



(51) International Patent Classification:

G06F 9/50 (2006.01) G06F 3/06 (2006.01)
G06T 1/20 (2006.01) G06F 11/34 (2006.01)
G06F 9/38 (2006.01)

(21) International Application Number:

PCT/US2016/031870

(22) International Filing Date:

11 May 2016 (11.05.2016)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

14/738,660 12 June 2015 (12.06.2015) US

(71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).

(72) Inventors: SCHLUESSLER, Travis T.; 3719 Coyote Trail, Berthoud, Colorado 80513 (US). GIERACH, John G.; 4232 NW Silverleaf Drive, Portland, Oregon 97229 (US).

(74) Agents: MALLIE, Michael J. et al.; Blakely Sokoloff Taylor & Zafman LLP, 1279 Oakmead Parkway, Sunnyvale, California 94085 (US).

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

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: FACILITATING CONFIGURATION OF COMPUTING ENGINES BASED ON RUNTIME WORKLOAD MEASUREMENTS AT COMPUTING DEVICES

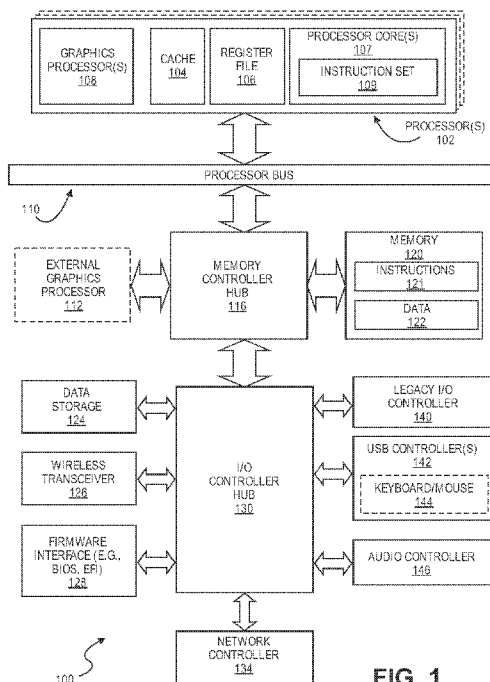


FIG. 1

(57) Abstract: A mechanism is described for facilitating configuration of computing engines based on runtime workload measurements at computing devices. A method of embodiments, as described herein, includes detecting a work unit corresponding to a workload, and collecting metrics relating to the work unit, where the metrics to indicate one or more characteristics of the work unit. The method may further include evaluating the one or more characteristics based on one or more configuration parameters relating to computing resources, and generating, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit. The method may further include applying and executing the configuration plan at a computing device upon execution of the one or more subsequent work units.



Published:

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

FACILITATING CONFIGURATION OF COMPUTING ENGINES BASED ON RUNTIME WORKLOAD MEASUREMENTS AT COMPUTING DEVICES

FIELD

5 Embodiments described herein generally relate to computers. More particularly, embodiments relate to a mechanism for facilitating configuration of computing engines based on runtime workload measurements at computing devices.

BACKGROUND

10 Conventional configuration techniques for three-dimensional (3D) workloads are incapable of executing optimal configuration of 3D workloads for any number of reasons, such as the entire previous and next time windows are not known for executing the same type of workload. For example, conventional configuration techniques are not capable of knowing optimal machine configurations until after workloads have been executed and such conventional techniques average measurements over a time window that includes different types of work, resulting in a
15 machine configuration that may be acceptable for average characteristics of that work, but not optimal for any specific subslice of it and further, such a configuration is incapable of being applied to previous chunks of work.

BRIEF DESCRIPTION OF THE DRAWINGS

20 Embodiments are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

Figure 1 is a block diagram of a processing system, according to an embodiment.

Figure 2 is a block diagram of an embodiment of a processor having one or more processor cores, an integrated memory controller, and an integrated graphics processor.

25 **Figure 3** is a block diagram of a graphics processor, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores.

Figure 4 is a block diagram of a graphics processing engine of a graphics processor in accordance with some embodiments.

Figure 5 is a block diagram of another embodiment of a graphics processor.

30 **Figure 6** illustrates thread execution logic including an array of processing elements employed in some embodiments of a graphics processing engine.

Figure 7 is a block diagram illustrating a graphics processor instruction formats according to some embodiments.

Figure 8 is a block diagram of another embodiment of a graphics processor.

35 **Figure 9A** is a block diagram illustrating a graphics processor command format according to an embodiment and **Figure 9B** is a block diagram illustrating a graphics processor command

sequence according to an embodiment.

Figure 10 illustrates exemplary graphics software architecture for a data processing system according to some embodiments.

Figure 11 is a block diagram illustrating an IP core development system that may be used to manufacture an integrated circuit to perform operations according to an embodiment.

Figure 12 is a block diagram illustrating an exemplary system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

Figure 13 illustrates a computing device employing a runtime metrics-based workload management mechanism according to one embodiment.

Figure 14 illustrates a runtime metrics-based workload management mechanism according to one embodiment.

Figure 15 illustrates a method for facilitating metrics-based workload management according to one embodiment.

DETAILED DESCRIPTION

In the following description, numerous specific details are set forth. However, embodiments, as described herein, may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in details in order not to obscure the understanding of this description.

Embodiments provide for a novel technique for measuring and collecting metrics relating to each type of draw each time it occurs, where such metrics are then predictively used for efficient use of a GPU and other relevant resource of a computing device, leading to increased efficiency in power, speed, performance, etc. For example and in one embodiment, metrics collected regarding a draw (e.g., relating to a portion of a human nose) in a first frame may be used to determine the resources needed to efficiently process a similar draw (e.g., relating to the same portion of the same human nose, but possibly slightly different, such as a bit shaded, etc.) in a second frame. Using this novel technique, metrics relating to all draws in all frames may be continuously collected and predictively used for subsequent draws in subsequent frames and so on and so forth.

It is contemplated that a workload (also referred to as “application”) may include any number of draws (also referred to as “work units”) such that these draws may vary for each frame within the workload, such as a draw profile of previous 50 milliseconds (*ms*) is likely to be different from a draw profile of next 50 milliseconds. For example, a first draw may run over 10 *ms* having 50 small triangles, where a second draw may run over 100 *ms* having one large triangle and thus a graphics processing unit (GPU) may need to be configured differently to process each such draw differently from the other. As aforementioned, this is one of several

reasons why conventional techniques have been unsuccessful.

For example, a workload or an application, such as a game, a workstation application, etc., may be offered thorough an application programming interface (API), such as a free rendering API, such as Open Graphics Library (OpenGL®), DirectX® 11, etc. It is contemplated that each application may include a number of frames, where each frame represents an image (e.g., classroom, sky, face, etc.) and includes a number of draws, where each draw represents a portion of the image (e.g., desk in classroom, cloud in sky, nose on face, etc.) of its corresponding frame. To complete processing an image, each frame is processed in its entirety, including all its draws.

In some embodiments, terms like “display screen” and “display surface” may be used interchangeably referring to the visible portion of a display device while the rest of the display device may be embedded into a computing device, such as a smartphone, a wearable device, etc. It is contemplated and to be noted that embodiments are not limited to any particular computing device, software application, hardware component, display device, display screen or surface, protocol, standard, etc. For example, embodiments may be applied to and used with any number and type of real-time applications on any number and type of computers, such as desktops, laptops, tablet computers, smartphones, head-mounted displays and other wearable devices, and/or the like. Further, for example, rendering scenarios using this novel technique for efficient performance of 3D workloads may range from simple scenarios, such as desktop compositing, to complex scenarios, such as 3D games, augmented reality applications, etc.

