, although hardware 3822 is illustrated as 30 discrete components, this is not intended to be limiting, and any components of hardware 3822 may be combined with, or leveraged by, any other components of hardware 3822.

In at least one embodiment, AI system 3924 may include 35 a purpose-built computing system (e.g., a super-computer or an HPC) configured for inferencing, deep learning, machine learning, and/or other artificial intelligence tasks. In at least one embodiment, AI system 3924 (e.g., NVIDIA's DGX) 40 may include GPU-optimized software (e.g., a software stack) that may be executed using a plurality of GPUs 3922, in addition to CPUs, RAM, storage, and/or other components, features, or functionality. In at least one embodiment, one or more AI systems 3924 may be implemented in cloud 45 3926 (e.g., in a data center) for performing some or all of AI-based processing tasks of system 3900.

In at least one embodiment, cloud 3926 may include 50 a GPU-accelerated infrastructure (e.g., NVIDIA's NGC) that may provide a GPU-optimized platform for executing processing tasks of system 3900. In at least one embodiment, cloud 3926 may include an AI system(s) 3924 for performing one or more of AI-based tasks of system 3900 (e.g., as a hardware abstraction and scaling platform). In at least one embodiment, cloud 3926 may integrate with application 55 orchestration system 3928 leveraging multiple GPUs to enable seamless scaling and load balancing between and among applications and services 3820. In at least one embodiment, cloud 3926 may task with executing at least some of services 3820 of system 3900, including compute services 3916, AI services 3918, and/or visualization services 3920, as described herein. In at least one embodiment, cloud 3926 may perform small and large batch inference (e.g., executing NVIDIA's TENSOR RT), provide an accelerated parallel computing API and platform 3930 (e.g., 60 NVIDIA's CUDA), execute application orchestration system 3928 (e.g., KUBERNETES), provide a graphics rendering API and platform (e.g., for ray-tracing, 2D graphics, 65

137

3D graphics, and/or other rendering techniques to produce higher quality cinematics), and/or may provide other functionality for system 3900.

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

FIG. 40 includes an example illustration of a deployment pipeline 3910A for processing imaging data, in accordance with at least one embodiment. In at least one embodiment, system 3900—and specifically deployment system 3806—may be used to customize, update, and/or integrate deployment pipeline(s) 3910A into one or more production environments. In at least one embodiment, deployment pipeline 3910A of FIG. 40 includes a non-limiting example of a deployment pipeline 3910A that may be custom defined by a particular user (or team of users) at a facility (e.g., at a hospital, clinic, lab, research environment, etc.). In at least one embodiment, to define deployment pipelines 3910A for a CT scanner 4002, a user may select—from a container registry, for example—one or more applications that perform specific functions or tasks with respect to imaging data generated by CT scanner 4002. In at least one embodiment, applications may be applied to deployment pipeline 3910A as containers that may leverage services 3820 and/or hardware 3822 of system 3900. In addition, deployment pipeline 3910A may include additional processing tasks or applications that may be implemented to prepare data for use by applications (e.g., DICOM adapter 3902B and DICOM reader 4006 may be used in deployment pipeline 3910A to prepare data for use by CT reconstruction 4008, organ segmentation 4010, etc.). In at least one embodiment, deployment pipeline 3910A may be customized or selected for consistent deployment, one time use, or for another frequency or interval. In at least one embodiment, a user may desire to have CT reconstruction 4008 and organ segmentation 4010 for several subjects over a specific interval, and thus may deploy pipeline 3910A for that period of time. In at least one embodiment, a user may select, for each request from system 3900, applications that a user wants to perform processing on that data for that request. In at least one embodiment, deployment pipeline 3910A may be adjusted at any interval and, because of adaptability and scalability of a container structure within system 3900, this may be a seamless process.