System Overview

Figure 1 is a block diagram of a processing system 100, according to an embodiment. In various embodiments the system 100 includes one or more processors 102 and one or more graphics processors 108, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 102 or processor cores 107. In on embodiment, the system 100 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

An embodiment of system 100 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 some embodiments system 100 is a mobile phone, smart phone, tablet computing device or mobile Internet device. Data processing system 100 can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, smart eyewear device, augmented reality device, or virtual reality device. In some embodiments, data processing system 100 is a television or set top box device having one or more processors 102 and a graphical interface generated by one or more graphics processors 108.

In some embodiments, the one or more processors 102 each include one or more processor cores 107 to process instructions which, when executed, perform operations for system and user software. In some embodiments, each of the one or more processor cores 107 is configured to process a specific instruction set 109. In some embodiments, instruction set 109 may facilitate
5 Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). Multiple processor cores 107 may each process a different instruction set 109, which may include instructions to facilitate the emulation of other instruction sets. Processor core 107 may also include other processing devices, such as Digital Signal Processor (DSP).

10 In some embodiments, the processor 102 includes cache memory 104. Depending on the architecture, the processor 102 can have a single internal cache or multiple levels of internal cache. In some embodiments, the cache memory is shared among various components of the processor 102. In some embodiments, the processor 102 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among
15 processor cores 107 using known cache coherency techniques. A register file 106 is additionally included in processor 102 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). Some registers may be general-purpose registers, while other registers may be specific to the design of the processor 102.

20 In some embodiments, processor 102 is coupled to a processor bus 110 to transmit communication signals such as address, data, or control signals between processor 102 and other components in system 100. In one embodiment the system 100 uses an exemplary 'hub' system architecture, including a memory controller hub 116 and an Input Output (I/O) controller hub 130. A memory controller hub 116 facilitates communication between a memory device and
25 other components of system 100, while an I/O Controller Hub (ICH) 130 provides connections to I/O devices via a local I/O bus. In one embodiment, the logic of the memory controller hub 116 is integrated within the processor.

Memory device 120 can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or
30 some other memory device having suitable performance to serve as process memory. In one embodiment the memory device 120 can operate as system memory for the system 100, to store data 122 and instructions 121 for use when the one or more processors 102 executes an application or process. Memory controller hub 116 also couples with an optional external graphics processor 112, which may communicate with the one or more graphics processors 108
35 in processors 102 to perform graphics and media operations.

In some embodiments, ICH 130 enables peripherals to connect to memory device 120 and processor 102 via a high-speed I/O bus. The I/O peripherals include, but are not limited to, an audio controller 146, a firmware interface 128, a wireless transceiver 126 (e.g., Wi-Fi, Bluetooth), a data storage device 124 (e.g., hard disk drive, flash memory, etc.), and a legacy I/O controller 140 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to the system. One or more Universal Serial Bus (USB) controllers 142 connect input devices, such as keyboard and mouse 144 combinations. A network controller 134 may also couple to ICH 130. In some embodiments, a high-performance network controller (not shown) couples to processor bus 110. It will be appreciated that the system 100 shown is exemplary and not limiting, as other types of data processing systems that are differently configured may also be used. For example, the I/O controller hub 130 may be integrated within the one or more processor 102, or the memory controller hub 116 and I/O controller hub 130 may be integrated into a discrete external graphics processor, such as the external graphics processor 112.

Figure 2 is a block diagram of an embodiment of a processor 200 having one or more processor cores 202A-202N, an integrated memory controller 214, and an integrated graphics processor 208. Those elements of **Fig. 2** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such. Processor 200 can include additional cores up to and including additional core 202N represented by the dashed lined boxes. Each of processor cores 202A-202N includes one or more internal cache units 204A-204N. In some embodiments each processor core also has access to one or more shared cache units 206.

The internal cache units 204A-204N and shared cache units 206 represent a cache memory hierarchy within the processor 200. The cache memory hierarchy 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 the highest level of cache before external memory is classified as the LLC. In some embodiments, cache coherency logic maintains coherency between the various cache units 206 and 204A-204N.

In some embodiments, processor 200 may also include a set of one or more bus controller units 216 and a system agent core 210. The one or more bus controller units 216 manage a set of peripheral buses, such as one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express). System agent core 210 provides management functionality for the various processor components. In some embodiments, system agent core 210 includes one or more integrated memory controllers 214 to manage access to various external memory devices (not shown).

In some embodiments, one or more of the processor cores 202A-202N include support for

simultaneous multi-threading. In such embodiment, the system agent core 210 includes components for coordinating and operating cores 202A-202N during multi-threaded processing. System agent core 210 may additionally include a power control unit (PCU), which includes logic and components to regulate the power state of processor cores 202A-202N and graphics processor 208.

In some embodiments, processor 200 additionally includes graphics processor 208 to execute graphics processing operations. In some embodiments, the graphics processor 208 couples with the set of shared cache units 206, and the system agent core 210, including the one or more integrated memory controllers 214. In some embodiments, a display controller 211 is coupled with the graphics processor 208 to drive graphics processor output to one or more coupled displays. In some embodiments, display controller 211 may be a separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 208 or system agent core 210.

In some embodiments, a ring based interconnect unit 212 is used to couple the internal components of the processor 200. However, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques, including techniques well known in the art. In some embodiments, graphics processor 208 couples with the ring interconnect 212 via an I/O link 213.

The exemplary I/O link 213 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 218, such as an eDRAM module. In some embodiments, each of the processor cores 202-202N and graphics processor 208 use embedded memory modules 218 as a shared Last Level Cache.

In some embodiments, processor cores 202A-202N are homogenous cores executing the same instruction set architecture. In another embodiment, processor cores 202A-202N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 202A-N execute a first instruction set, while at least one of the other cores executes a subset of the first instruction set or a different instruction set. In one embodiment processor cores 202A-202N 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. Additionally, processor 200 can be implemented on one or more chips or as an SoC integrated circuit having the illustrated components, in addition to other components.

Figure 3 is a block diagram of a graphics processor 300, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores.

In some embodiments, the graphics processor communicates via a memory mapped I/O interface to registers on the graphics processor and with commands placed into the processor memory. In some embodiments, graphics processor 300 includes a memory interface 314 to access memory. Memory interface 314 can be an interface to local memory, one or more internal caches, one or
5 more shared external caches, and/or to system memory.

In some embodiments, graphics processor 300 also includes a display controller 302 to drive display output data to a display device 320. Display controller 302 includes hardware for one or more overlay planes for the display and composition of multiple layers of video or user interface elements. In some embodiments, graphics processor 300 includes a video codec engine
10 306 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 some embodiments, graphics processor 300 includes a block image transfer (BLIT) engine 304 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in one embodiment, 2D graphics operations are performed using one or more components of graphics processing engine (GPE) 310. In some embodiments, graphics processing engine 310 is a compute engine for performing graphics operations,
15 including three-dimensional (3D) graphics operations and media operations.

In some embodiments, GPE 310 includes a 3D pipeline 312 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.). The 3D pipeline 312 includes programmable and fixed function elements that perform various tasks within the element and/or spawn
25 execution threads to a 3D/Media sub-system 315. While 3D pipeline 312 can be used to perform media operations, an embodiment of GPE 310 also includes a media pipeline 316 that is specifically used to perform media operations, such as video post-processing and image enhancement.

In some embodiments, media pipeline 316 includes fixed function or programmable logic
30 units to perform one or more specialized media operations, such as video decode acceleration, video de-interlacing, and video encode acceleration in place of, or on behalf of video codec engine 306. In some embodiments, media pipeline 316 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 315. The spawned threads perform computations for the media operations on one or more graphics execution units included in
35 3D/Media sub-system 315.

In some embodiments, 3D/Media subsystem 315 includes logic for executing threads spawned by 3D pipeline 312 and media pipeline 316. In one embodiment, the pipelines send thread execution requests to 3D/Media subsystem 315, which includes thread dispatch logic for arbitrating and dispatching the various requests to available thread execution resources. The execution resources include an array of graphics execution units to process the 3D and media threads. In some embodiments, 3D/Media subsystem 315 includes one or more internal caches for thread instructions and data. In some embodiments, the subsystem also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

3D/Media Processing

Figure 4 is a block diagram of a graphics processing engine 410 of a graphics processor in accordance with some embodiments. In one embodiment, the GPE 410 is a version of the GPE 310 shown in **Fig. 3**. Elements of **Fig. 4** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

In some embodiments, GPE 410 couples with a command streamer 403, which provides a command stream to the GPE 3D and media pipelines 412, 416. In some embodiments, command streamer 403 is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In some embodiments, command streamer 403 receives commands from the memory and sends the commands to 3D pipeline 412 and/or media pipeline 416. The commands are directives fetched from a ring buffer, which stores commands for the 3D and media pipelines 412, 416. In one embodiment, the ring buffer can additionally include batch command buffers storing batches of multiple commands. The 3D and media pipelines 412, 416 process the commands by performing operations via logic within the respective pipelines or by dispatching one or more execution threads to an execution unit array 414. In some embodiments, execution unit array 414 is scalable, such that the array includes a variable number of execution units based on the target power and performance level of GPE 410.

In some embodiments, a sampling engine 430 couples with memory (e.g., cache memory or system memory) and execution unit array 414. In some embodiments, sampling engine 430 provides a memory access mechanism for execution unit array 414 that allows execution array 414 to read graphics and media data from memory. In some embodiments, sampling engine 430 includes logic to perform specialized image sampling operations for media.

In some embodiments, the specialized media sampling logic in sampling engine 430 includes a de-noise/de-interlace module 432, a motion estimation module 434, and an image scaling and filtering module 436. In some embodiments, de-noise/de-interlace module 432

includes logic to perform one or more of a de-noise or a de-interlace algorithm on decoded video data. The de-interlace logic combines alternating fields of interlaced video content into a single frame of video. The de-noise logic reduces or removes data noise from video and image data. In some embodiments, the de-noise logic and de-interlace logic are motion adaptive and use spatial or temporal filtering based on the amount of motion detected in the video data. In some embodiments, the de-noise/de-interlace module 432 includes dedicated motion detection logic (e.g., within the motion estimation engine 434).

In some embodiments, motion estimation engine 434 provides hardware acceleration for video operations by performing video acceleration functions such as motion vector estimation and prediction on video data. The motion estimation engine determines motion vectors that describe the transformation of image data between successive video frames. In some embodiments, a graphics processor media codec uses video motion estimation engine 434 to perform operations on video at the macro-block level that may otherwise be too computationally intensive to perform with a general-purpose processor. In some embodiments, motion estimation engine 434 is generally available to graphics processor components to assist with video decode and processing functions that are sensitive or adaptive to the direction or magnitude of the motion within video data.

In some embodiments, image scaling and filtering module 436 performs image-processing operations to enhance the visual quality of generated images and video. In some embodiments, scaling and filtering module 436 processes image and video data during the sampling operation before providing the data to execution unit array 414.

In some embodiments, the GPE 410 includes a data port 444, which provides an additional mechanism for graphics subsystems to access memory. In some embodiments, data port 444 facilitates memory access for operations including render target writes, constant buffer reads, scratch memory space reads/writes, and media surface accesses. In some embodiments, data port 444 includes cache memory space to cache accesses to memory. The cache memory can be a single data cache or separated into multiple caches for the multiple subsystems that access memory via the data port (e.g., a render buffer cache, a constant buffer cache, etc.). In some embodiments, threads executing on an execution unit in execution unit array 414 communicate with the data port by exchanging messages via a data distribution interconnect that couples each of the sub-systems of GPE 410.

Execution Units

Figure 5 is a block diagram of another embodiment of a graphics processor 500. Elements of **Fig. 5** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are

not limited to such.

In some embodiments, graphics processor 500 includes a ring interconnect 502, a pipeline front-end 504, a media engine 537, and graphics cores 580A-580N. In some embodiments, ring interconnect 502 couples the graphics processor to other processing units, including other
5 graphics processors or one or more general-purpose processor cores. In some embodiments, the graphics processor is one of many processors integrated within a multi-core processing system.

In some embodiments, graphics processor 500 receives batches of commands via ring interconnect 502. The incoming commands are interpreted by a command streamer 503 in the pipeline front-end 504. In some embodiments, graphics processor 500 includes scalable
10 execution logic to perform 3D geometry processing and media processing via the graphics core(s) 580A-580N. For 3D geometry processing commands, command streamer 503 supplies commands to geometry pipeline 536. For at least some media processing commands, command streamer 503 supplies the commands to a video front end 534, which couples with a media engine 537. In some embodiments, media engine 537 includes a Video Quality Engine (VQE)
15 530 for video and image post-processing and a multi-format encode/decode (MFX) 533 engine to provide hardware-accelerated media data encode and decode. In some embodiments, geometry pipeline 536 and media engine 537 each generate execution threads for the thread execution resources provided by at least one graphics core 580A.

In some embodiments, graphics processor 500 includes scalable thread execution resources
20 featuring modular cores 580A-580N (sometimes referred to as core slices), each having multiple sub-cores 550A-550N, 560A-560N (sometimes referred to as core sub-slices). In some embodiments, graphics processor 500 can have any number of graphics cores 580A through 580N. In some embodiments, graphics processor 500 includes a graphics core 580A having at least a first sub-core 550A and a second core sub-core 560A. In other embodiments, the
25 graphics processor is a low power processor with a single sub-core (e.g., 550A). In some embodiments, graphics processor 500 includes multiple graphics cores 580A-580N, each including a set of first sub-cores 550A-550N and a set of second sub-cores 560A-560N. Each sub-core in the set of first sub-cores 550A-550N includes at least a first set of execution units 552A-552N and media/texture samplers 554A-554N. Each sub-core in the set of second sub-
30 cores 560A-560N includes at least a second set of execution units 562A-562N and samplers 564A-564N. In some embodiments, each sub-core 550A-550N, 560A-560N shares a set of shared resources 570A-570N. In some embodiments, the shared resources include shared cache memory and pixel operation logic. Other shared resources may also be included in the various embodiments of the graphics processor.

35 **Figure 6** illustrates thread execution logic 600 including an array of processing elements

employed in some embodiments of a GPE. Elements of **Fig. 6** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

In some embodiments, thread execution logic 600 includes a pixel shader 602, a thread dispatcher 604, instruction cache 606, a scalable execution unit array including a plurality of execution units 608A-608N, a sampler 610, a data cache 612, and a data port 614. In one embodiment the included components are interconnected via an interconnect fabric that links to each of the components. In some embodiments, thread execution logic 600 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 606, data port 614, sampler 610, and execution unit array 608A-608N. In some embodiments, each execution unit (e.g. 608A) is an individual vector processor capable of executing multiple simultaneous threads and processing multiple data elements in parallel for each thread. In some embodiments, execution unit array 608A-608N includes any number individual execution units.

In some embodiments, execution unit array 608A-608N is primarily used to execute “shader” programs. In some embodiments, the execution units in array 608A-608N execute 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. The execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, vertex shaders), pixel processing (e.g., pixel shaders, fragment shaders) and general-purpose processing (e.g., compute and media shaders).