In at least one embodiment, deployment pipeline 3910A of FIG. 40 may include CT scanner 4002 generating imaging data of a patient or subject. In at least one embodiment, imaging data from CT scanner 4002 may be stored on a PACS server(s) 4004 associated with a facility housing CT scanner 4002. In at least one embodiment, PACS server(s) 4004 may include software and/or hardware components that may directly interface with imaging modalities (e.g., CT scanner 4002) at a facility. In at least one embodiment,

138

DICOM adapter 3902B may enable sending and receipt of DICOM objects using DICOM protocols. In at least one embodiment, DICOM adapter 3902B may aid in preparation or configuration of DICOM data from PACS server(s) 4004 for use by deployment pipeline 3910A. In at least one embodiment, once DICOM data is processed through DICOM adapter 3902B, pipeline manager 3912 may route data through to deployment pipeline 3910A. In at least one embodiment, DICOM reader 4006 may extract image files and any associated metadata from DICOM data (e.g., raw sinogram data, as illustrated in visualization 4016A). In at least one embodiment, working files that are extracted may be stored in a cache for faster processing by other applications in deployment pipeline 3910A. In at least one embodiment, once DICOM reader 4006 has finished extracting and/or storing data, a signal of completion may be communicated to pipeline manager 3912. In at least one embodiment, pipeline manager 3912 may then initiate or call upon one or more other applications or containers in deployment pipeline 3910A.

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

In at least one embodiment, once applications that process DICOM data and/or data extracted from DICOM data have completed processing, a signal may be generated for pipeline manager 3912. In at least one embodiment, pipeline manager 3912 may then execute DICOM writer 4012 to read results from a cache (or other storage device), package results into a DICOM format (e.g., as DICOM output 4014) for use by users at a facility who generated a request. In at least one embodiment, DICOM output 4014 may then be transmitted to DICOM adapter 3902B to prepare DICOM output 4014 for storage on PACS server(s) 4004 (e.g., for viewing by a DICOM viewer at a facility). In at least one embodiment, in response to a request for reconstruction and

139

segmentation, visualizations **4016B** and **4016C** may be generated and available to a user for diagnoses, research, and/or for other purposes.

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

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

In at least one embodiment, system **3900** may be instantiated or executed as one or more virtual instruments on-premise at a facility in, for example, a computing system deployed next to or otherwise in communication with a radiology machine, an imaging device, and/or another device type at a facility. In at least one embodiment, however, an on-premise installation may be instantiated or executed within a computing system of a device itself (e.g., a computing system integral to an imaging device), in a local datacenter (e.g., a datacenter on-premise), and/or in a cloud-environment (e.g., in cloud **3926**). In at least one embodiment, deployment system **3806**, operating as a virtual instrument, may be instantiated by a supercomputer or other HPC system in some examples. In at least one embodiment, on-premise installation may allow for high-bandwidth uses (via, for example, higher throughput local communication interfaces, such as RF over Ethernet) for real-time processing. In at least one embodiment, real-time or near real-time processing may be particularly useful where a virtual instrument supports an ultrasound device or other imaging modality where immediate visualizations are expected or required

140

for accurate diagnoses and analyses. In at least one embodiment, a cloud-computing architecture may be capable of dynamic bursting to a cloud computing service provider, or other compute cluster, when local demand exceeds on-premise capacity or capability. In at least one embodiment, a cloud architecture, when implemented, may be tuned for training neural networks or other machine learning models, as described herein with respect to training system **3804**. In at least one embodiment, with training pipelines in place, machine learning models may be continuously learn and improve as they process additional data from devices they support. In at least one embodiment, virtual instruments may be continually improved using additional data, new data, existing machine learning models, and/or new or updated machine learning models.

In at least one embodiment, a computing system may include some or all of hardware **3822** described herein, and hardware **3822** may be distributed in any of a number of ways including within a device, as part of a computing device coupled to and located proximate a device, in a local datacenter at a facility, and/or in cloud **3926**. In at least one embodiment, because deployment system **3806** and associated applications or containers are created in software (e.g., as discrete containerized instantiations of applications), behavior, operation, and configuration of virtual instruments, as well as outputs generated by virtual instruments, may be modified or customized as desired, without having to change or alter raw output of a device that a virtual instrument supports.