Each execution unit in execution unit array 608A-608N operates on arrays of data elements. The number of data elements is the “execution size,” or the number of channels for the instruction. An execution channel is a logical unit of execution for data element access, masking, and flow control within instructions. The number of channels may be independent of the number of physical Arithmetic Logic Units (ALUs) or Floating Point Units (FPUs) for a particular graphics processor. In some embodiments, execution units 608A-608N support integer and floating-point data types.

The execution unit instruction set includes single instruction multiple data (SIMD) instructions. The various data elements can be stored as a packed data type in a register and the execution unit will process the various elements based on the data size of the elements. For example, when operating on a 256-bit wide vector, the 256 bits of the vector are stored in a register and the execution unit operates on the 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, different vector widths and register sizes are possible.

One or more internal instruction caches (e.g., 606) are included in the thread execution logic 600 to cache thread instructions for the execution units. In some embodiments, one or
5 more data caches (e.g., 612) are included to cache thread data during thread execution. In some embodiments, sampler 610 is included to provide texture sampling for 3D operations and media sampling for media operations. In some embodiments, sampler 610 includes specialized texture or media sampling functionality to process texture or media data during the sampling process before providing the sampled data to an execution unit.

10 During execution, the graphics and media pipelines send thread initiation requests to thread execution logic 600 via thread spawning and dispatch logic. In some embodiments, thread execution logic 600 includes a local thread dispatcher 604 that arbitrates thread initiation requests from the graphics and media pipelines and instantiates the requested threads on one or more execution units 608A-608N. For example, the geometry pipeline (e.g., 536 of **Fig. 5**)
15 dispatches vertex processing, tessellation, or geometry processing threads to thread execution logic 600 (**Fig. 6**). In some embodiments, thread dispatcher 604 can also process runtime thread spawning requests from the executing shader programs.

Once a group of geometric objects has been processed and rasterized into pixel data, pixel shader 602 is invoked to further compute output information and cause results to be written to
20 output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In some embodiments, pixel shader 602 calculates the values of the various vertex attributes that are to be interpolated across the rasterized object. In some embodiments, pixel shader 602 then executes an application programming interface (API)-supplied pixel shader program. To execute the pixel shader program, pixel shader 602 dispatches threads to an execution unit (e.g., 608A) via thread
25 dispatcher 604. In some embodiments, pixel shader 602 uses texture sampling logic in sampler 610 to access texture data in texture maps stored in memory. Arithmetic operations on the texture data and the input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

In some embodiments, the data port 614 provides a memory access mechanism for the
30 thread execution logic 600 output processed data to memory for processing on a graphics processor output pipeline. In some embodiments, the data port 614 includes or couples to one or more cache memories (e.g., data cache 612) to cache data for memory access via the data port.

Figure 7 is a block diagram illustrating a graphics processor instruction formats 700 according to some embodiments. In one or more embodiment, the graphics processor execution
35 units support an instruction set having instructions in multiple formats. The solid lined boxes

illustrate the components that are generally included in an execution unit instruction, while the dashed lines include components that are optional or that are only included in a sub-set of the instructions. In some embodiments, instruction format 700 described and illustrated are macro-instructions, in that they are instructions supplied to the execution unit, as opposed to micro-operations resulting from instruction decode once the instruction is processed.

In some embodiments, the graphics processor execution units natively support instructions in a 128-bit format 710. A 64-bit compacted instruction format 730 is available for some instructions based on the selected instruction, instruction options, and number of operands. The native 128-bit format 710 provides access to all instruction options, while some options and operations are restricted in the 64-bit format 730. The native instructions available in the 64-bit format 730 vary by embodiment. In some embodiments, the instruction is compacted in part using a set of index values in an index field 713. The execution unit hardware references a set of compaction tables based on the index values and uses the compaction table outputs to reconstruct a native instruction in the 128-bit format 710.

For each format, instruction opcode 712 defines the operation that the execution unit is to perform. The execution units execute each instruction in parallel across the multiple data elements of each operand. For example, in response to an add instruction the execution unit performs a simultaneous add operation across each color channel representing a texture element or picture element. By default, the execution unit performs each instruction across all data channels of the operands. In some embodiments, instruction control field 714 enables control over certain execution options, such as channels selection (e.g., predication) and data channel order (e.g., swizzle). For 128-bit instructions 710 an exec-size field 716 limits the number of data channels that will be executed in parallel. In some embodiments, exec-size field 716 is not available for use in the 64-bit compact instruction format 730.

Some execution unit instructions have up to three operands including two source operands, src0 722, src1 722, and one destination 718. In some embodiments, the execution units support dual destination instructions, where one of the destinations is implied. Data manipulation instructions can have a third source operand (e.g., SRC2 724), where the instruction opcode 712 determines the number of source operands. An instruction's last source operand can be an immediate (e.g., hard-coded) value passed with the instruction.

In some embodiments, the 128-bit instruction format 710 includes an access/address mode information 726 specifying, for example, whether direct register addressing mode or indirect register addressing mode is used. When direct register addressing mode is used, the register address of one or more operands is directly provided by bits in the instruction 710.

In some embodiments, the 128-bit instruction format 710 includes an access/address mode

field 726, which specifies an address mode and/or an access mode for the instruction. In one embodiment the access mode to define a data access alignment for the instruction. Some embodiments support access modes including a 16-byte aligned access mode and a 1-byte aligned access mode, where the byte alignment of the access mode determines the access alignment of the instruction operands. For example, when in a first mode, the instruction 710 may use byte-aligned addressing for source and destination operands and when in a second mode, the instruction 710 may use 16-byte-aligned addressing for all source and destination operands.

In one embodiment, the address mode portion of the access/address mode field 726 determines whether the instruction is to use direct or indirect addressing. When direct register addressing mode is used bits in the instruction 710 directly provide the register address of one or more operands. When indirect register addressing mode is used, the register address of one or more operands may be computed based on an address register value and an address immediate field in the instruction.

In some embodiments instructions are grouped based on opcode 712 bit-fields to simplify Opcode decode 740. For an 8-bit opcode, bits 4, 5, and 6 allow the execution unit to determine the type of opcode. The precise opcode grouping shown is merely an example. In some embodiments, a move and logic opcode group 742 includes data movement and logic instructions (e.g., move (mov), compare (cmp)). In some embodiments, move and logic group 742 shares the five most significant bits (MSB), where move (mov) instructions are in the form of 0000xxxxb and logic instructions are in the form of 0001xxxxb. A flow control instruction group 744 (e.g., call, jump (jmp)) includes instructions in the form of 0010xxxxb (e.g., 0x20). A miscellaneous instruction group 746 includes a mix of instructions, including synchronization instructions (e.g., wait, send) in the form of 0011xxxxb (e.g., 0x30). A parallel math instruction group 748 includes component-wise arithmetic instructions (e.g., add, multiply (mul)) in the form of 0100xxxxb (e.g., 0x40). The parallel math group 748 performs the arithmetic operations in parallel across data channels. The vector math group 750 includes arithmetic instructions (e.g., dp4) in the form of 0101xxxxb (e.g., 0x50). The vector math group performs arithmetic such as dot product calculations on vector operands.

Graphics Pipeline

Figure 8 is a block diagram of another embodiment of a graphics processor 800. Elements of **Fig. 8** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

In some embodiments, graphics processor 800 includes a graphics pipeline 820, a media

pipeline 830, a display engine 840, thread execution logic 850, and a render output pipeline 870. In some embodiments, graphics processor 800 is a graphics processor within a multi-core processing system that includes one or more general purpose processing cores. The graphics processor is controlled by register writes to one or more control registers (not shown) or via
5 commands issued to graphics processor 800 via a ring interconnect 802. In some embodiments, ring interconnect 802 couples graphics processor 800 to other processing components, such as other graphics processors or general-purpose processors. Commands from ring interconnect 802 are interpreted by a command streamer 803, which supplies instructions to individual components of graphics pipeline 820 or media pipeline 830.

10 In some embodiments, command streamer 803 directs the operation of a vertex fetcher 805 that reads vertex data from memory and executes vertex-processing commands provided by command streamer 803. In some embodiments, vertex fetcher 805 provides vertex data to a vertex shader 807, which performs coordinate space transformation and lighting operations to each vertex. In some embodiments, vertex fetcher 805 and vertex shader 807 execute vertex-
15 processing instructions by dispatching execution threads to execution units 852A, 852B via a thread dispatcher 831.

In some embodiments, execution units 852A, 852B are an array of vector processors having an instruction set for performing graphics and media operations. In some embodiments, execution units 852A, 852B have an attached L1 cache 851 that is specific for each array or
20 shared between the arrays. The cache can be configured as a data cache, an instruction cache, or a single cache that is partitioned to contain data and instructions in different partitions.

In some embodiments, graphics pipeline 820 includes tessellation components to perform hardware-accelerated tessellation of 3D objects. In some embodiments, a programmable hull
25 shader 811 configures the tessellation operations. A programmable domain shader 817 provides back-end evaluation of tessellation output. A tessellator 813 operates at the direction of hull shader 811 and contains special purpose logic to generate a set of detailed geometric objects based on a coarse geometric model that is provided as input to graphics pipeline 820. In some embodiments, if tessellation is not used, tessellation components 811, 813, 817 can be bypassed.

In some embodiments, complete geometric objects can be processed by a geometry shader
30 819 via one or more threads dispatched to execution units 852A, 852B, or can proceed directly to the clipper 829. In some embodiments, the geometry shader operates on entire geometric objects, rather than vertices or patches of vertices as in previous stages of the graphics pipeline. If the tessellation is disabled the geometry shader 819 receives input from the vertex shader 807. In some embodiments, geometry shader 819 is programmable by a geometry shader program to
35 perform geometry tessellation if the tessellation units are disabled.

Before rasterization, a clipper 829 processes vertex data. The clipper 829 may be a fixed function clipper or a programmable clipper having clipping and geometry shader functions. In some embodiments, a rasterizer and depth test component 873 in the render output pipeline 870 dispatches pixel shaders to convert the geometric objects into their per pixel representations. In some embodiments, pixel shader logic is included in thread execution logic 850. In some embodiments, an application can bypass the rasterizer 873 and access un-rasterized vertex data via a stream out unit 823.

The graphics processor 800 has an interconnect bus, interconnect fabric, or some other interconnect mechanism that allows data and message passing amongst the major components of the processor. In some embodiments, execution units 852A, 852B and associated cache(s) 851, texture and media sampler 854, and texture/sampler cache 858 interconnect via a data port 856 to perform memory access and communicate with render output pipeline components of the processor. In some embodiments, sampler 854, caches 851, 858 and execution units 852A, 852B each have separate memory access paths.

In some embodiments, render output pipeline 870 contains a rasterizer and depth test component 873 that converts vertex-based objects into an associated pixel-based representation. In some embodiments, the rasterizer logic includes a windower/masker unit to perform fixed function triangle and line rasterization. An associated render cache 878 and depth cache 879 are also available in some embodiments. A pixel operations component 877 performs pixel-based operations on the data, though in some instances, pixel operations associated with 2D operations (e.g. bit block image transfers with blending) are performed by the 2D engine 841, or substituted at display time by the display controller 843 using overlay display planes. In some embodiments, a shared L3 cache 875 is available to all graphics components, allowing the sharing of data without the use of main system memory.

In some embodiments, graphics processor media pipeline 830 includes a media engine 837 and a video front end 834. In some embodiments, video front end 834 receives pipeline commands from the command streamer 803. In some embodiments, media pipeline 830 includes a separate command streamer. In some embodiments, video front-end 834 processes media commands before sending the command to the media engine 837. In some embodiments, media engine 337 includes thread spawning functionality to spawn threads for dispatch to thread execution logic 850 via thread dispatcher 831.

In some embodiments, graphics processor 800 includes a display engine 840. In some embodiments, display engine 840 is external to processor 800 and couples with the graphics processor via the ring interconnect 802, or some other interconnect bus or fabric. In some embodiments, display engine 840 includes a 2D engine 841 and a display controller 843. In

some embodiments, display engine 840 contains special purpose logic capable of operating independently of the 3D pipeline. In some embodiments, display controller 843 couples with a display device (not shown), which may be a system integrated display device, as in a laptop computer, or an external display device attached via a display device connector.

5 In some embodiments, graphics pipeline 820 and media pipeline 830 are configurable to perform operations based on multiple graphics and media programming interfaces and are not specific to any one application programming interface (API). In some embodiments, driver software for the graphics processor translates API calls that are specific to a particular graphics or media library into commands that can be processed by the graphics processor. In some
10 embodiments, support is provided for the Open Graphics Library (OpenGL) and Open Computing Language (OpenCL) from the Khronos Group, the Direct3D library from the Microsoft Corporation, or support may be provided to both OpenGL and D3D. Support may also be provided for the Open Source Computer Vision Library (OpenCV). A future API with a compatible 3D pipeline would also be supported if a mapping can be made from the pipeline of
15 the future API to the pipeline of the graphics processor.

Graphics Pipeline Programming

Figure 9A is a block diagram illustrating a graphics processor command format 900 according to some embodiments. **Figure 9B** is a block diagram illustrating a graphics processor command sequence 910 according to an embodiment. The solid lined boxes in **Fig. 9A** illustrate
20 the components that are generally included in a graphics command while the dashed lines include components that are optional or that are only included in a sub-set of the graphics commands. The exemplary graphics processor command format 900 of **Fig. 9A** includes data fields to identify a target client 902 of the command, a command operation code (opcode) 904, and the relevant data 906 for the command. A sub-opcode 905 and a command size 908 are also
25 included in some commands.

In some embodiments, client 902 specifies the client unit of the graphics device that processes the command data. In some embodiments, a graphics processor command parser examines the client field of each command to condition the further processing of the command and route the command data to the appropriate client unit. In some embodiments, the graphics
30 processor client units include a memory interface unit, a render unit, a 2D unit, a 3D unit, and a media unit. Each client unit has a corresponding processing pipeline that processes the commands. Once the command is received by the client unit, the client unit reads the opcode 904 and, if present, sub-opcode 905 to determine the operation to perform. The client unit performs the command using information in data field 906. For some commands an explicit
35 command size 908 is expected to specify the size of the command. In some embodiments, the

command parser automatically determines the size of at least some of the commands based on the command opcode. In some embodiments commands are aligned via multiples of a double word.

The flow diagram in **Fig. 9B** shows an exemplary graphics processor command sequence

5 910. In some embodiments, software or firmware of a data processing system that features an embodiment of a graphics processor uses a version of the command sequence shown to set up, execute, and terminate a set of graphics operations. A sample command sequence is shown and described for purposes of example only as embodiments are not limited to these specific commands or to this command sequence. Moreover, the commands may be issued as batch of
10 commands in a command sequence, such that the graphics processor will process the sequence of commands in at least partially concurrence.