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

In at least one embodiment, process **4100** may include receipt of imaging data from an ultrasound device **4102**. In at least one embodiment, imaging data may be stored on PACS server(s) in a DICOM format (or other format, such as RIS, CIS, REST compliant, RPC, raw, etc.), and may be received by system **3900** for processing through deployment pipeline **3910** selected or customized as a virtual instrument (e.g., a virtual ultrasound) for ultrasound device **4102**. In at least one embodiment, imaging data may be received directly from an imaging device (e.g., ultrasound device **4102**) and processed by a virtual instrument. In at least one embodiment, a transducer or other signal converter communicatively coupled between an imaging device and a virtual instrument may convert signal data generated by an imaging device to image data that may be processed by a virtual instrument. In at least one embodiment, raw data and/or image data may be applied to DICOM reader **4006** to extract data for use by applications or containers of deployment pipeline **3910B**. In at least one embodiment, DICOM reader **4006** may leverage data augmentation library **4114** (e.g., NVIDIA's DALI) as a service **3820** (e.g., as one of compute service(s) **3916**) for extracting, resizing, rescaling, and/or otherwise preparing data for use by applications or containers.

In at least one embodiment, once data is prepared, a reconstruction **4106** application and/or container may be executed to reconstruct data from ultrasound device **4102**

141

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

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

FIG. 41B includes an example data flow diagram of a virtual instrument supporting a CT scanner, in accordance with at least one embodiment. In at least one embodiment, deployment pipeline 3910C may leverage one or more of services 3820 of system 3900. In at least one embodiment, deployment pipeline 3910C and services 3820 may leverage hardware 3822 of a system either locally or in cloud 3926. In at least one embodiment, although not illustrated, process 4120 may be facilitated by pipeline manager 3912, application orchestration system 3928, and/or parallel computing platform 3930.

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

In at least one embodiment, deployment pipeline 3910C may include a non-real-time pipeline for analyzing data generated by CT scanner 4122. In at least one embodiment, a second pipeline may include CT reconstruction 4008 application and/or container, a coarse detection AI 4128 application and/or container, a fine detection AI 4132 application and/or container (e.g., where certain results are detected by coarse detection AI 4128), a visualization 4130 application and/or container, and a DICOM writer 4012 (and/or other data type writer, such as RIS, CIS, REST compliant, RPC, raw, etc.) application and/or container. In at

142

least one embodiment, raw data generated by CT scanner 4122 may be passed through pipelines of deployment pipeline 3910C (instantiated as a virtual CT instrument) to generate results. In at least one embodiment, results from DICOM writer 4012 may be transmitted for display and/or may be stored on PACS server(s) 4004 for later retrieval, analysis, or display by a technician, practitioner, or other user.

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

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

In at least one embodiment, pre-trained models 3906 may be stored in a data store, or registry (e.g., model registry 3824 of FIG. 38). In at least one embodiment, pre-trained models 3906 may have been trained, at least in part, at one or more facilities other than a facility executing process 4200. In at least one embodiment, to protect privacy and rights of patients, subjects, or clients of different facilities, pre-trained models 3906 may have been trained, on-premise, using customer or patient data generated on-premise. In at least one embodiment, pre-trained models 3906 may be trained using cloud 3926 and/or other hardware 3822, but confidential, privacy protected patient data may not be transferred to, used by, or accessible to any components of cloud 3926 (or other off premise hardware). In at least one embodiment, where a pre-trained model 3906 is trained at using patient data from more than one facility, pre-trained model 3906 may have been individually trained for each facility prior to being trained on patient or customer data from another facility. In at least one embodiment, such as where a customer or patient data has been released of privacy concerns (e.g., by waiver, for experimental use, etc.), or where a customer or patient data is included in a public data set, a customer or patient data from any number of facilities may be used to train pre-trained model 3906 on-premise and/or off premise, such as in a datacenter or other cloud computing infrastructure.

In at least one embodiment, when selecting applications for use in deployment pipelines 3910, a user may also select machine learning models to be used for specific applica-

143

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

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

In at least one embodiment, AI-assisted annotation **3810** may be used in some examples to generate ground truth data. In at least one embodiment, AI-assisted annotation **3810** (e.g., implemented using an AI-assisted annotation SDK) may leverage machine learning models (e.g., neural networks) to generate suggested or predicted ground truth data for a customer dataset. In at least one embodiment, user **4210** may use annotation tools within a user interface (a graphical user interface (GUI)) on computing device **4208**.

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

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

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

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

144

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

Inference and/or training logic **915** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **915** are provided herein in conjunction with FIGS. 9A and/or 9B.