In some embodiments, the graphics processor command sequence 910 may begin with a pipeline flush command 912 to cause any active graphics pipeline to complete the currently pending commands for the pipeline. In some embodiments, the 3D pipeline 922 and the media
15 pipeline 924 do not operate concurrently. The pipeline flush is performed to cause the active graphics pipeline to complete any pending commands. In response to a pipeline flush, the command parser for the graphics processor will pause command processing until the active drawing engines complete pending operations and the relevant read caches are invalidated. Optionally, any data in the render cache that is marked 'dirty' can be flushed to memory. In
20 some embodiments, pipeline flush command 912 can be used for pipeline synchronization or before placing the graphics processor into a low power state.

In some embodiments, a pipeline select command 913 is used when a command sequence requires the graphics processor to explicitly switch between pipelines. In some embodiments, a pipeline select command 913 is required only once within an execution context before issuing
25 pipeline commands unless the context is to issue commands for both pipelines. In some embodiments, a pipeline flush command is 912 is required immediately before a pipeline switch via the pipeline select command 913.

In some embodiments, a pipeline control command 914 configures a graphics pipeline for operation and is used to program the 3D pipeline 922 and the media pipeline 924. In some
30 embodiments, pipeline control command 914 configures the pipeline state for the active pipeline. In one embodiment, the pipeline control command 914 is used for pipeline synchronization and to clear data from one or more cache memories within the active pipeline before processing a batch of commands.

In some embodiments, return buffer state commands 916 are used to configure a set of
35 return buffers for the respective pipelines to write data. Some pipeline operations require the

allocation, selection, or configuration of one or more return buffers into which the operations write intermediate data during processing. In some embodiments, the graphics processor also uses one or more return buffers to store output data and to perform cross thread communication. In some embodiments, the return buffer state 916 includes selecting the size and number of
5 return buffers to use for a set of pipeline operations.

The remaining commands in the command sequence differ based on the active pipeline for operations. Based on a pipeline determination 920, the command sequence is tailored to the 3D pipeline 922 beginning with the 3D pipeline state 930, or the media pipeline 924 beginning at the media pipeline state 940.

10 The commands for the 3D pipeline state 930 include 3D state setting commands for vertex buffer state, vertex element state, constant color state, depth buffer state, and other state variables that are to be configured before 3D primitive commands are processed. The values of these commands are determined at least in part based the particular 3D API in use. In some embodiments, 3D pipeline state 930 commands are also able to selectively disable or bypass
15 certain pipeline elements if those elements will not be used.

In some embodiments, 3D primitive 932 command is used to submit 3D primitives to be processed by the 3D pipeline. Commands and associated parameters that are passed to the graphics processor via the 3D primitive 932 command are forwarded to the vertex fetch function in the graphics pipeline. The vertex fetch function uses the 3D primitive 932 command data to
20 generate vertex data structures. The vertex data structures are stored in one or more return buffers. In some embodiments, 3D primitive 932 command is used to perform vertex operations on 3D primitives via vertex shaders. To process vertex shaders, 3D pipeline 922 dispatches shader execution threads to graphics processor execution units.

In some embodiments, 3D pipeline 922 is triggered via an execute 934 command or event.
25 In some embodiments, a register write triggers command execution. In some embodiments execution is triggered via a 'go' or 'kick' command in the command sequence. In one embodiment command execution is triggered using a pipeline synchronization command to flush the command sequence through the graphics pipeline. The 3D pipeline will perform geometry processing for the 3D primitives. Once operations are complete, the resulting geometric objects
30 are rasterized and the pixel engine colors the resulting pixels. Additional commands to control pixel shading and pixel back end operations may also be included for those operations.

In some embodiments, the graphics processor command sequence 910 follows the media pipeline 924 path when performing media operations. In general, the specific use and manner of programming for the media pipeline 924 depends on the media or compute operations to be
35 performed. Specific media decode operations may be offloaded to the media pipeline during

media decode. In some embodiments, the media pipeline can also be bypassed and media decode can be performed in whole or in part using resources provided by one or more general purpose processing cores. In one embodiment, the media pipeline also includes elements for general-purpose graphics processor unit (GPGPU) operations, where the graphics processor is used to perform SIMD vector operations using computational shader programs that are not explicitly related to the rendering of graphics primitives.

In some embodiments, media pipeline 924 is configured in a similar manner as the 3D pipeline 922. A set of media pipeline state commands 940 are dispatched or placed into in a command queue before the media object commands 942. In some embodiments, media pipeline state commands 940 include data to configure the media pipeline elements that will be used to process the media objects. This includes data to configure the video decode and video encode logic within the media pipeline, such as encode or decode format. In some embodiments, media pipeline state commands 940 also support the use one or more pointers to “indirect” state elements that contain a batch of state settings.

In some embodiments, media object commands 942 supply pointers to media objects for processing by the media pipeline. The media objects include memory buffers containing video data to be processed. In some embodiments, all media pipeline states must be valid before issuing a media object command 942. Once the pipeline state is configured and media object commands 942 are queued, the media pipeline 924 is triggered via an execute command 944 or an equivalent execute event (e.g., register write). Output from media pipeline 924 may then be post processed by operations provided by the 3D pipeline 922 or the media pipeline 924. In some embodiments, GPGPU operations are configured and executed in a similar manner as media operations.

Graphics Software Architecture

Figure 10 illustrates exemplary graphics software architecture for a data processing system 1000 according to some embodiments. In some embodiments, software architecture includes a 3D graphics application 1010, an operating system 1020, and at least one processor 1030. In some embodiments, processor 1030 includes a graphics processor 1032 and one or more general-purpose processor core(s) 1034. The graphics application 1010 and operating system 1020 each execute in the system memory 1050 of the data processing system.

In some embodiments, 3D graphics application 1010 contains one or more shader programs including shader instructions 1012. The shader language instructions may be in a high-level shader language, such as the High Level Shader Language (HLSL) or the OpenGL Shader Language (GLSL). The application also includes executable instructions 1014 in a

machine language suitable for execution by the general-purpose processor core 1034. The application also includes graphics objects 1016 defined by vertex data.

In some embodiments, operating system 1020 is a Microsoft® Windows® operating system from the Microsoft Corporation, a proprietary UNIX-like operating system, or an open source UNIX-like operating system using a variant of the Linux kernel. When the Direct3D API is in use, the operating system 1020 uses a front-end shader compiler 1024 to compile any shader instructions 1012 in HLSL into a lower-level shader language. The compilation may be a just-in-time (JIT) compilation or the application can perform shader pre-compilation. In some embodiments, high-level shaders are compiled into low-level shaders during the compilation of the 3D graphics application 1010.

In some embodiments, user mode graphics driver 1026 contains a back-end shader compiler 1027 to convert the shader instructions 1012 into a hardware specific representation. When the OpenGL API is in use, shader instructions 1012 in the GLSL high-level language are passed to a user mode graphics driver 1026 for compilation. In some embodiments, user mode graphics driver 1026 uses operating system kernel mode functions 1028 to communicate with a kernel mode graphics driver 1029. In some embodiments, kernel mode graphics driver 1029 communicates with graphics processor 1032 to dispatch commands and instructions.

IP Core Implementations

One or more aspects of at least one embodiment may be implemented by representative code stored on a machine-readable medium which represents and/or defines logic within an integrated circuit such as a processor. For example, the machine-readable medium may include instructions which represent various logic within the processor. When read by a machine, the instructions may cause the machine to fabricate the logic to perform the techniques described herein. Such representations, known as “IP cores,” are reusable units of logic for an integrated circuit that may be stored on a tangible, machine-readable medium as a hardware model that describes the structure of the integrated circuit. The hardware model may be supplied to various customers or manufacturing facilities, which load the hardware model on fabrication machines that manufacture the integrated circuit. The integrated circuit may be fabricated such that the circuit performs operations described in association with any of the embodiments described herein.