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

1. A method comprising: obtaining data associated with a streaming session; generating a prediction of an occurrence of an event in the streaming session at an interval of time; based at least in part on the prediction, causing the streaming session to be modified to generate a modified streaming session by at least applying an effect to one or more frames corresponding to the streaming session; and causing the modified streaming session to be broadcast.
2. The method of clause 1, wherein the method further comprises: capturing a set of inputs associated with the streaming session; generating a determination that the event has occurred based at least in part on the set of inputs; and causing the effect to no longer be applied to one or more frames corresponding to the modified streaming session.
3. The method of clause 1 or 2, wherein the set of inputs include at least one of an image, audio, or user input.
4. The method of any of clauses 1-3, wherein generating the determination that the event has occurred is performed using a model trained to detect objects in one or more image frames corresponding to the streaming session.
5. The method of any of clauses 1-4, wherein the method further comprises: determining an expiration of the interval of time; and causing the effect to no longer be applied to one or more other frames corresponding to the streaming session.
6. The method of any of clauses 1-5, wherein the streaming session corresponds to a gameplay session initiated by a server.
7. The method of any of clauses 1-6, wherein data associated with the streaming session includes at least one of: input data obtained from a remote client, one or more image frames

**145**

- corresponding to the streaming session, audio of the streaming session, and audio captured from the remote client.
8. The method of any of clauses 1-7, wherein the effect further comprises an overlay displayed in one or more image frames corresponding to the streaming session.
  9. The method of any of clauses 1-8, wherein the effect further comprises an audio cue included in one or more audio frames corresponding to the streaming session.
  10. The method of any of clauses 1-9, wherein generating the prediction further comprises analyzing inputs obtained from a controller device associated with a game session.
  11. The method of any of clauses 1-10, wherein the controller device include at least one of: a gaming console controller, a track pad, a mouse, a keyboard, or a motion controller.
  12. A machine-readable medium having stored thereon a set of instructions, which if performed by one or more processors, cause the one or more processors to: obtain data generated during a streaming session; predict an event at an interval of time in the future; and based at least in part on the event, cause the streaming session to be modified by at least applying an asset to the streaming session.
  13. The machine-readable medium of clause 12, wherein the set of instructions further include instructions which, if performed by the one or more processors, cause the one or more processors to: capture an image corresponding to the streaming session; generate a determination that the event has occurred based at least in part on the image; and cause a state of the asset to be modified.
  14. The machine-readable medium of clause 12 or 13, wherein the set of instructions further include instructions which, if performed by the one or more processors, cause the one or more processors to determine the interval of time based at least in part on an event type associated with the event and a type associated with the asset.
  15. The machine-readable medium of any of clauses 12-14, wherein the asset further comprises an effect applied to the streaming session.
  16. The machine-readable medium of any of clauses 12-15, wherein the effect adds a delay between successive frames of the streaming session.
  17. The machine-readable medium of any of clauses 12-16, wherein the set of instructions that cause the one or more processors to predict the event further include instructions which, if performed by the one or more processors, cause the one or more processors to predict the event based at least in part on audio or video captured at a host computer system associated with the streaming session.
  18. The machine-readable medium of any of clauses 12-17, wherein the set of instructions that cause the one or more processors to predict the event further include instructions which, if performed by the one or more processors, cause the one or more processors to predict the event based at least in part on inputs from a controller device logged at a host computer system associated with the streaming session.
  19. A system comprising: one or more processors; and memory storing instructions that, as a result of being executed by the one or more processors, cause the system to: cause a determination to be generated, the determination indicating that an event will occur within an interval of time based at least in part on data associated with a streaming session; and in response to the determination, cause an effect to be applied to the streaming session.
  20. The system of clause 19, wherein the memory further stores instructions that, as a result of being executed by the one or more processors, cause the system to: during the interval of time, obtain a set of images of the streaming