Figure 11 is a block diagram illustrating an IP core development system 1100 that may be used to manufacture an integrated circuit to perform operations according to an embodiment. The IP core development system 1100 may be used to generate modular, re-usable designs that can be incorporated into a larger design or used to construct an entire integrated circuit (e.g., an SOC integrated circuit). A design facility 1130 can generate a software simulation 1110 of an IP

core design in a high level programming language (e.g., C/C++). The software simulation 1110 can be used to design, test, and verify the behavior of the IP core. A register transfer level (RTL) design can then be created or synthesized from the simulation model 1100. The RTL design 1115 is an abstraction of the behavior of the integrated circuit that models the flow of digital signals between hardware registers, including the associated logic performed using the modeled digital signals. In addition to an RTL design 1115, lower-level designs at the logic level or transistor level may also be created, designed, or synthesized. Thus, the particular details of the initial design and simulation may vary.

The RTL design 1115 or equivalent may be further synthesized by the design facility into a hardware model 1120, which may be in a hardware description language (HDL), or some other representation of physical design data. The HDL may be further simulated or tested to verify the IP core design. The IP core design can be stored for delivery to a 3rd party fabrication facility 1165 using non-volatile memory 1140 (e.g., hard disk, flash memory, or any non-volatile storage medium). Alternatively, the IP core design may be transmitted (e.g., via the Internet) over a wired connection 1150 or wireless connection 1160. The fabrication facility 1165 may then fabricate an integrated circuit that is based at least in part on the IP core design. The fabricated integrated circuit can be configured to perform operations in accordance with at least one embodiment described herein.

Figure 12 is a block diagram illustrating an exemplary system on a chip integrated circuit 1200 that may be fabricated using one or more IP cores, according to an embodiment. The exemplary integrated circuit includes one or more application processors 1205 (e.g., CPUs), at least one graphics processor 1210, and may additionally include an image processor 1215 and/or a video processor 1220, any of which may be a modular IP core from the same or multiple different design facilities. The integrated circuit includes peripheral or bus logic including a USB controller 1225, UART controller 1230, an SPI/SDIO controller 1235, and an I²S/I²C controller 1240. Additionally, the integrated circuit can include a display device 1245 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1250 and a mobile industry processor interface (MIPI) display interface 1255. Storage may be provided by a flash memory subsystem 1260 including flash memory and a flash memory controller. Memory interface may be provided via a memory controller 1265 for access to SDRAM or SRAM memory devices. Some integrated circuits additionally include an embedded security engine 1270.

Additionally, other logic and circuits may be included in the processor of integrated circuit 1200, including additional graphics processors/cores, peripheral interface controllers, or general purpose processor cores.

Figure 13 illustrates a computing device 1300 employing a runtime metrics-based workload management mechanism 1300 according to one embodiment. Computing device 1300 (e.g., mobile computer, laptop computer, desktop computer, etc.) may be the same as data processing system 100 of **Figure 1** and accordingly, for brevity and ease of understanding, many of the details stated above with reference to **Figures 1-12** are not further discussed or repeated hereafter. For example, computing device 1300 may include a mobile computer (e.g., smartphone, tablet computer, laptops, game consoles, portable workstations, smart glasses and other smart wearable devices, etc.) serving as a host machine for hosting runtime metrics-based workload management mechanism (“workload mechanism”) 1310.

Workload mechanism 1310 may include any number and type of components for facilitating measurement and use of metrics relating to draws (“work units”) for efficient performance of 3D applications (“workloads”), wherein each workload may contain any number and type of work units. As previously described, throughout this document, terms like “work unit” and “draw” may be referenced interchangeably and similarly, terms like “workload” and “application” may be referenced interchangeably according to one embodiment. Throughout the document, the term “user” may be interchangeably referred to as “viewer”, “observer”, “person”, “individual”, “end-user”, and/or the like. It is to be noted that throughout this document, terms like “graphics domain” may be referenced interchangeably with “graphics processing unit” or simply “GPU” and similarly, “CPU domain” or “host domain” may be referenced interchangeably with “computer processing unit” or simply “CPU”.

Computing device 1300 may include any number and type of communication devices, such as large computing systems, such as server computers, desktop computers, etc., and may further include set-top boxes (e.g., Internet-based cable television set-top boxes, etc.), global positioning system (GPS)-based devices, etc. Computing device 1300 may include mobile computing devices serving as communication devices, such as cellular phones including smartphones, personal digital assistants (PDAs), tablet computers, laptop computers, e-readers, smart televisions, television platforms, wearable devices (e.g., glasses, watches, bracelets, smartcards, jewelry, clothing items, etc.), media players, etc. For example, in one embodiment, computing device 1300 may include a mobile computing device employing an integrated circuit (“IC”), such as system on a chip (“SoC” or “SOC”), integrating various hardware and/or software components of computing device 1300 on a single chip.

As illustrated, in one embodiment, in addition to employing workload mechanism 1310, computing device 1300 may further include any number and type of hardware components and/or software components, such as (but not limited to) GPU 1314 (having driver logic 1316), CPU 1312, memory 1308, network devices, drivers, or the like, as well as input/output (I/O)

sources 1304, such as touchscreens, touch panels, touch pads, virtual or regular keyboards, virtual or regular mice, ports, connectors, etc. Computing device 1300 may include operating system (OS) 1306 serving as an interface between hardware and/or physical resources of the computer device 1300 and a user. It is contemplated that CPU 1312 may include one or
5 processors, such as processor(s) 102 of **Figure 1**, while GPU 1314 may include one or more graphics processors, such as graphics processor(s) 108 of **Figure 1**. In one embodiment and as will be further described with reference to the subsequent figures, workload mechanism 1310 may be in communication with its host driver logic 1316 which cooperates with GPU 1314 to facilitate any number and type of tasks facilitating generation and rendering of virtual 3D images
10 as is described through this document.

It is to be noted that terms like “node”, “computing node”, “server”, “server device”, “cloud computer”, “cloud server”, “cloud server computer”, “machine”, “host machine”, “device”, “computing device”, “computer”, “computing system”, and the like, may be used interchangeably throughout this document. It is to be further noted that terms like “application”,
15 “software application”, “program”, “software program”, “package”, “software package”, and the like, may be used interchangeably throughout this document. Also, terms like “job”, “input”, “request”, “message”, and the like, may be used interchangeably throughout this document.

It is contemplated and as further described with reference to **Figures 1-12**, some processes of the graphics pipeline as described above are implemented in software, while the rest are
20 implemented in hardware. A graphics pipeline may be implemented in a graphics coprocessor design, where CPU 1312 is designed to work with GPU 1314 which may be included in or co-located with CPU 1312. In one embodiment, GPU 1314 may employ any number and type of conventional software and hardware logic to perform the conventional functions relating to graphics rendering as well as novel software and hardware logic to execute any number and type
25 of instructions, such as instructions 121 of **Figure 1**, to perform the various novel functions of workload mechanism 1310 as disclosed throughout this document.

As aforementioned, memory 1308 may include a random access memory (RAM) comprising application database having object information. A memory controller hub, such as memory controller hub 116 of **Figure 1**, may access data in the RAM and forward it to GPU
30 1314 for graphics pipeline processing. RAM may include double data rate RAM (DDR RAM), extended data output RAM (EDO RAM), etc. CPU 1312 interacts with a hardware graphics pipeline, as illustrated with reference to **Figure 3**, to share graphics pipelining functionality. Processed data is stored in a buffer in the hardware graphics pipeline, and state information is stored in memory 1308. The resulting image is then transferred to I/O sources 1304, such as a
35 display component, such as display device 320 of **Figure 3**, for displaying of the image. It is

contemplated that the display device may be of various types, such as Cathode Ray Tube (CRT), Thin Film Transistor (TFT), Liquid Crystal Display (LCD), Organic Light Emitting Diode (OLED) array, etc., to display information to a user.