**146**

- session; and generate a second determination that the event occurred prior to an expiration of the interval of time.
21. The system of clause 19 or 20, wherein the memory further stores instructions that, as a result of being executed by the one or more processors, cause the system to, in response to an expiration of the interval of time, cause the effect to no longer be applied to the streaming session.
  22. The system of any of clauses 19-21, wherein the data associated with the streaming session includes keystroke data obtained from a remote client.
  23. The system of any of clauses 19-22, wherein the instructions that cause the determination to be generated further include instructions that, as a result of being executed by the one or more processors, cause the system to cause a machine learning model, executed by a server, to generate the determination.
  24. The system of any of clauses 19-23, wherein the machine learning model generates the determination based at least in part on images of a game session captured by the system.
  25. The system of any of clauses 19-24, wherein the machine learning model generates the determination based at least in part on information obtained from a game engine.
  26. The system of any of clauses 19-25, wherein the machine learning model generates the determination based at least in part on information obtained from a game associated with the streaming session.
  27. The system of any of clauses 19-26, wherein the determination further comprises information indicating a type associated with the event.
  28. The system of any of clauses 19-27, wherein the instructions that cause the effect to be applied to the streaming session further include instructions that, as a result of being executed by the one or more processors, cause the system to continue causing the effect to be applied as a result of determining the event occurred.
  - 30
  - 35
  - 40
  - 45
  - 50
  - 55
  - 60
  - 65

In at least one embodiment, a single semiconductor platform may refer to a sole unitary semiconductor-based integrated circuit or chip. In at least one embodiment, multi-chip modules may be used with increased connectivity which simulate on-chip operation, and make substantial improvements over utilizing a conventional central processing unit ("CPU") and bus implementation. In at least one embodiment, various modules may also be situated separately or in various combinations of semiconductor platforms per desires of user.

In at least one embodiment, referring back to FIG. 15, computer programs in form of machine-readable executable code or computer control logic algorithms are stored in main memory 1504 and/or secondary storage. Computer programs, if executed by one or more processors, enable system 1500 to perform various functions in accordance with at least one embodiment. In at least one embodiment, memory 1504, storage, and/or any other storage are possible examples of computer-readable media. In at least one embodiment, secondary storage may refer to any suitable storage device or system such as a hard disk drive and/or a removable storage drive, representing a floppy disk drive, a magnetic tape drive, a compact disk drive, digital versatile disk ("DVD") drive, recording device, universal serial bus ("USB") flash memory, etc. In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of CPU 1502, parallel processing system 1512, an integrated circuit capable of at least a portion of capabilities of both CPU 1502, parallel processing system 1512, a chipset (e.g., a group of integrated circuits

designed to work and sold as a unit for performing related functions, etc.), and/or any suitable combination of integrated circuit(s).

In at least one embodiment, architecture and/or functionality of various previous figures are implemented in context of a general computer system, a circuit board system, a game console system dedicated for entertainment purposes, an application-specific system, and more. In at least one embodiment, computer system **1500** may take form of a desktop computer, a laptop computer, a tablet computer, servers, supercomputers, a smart-phone (e.g., a wireless, hand-held device), personal digital assistant ("PDA"), a digital camera, a vehicle, a head mounted display, a hand-held electronic device, a mobile phone device, a television, workstation, game consoles, embedded system, and/or any other type of logic.

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

Other variations are within spirit of present disclosure. Thus, while disclosed techniques are susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in drawings and have been described above in detail. It should be understood, however, that there is no intention to limit disclosure to specific form or forms disclosed, but on contrary, intention is to cover all modifications, alternative constructions, and equivalents falling within spirit and scope of disclosure, as defined in appended claims.

Use of terms "a" and "an" and "the" and similar referents in context of describing disclosed embodiments (especially in context of following claims) are to be construed to cover both singular and plural, unless otherwise indicated herein or clearly contradicted by context, and not as a definition of a term. Terms "comprising," "having," "including," and "containing" are to be construed as open-ended terms (meaning "including, but not limited to,") unless otherwise noted. "Connected," when unmodified and referring to physical connections, is to be construed as partly or wholly contained within, attached to, or joined together, even if there is something intervening. Recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within range, unless otherwise indicated herein and each separate value is incorporated into specification as if it were individually recited herein. In at least one embodiment, use of term "set" (e.g., "a set of items") or "subset" unless otherwise noted or contradicted by context, is to be construed as a nonempty collection comprising one or more members. Further, unless otherwise noted or contradicted by

context, term "subset" of a corresponding set does not necessarily denote a proper subset of corresponding set, but subset and corresponding set may be equal.

Conjunctive language, such as phrases of form "at least one of A, B, and C," or "at least one of A, B and C," unless specifically stated otherwise or otherwise clearly contradicted by context, is otherwise understood with context as used in general to present that an item, term, etc., may be either A or B or C, or any nonempty subset of set of A and B and C. For instance, in illustrative example of a set having three members, conjunctive phrases "at least one of A, B, and C" and "at least one of A, B and C" refer to any of following sets: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}. Thus, such conjunctive language is not generally intended to imply that certain embodiments require at least one of A, at least one of B and at least one of C each to be present. In addition, unless otherwise noted or contradicted by context, term "plurality" indicates a state of being plural (e.g., "a plurality of items" indicates multiple items). In at least one embodiment, number of items in a plurality is at least two, but can be more when so indicated either explicitly or by context. Further, unless stated otherwise or otherwise clear from context, phrase "based on" means "based at least in part on" and not "based solely on."

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, as a result of being executed (i.e., as a result of being executed) by one or more processors of a computer system, cause computer system to perform operations described herein. In at least one embodiment, set of non-transitory computer-readable storage media comprises multiple non-transitory computer-readable storage media and one or more of individual non-transitory storage media of multiple non-transitory computer-readable storage media lack all of code while multiple non-transitory computer-readable storage media collectively store all of code. In at least one embodiment, executable instructions are executed such that different instructions are executed by different processors for example, a non-transitory computer-readable storage medium store instructions and a main central processing unit ("CPU") executes some of instructions while a graphics processing unit ("GPU") executes other instructions. In at least one embodiment, different components of a computer system have separate processors and different processors execute different subsets of instructions.

**149**

In at least one embodiment, an arithmetic logic unit is a set of combinational logic circuitry that takes one or more inputs to produce a result. In at least one embodiment, an arithmetic logic unit is used by a processor to implement mathematical operation such as addition, subtraction, or multiplication. In at least one embodiment, an arithmetic logic unit is used to implement logical operations such as logical AND/OR or XOR. In at least one embodiment, an arithmetic logic unit is stateless, and made from physical switching components such as semiconductor transistors arranged to form logical gates. In at least one embodiment, an arithmetic logic unit may operate internally as a stateful logic circuit with an associated clock. In at least one embodiment, an arithmetic logic unit may be constructed as an asynchronous logic circuit with an internal state not maintained in an associated register set. In at least one embodiment, an arithmetic logic unit is used by a processor to combine operands stored in one or more registers of the processor and produce an output that can be stored by the processor in another register or a memory location.

In at least one embodiment, as a result of processing an instruction retrieved by the processor, the processor presents one or more inputs or operands to an arithmetic logic unit, causing the arithmetic logic unit to produce a result based at least in part on an instruction code provided to inputs of the arithmetic logic unit. In at least one embodiment, the instruction codes provided by the processor to the ALU are based at least in part on the instruction executed by the processor. In at least one embodiment combinational logic in the ALU processes the inputs and produces an output which is placed on a bus within the processor. In at least one embodiment, the processor selects a destination register, memory location, output device, or output storage location on the output bus so that clocking the processor causes the results produced by the ALU to be sent to the desired location.

In the scope of this application, the term arithmetic logic unit, or ALU, is used to refer to any computational logic circuit that processes operands to produce a result. For example, in the present document, the term ALU can refer to a floating point unit, a DSP, a tensor core, a shader core, a coprocessor, or a CPU.

Accordingly, in at least one embodiment, computer systems are configured to implement one or more services that singly or collectively perform operations of processes described herein and such computer systems are configured with applicable hardware and/or software that enable performance of operations. Further, a computer system that implements at least one embodiment of present disclosure is a single device and, in another embodiment, is a distributed computer system comprising multiple devices that operate differently such that distributed computer system performs operations described herein and such that a single device does not perform all operations.

Use of any and all examples, or exemplary language (e.g., "such as") provided herein, is intended merely to better illuminate embodiments of disclosure and does not pose a limitation on scope of disclosure unless otherwise claimed. No language in specification should be construed as indicating any non-claimed element as essential to practice of disclosure.

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

**150**

In description and claims, terms "coupled" and "connected," along with their derivatives, may be used. It should be understood that these terms may be not intended as synonyms for each other. Rather, in particular examples, "connected" or "coupled" may be used to indicate that two or more elements are in direct or indirect physical or electrical contact with each other. "Coupled" may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other.

Unless specifically stated otherwise, it may be appreciated that throughout specification terms such as "processing," "computing," "calculating," "determining," or like, refer to action and/or processes of a computer or computing system, or similar electronic computing device, that manipulate and/or transform data represented as physical, such as electronic, quantities within computing system's registers and/or memories into other data similarly represented as physical quantities within computing system's memories, registers or other such information storage, transmission or display devices.

In a similar manner, term "processor" may refer to any device or portion of a device that processes electronic data from registers and/or memory and transform that electronic data into other electronic data that may be stored in registers and/or memory. As non-limiting examples, "processor" may be a CPU or a GPU. A "computing platform" may comprise one or more processors. As used herein, "software" processes may include, for example, software and/or hardware entities that perform work over time, such as tasks, threads, and intelligent agents. Also, each process may refer to multiple processes, for carrying out instructions in sequence or in parallel, continuously or intermittently. In at least one embodiment, terms "system" and "method" are used herein interchangeably insofar as system may embody one or more methods and methods may be considered a system.

In present document, references may be made to obtaining, acquiring, receiving, or inputting analog or digital data into a subsystem, computer system, or computer-implemented machine. In at least one embodiment, process of obtaining, acquiring, receiving, or inputting analog and digital data can be accomplished in a variety of ways such as by receiving data as a parameter of a function call or a call to an application programming interface. In at least one embodiment, processes of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a serial or parallel interface. In at least one embodiment, processes of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a computer network from providing entity to acquiring entity. In at least one embodiment, references may also be made to providing, outputting, transmitting, sending, or presenting analog or digital data. In various examples, processes of providing, outputting, transmitting, sending, or presenting analog or digital data can be accomplished by transferring data as an input or output parameter of a function call, a parameter of an application programming interface or interprocess communication mechanism.

Although descriptions herein set forth example implementations of described techniques, other architectures may be used to implement described functionality, and are intended to be within scope of this disclosure. Furthermore, although specific distributions of responsibilities may be defined above for purposes of description, various functions and responsibilities might be distributed and divided in different ways, depending on circumstances.

**151**

Furthermore, although subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that subject matter claimed in appended claims is not necessarily limited to specific features or acts described. Rather, specific features and acts are disclosed as exemplary forms of implementing the claims.

What is claimed is:

- 1.** A method comprising:  
obtaining data associated with a streaming session;  
predicting an interval of time during which an in-game highlight event is predicted to occur in the streaming session by analyzing user inputs associated with the streaming session;  
based at least in part on the prediction, causing the streaming session to be modified to generate a modified streaming session by at least applying an effect to one or more frames corresponding to the streaming session and the predicted interval of time; and  
causing the modified streaming session to be broadcast after the in-game highlight event occurs within the streaming session.
- 2.** The method of claim **1**, wherein the method further comprises:  
capturing a set of inputs associated with the streaming session;  
generating a determination that the in-game highlight event has occurred based at least in part on the set of inputs; and  
causing the effect to no longer be applied to one or more frames corresponding to the modified streaming session.
- 3.** The method of claim **2**, wherein the set of inputs include at least one of an image, audio, or user input.
- 4.** The method of claim **2**, wherein generating the determination that the in-game highlight event has occurred is performed using a model trained to detect objects in one or more image frames corresponding to the streaming session.
- 5.** The method of claim **1**, wherein the method further comprises:  
determining an expiration of the predicted interval of time; and  
causing the effect to no longer be applied to one or more other frames corresponding to the streaming session.
- 6.** The method of claim **1**, wherein the streaming session corresponds to a gameplay session initiated by a server.
- 7.** The method of claim **1**, wherein data associated with the streaming session includes at least one of: input data obtained from a remote client, one or more image frames corresponding to the streaming session, audio of the streaming session, or audio captured from the remote client.
- 8.** The method of claim **1**, wherein the effect further comprises an overlay displayed in one or more image frames corresponding to the streaming session.
- 9.** The method of claim **1**, wherein the effect further comprises an audio cue included in one or more audio frames corresponding to the streaming session.
- 10.** The method of claim **1**, wherein the user inputs comprise inputs obtained from a controller device associated with a game session.
- 11.** The method of claim **10**, wherein the controller device includes at least one of: a gaming console controller, a track pad, a mouse, a keyboard, or a motion controller.
- 12.** A non-transitory machine-readable medium having stored thereon a set of instructions, which, if performed by one or more processors, cause the one or more processors to:

**152**

- obtain data generated during a streaming session;  
predict an interval of time during which an in-game highlight event is predicted to occur in the streaming session by analyzing user inputs associated with the streaming session;  
based at least in part on the interval of time, cause the streaming session to be modified by at least applying an asset to the streaming session corresponding to the predicted interval of time; and  
broadcast the modified streaming session after the in-game highlight event occurs within the streaming session.
- 13.** The non-transitory machine-readable medium of claim **12**, wherein the set of instructions further include instructions, which, if performed by the one or more processors, cause the one or more processors to:  
capture an image corresponding to the streaming session; generate a determination that the in-game highlight event has occurred based at least in part on the image; and cause a state of the asset to be modified.
- 14.** The non-transitory machine-readable medium of claim **12**, wherein the set of instructions further include instructions, which, if performed by the one or more processors, cause the one or more processors to determine the predicted interval of time based at least in part on an event type associated with the in-game highlight event and a type associated with the asset.
- 15.** The non-transitory machine-readable medium of claim **12**, wherein the asset further comprises an effect applied to the streaming session.
- 16.** The non-transitory machine-readable medium of claim **15**, wherein the effect adds a delay between successive frames of the streaming session.
- 17.** The non-transitory machine-readable medium of claim **12**, wherein the set of instructions that cause the one or more processors to predict the in-game highlight event further include instructions, which, if performed by the one or more processors, cause the one or more processors to predict the in-game highlight event based at least in part on audio or video captured at a host computer system associated with the streaming session.
- 18.** The non-transitory machine-readable medium of claim **12**, wherein the user inputs comprise inputs from a controller device logged at a host computer system associated with the streaming session.
- 19.** A system comprising:  
one or more processors; and  
memory storing instructions that, as a result of being executed by the one or more processors, cause the system to:  
cause a determination to be generated, the determination predicting an interval of time during which an in-game highlight event is predicted to occur in a streaming session by analyzing user inputs associated with the streaming session, based at least in part on data associated with a streaming session;  
in response to the determination, cause an effect to be applied to the streaming session corresponding to the predicted interval of time; and  
broadcast the streaming session with the effect applied after the in-game highlight event occurs within the streaming session.
- 20.** The system of claim **19**, wherein the memory further stores instructions that, as a result of being executed by the one or more processors, cause the system to:  
during the predicted interval of time, obtain a set of images of the streaming session; and

**153**

generate a second determination that the in-game highlight event occurred prior to an expiration of the predicted interval of time.

**21.** The system of claim **19**, wherein the memory further stores instructions that, as a result of being executed by the one or more processors, cause the system to, in response to an expiration of the predicted interval of time, cause the effect to no longer be applied to the streaming session.

**22.** The system of claim **19**, wherein the data associated with the streaming session includes keystroke data obtained from a remote client.

**23.** The system of claim **19**, wherein the instructions that cause the determination to be generated further include instructions that, as a result of being executed by the one or more processors, cause the system to cause a machine learning model, executed by a server, to generate the determination.

**24.** The system of claim **23**, wherein the machine learning model generates the determination based at least in part on images of a game session captured by the system.

**154**

**25.** The system of claim **23**, wherein the machine learning model generates the determination based at least in part on information obtained from a game engine.

**26.** The system of claim **23**, wherein the machine learning model generates the determination based at least in part on information obtained from a game associated with the streaming session.

**27.** The system of claim **23**, wherein the determination further comprises information indicating a type associated with the in-game highlight event.

**28.** The system of claim **19**, wherein the instructions that cause the effect to be applied to the streaming session further include instructions that, as a result of being executed by the one or more processors, cause the system to continue causing the effect to be applied as a result of determining the in-game highlight event occurred.

\* \* \* \* \*