Memory 1308 may comprise a pre-allocated region of a buffer (e.g., frame buffer);
5 however, it should be understood by one of ordinary skill in the art that the embodiments are not so limited, and that any memory accessible to the lower graphics pipeline may be used. Computing device 1300 may further include input/output (I/O) control hub (ICH) 130 as referenced in **Figure 1**, one or more I/O sources 1304, etc.

CPU 1312 may include one or more processors to execute instructions in order to perform
10 whatever software routines the computing system implements. The instructions frequently involve some sort of operation performed upon data. Both data and instructions may be stored in system memory 1308 and any associated cache. Cache is typically designed to have shorter latency times than system memory 1308; for example, cache might be integrated onto the same silicon chip(s) as the processor(s) and/or constructed with faster static RAM (SRAM) cells whilst
15 the system memory 1308 might be constructed with slower dynamic RAM (DRAM) cells. By tending to store more frequently used instructions and data in the cache as opposed to the system memory 1308, the overall performance efficiency of computing device 1300 improves. It is contemplated that in some embodiments, GPU 1314 may exist as part of CPU 1312 (such as part of a physical CPU package) in which case, memory 1308 may be shared by CPU 1312 and GPU
20 1314 or kept separated.

System memory 1308 may be made available to other components within the computing device 1300. For example, any data (e.g., input graphics data) received from various interfaces to the computing device 1300 (e.g., keyboard and mouse, printer port, Local Area Network (LAN) port, modem port, etc.) or retrieved from an internal storage element of the computer
25 device 1300 (e.g., hard disk drive) are often temporarily queued into system memory 1308 prior to their being operated upon by the one or more processor(s) in the implementation of a software program. Similarly, data that a software program determines should be sent from the computing device 1300 to an outside entity through one of the computing system interfaces, or stored into an internal storage element, is often temporarily queued in system memory 1308 prior to its
30 being transmitted or stored.

Further, for example, an ICH, such as ICH 130 of **Figure 1**, may be used for ensuring that such data is properly passed between the system memory 1308 and its appropriate corresponding computing system interface (and internal storage device if the computing system is so designed) and may have bi-directional point-to-point links between itself and the observed I/O
35 sources/devices 1304. Similarly, an MCH, such as MCH 116 of **Figure 1**, may be used for

managing the various contending requests for system memory 1308 accesses amongst CPU 1312 and GPU 1314, interfaces and internal storage elements that may proximately arise in time with respect to one another.

I/O sources 1304 may include one or more I/O devices that are implemented for
5 transferring data to and/or from computing device 1300 (e.g., a networking adapter); or, for a large scale non-volatile storage within computing device 1300 (e.g., hard disk drive). User input device, including alphanumeric and other keys, may be used to communicate information and command selections to GPU 1314. Another type of user input device is cursor control, such as a mouse, a trackball, a touchscreen, a touchpad, or cursor direction keys to communicate direction
10 information and command selections to GPU 1314 and to control cursor movement on the display device. Camera and microphone arrays of computer device 1300 may be employed to observe gestures, record audio and video and to receive and transmit visual and audio commands.

Computing device 1300 may further include network interface(s) to provide access to a
15 network, such as a LAN, a wide area network (WAN), a metropolitan area network (MAN), a personal area network (PAN), Bluetooth, a cloud network, a mobile network (e.g., 3rd Generation (3G), etc.), an intranet, the Internet, etc. Network interface(s) may include, for example, a wireless network interface having antenna, which may represent one or more antenna(e). Network interface(s) may also include, for example, a wired network interface to communicate
20 with remote devices via network cable, which may be, for example, an Ethernet cable, a coaxial cable, a fiber optic cable, a serial cable, or a parallel cable.

Network interface(s) may provide access to a LAN, for example, by conforming to IEEE 802.11b and/or IEEE 802.11g standards, and/or the wireless network interface may provide access to a personal area network, for example, by conforming to Bluetooth standards. Other
25 wireless network interfaces and/or protocols, including previous and subsequent versions of the standards, may also be supported. In addition to, or instead of, communication via the wireless LAN standards, network interface(s) may provide wireless communication using, for example, Time Division, Multiple Access (TDMA) protocols, Global Systems for Mobile Communications (GSM) protocols, Code Division, Multiple Access (CDMA) protocols, and/or
30 any other type of wireless communications protocols.

Network interface(s) may include one or more communication interfaces, such as a modem, a network interface card, or other well-known interface devices, such as those used for coupling to the Ethernet, token ring, or other types of physical wired or wireless attachments for purposes of providing a communication link to support a LAN or a WAN, for example. In this
35 manner, the computer system may also be coupled to a number of peripheral devices, clients,

control surfaces, consoles, or servers via a conventional network infrastructure, including an Intranet or the Internet, for example.

It is to be appreciated that a lesser or more equipped system than the example described above may be preferred for certain implementations. Therefore, the configuration of computing device 1300 may vary from implementation to implementation depending upon numerous factors, such as price constraints, performance requirements, technological improvements, or other circumstances. Examples of the electronic device or computer system 1300 may include (without limitation) a mobile device, a personal digital assistant, a mobile computing device, a smartphone, a cellular telephone, a handset, a one-way pager, a two-way pager, a messaging device, a computer, a personal computer (PC), a desktop computer, a laptop computer, a notebook computer, a handheld computer, a tablet computer, a server, a server array or server farm, a web server, a network server, an Internet server, a work station, a mini-computer, a main frame computer, a supercomputer, a network appliance, a web appliance, a distributed computing system, multiprocessor systems, processor-based systems, consumer electronics, programmable consumer electronics, television, digital television, set top box, wireless access point, base station, subscriber station, mobile subscriber center, radio network controller, router, hub, gateway, bridge, switch, machine, or combinations thereof.

Embodiments may be implemented as any or a combination of: one or more microchips or integrated circuits interconnected using a parentboard, hardwired logic, software stored by a memory device and executed by a microprocessor, firmware, an application specific integrated circuit (ASIC), and/or a field programmable gate array (FPGA). The term "logic" may include, by way of example, software or hardware and/or combinations of software and hardware.

Embodiments may be provided, for example, as a computer program product which may include one or more machine-readable media having stored thereon machine-executable instructions that, when executed by one or more machines such as a computer, network of computers, or other electronic devices, may result in the one or more machines carrying out operations in accordance with embodiments described herein. A machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs, RAMs, EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electrically Erasable Programmable Read Only Memories), magnetic or optical cards, flash memory, or other type of media/machine-readable medium suitable for storing machine-executable instructions.

Moreover, embodiments may be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of one or more data signals embodied in and/or modulated by a carrier

wave or other propagation medium via a communication link (e.g., a modem and/or network connection).

Figure 14 illustrates a runtime metrics-based workload management mechanism 1310 according to one embodiment. In one embodiment, workload mechanism 1310 may include any number and type of components to perform various tasks relating to intelligent, dynamic, and real-time measurement of metrics relating to draws and using, in real-time, such metrics for efficient performing of workloads at computing device 1300. For example and in one embodiment, workload mechanism 1310 may include any number and type of components, such as (without limitation): reception/detection logic 1401; metrics collector 1403; evaluation logic 1405; configuration manager 1407; execution/application logic 1409; and communication/compatibility logic 1411. Computing device 1300 is further in communication with database 1430 (e.g., Cloud storage, non-Cloud storage, etc.).

As an initial matter, it is contemplated and to be noted that, in one embodiment, workload mechanism 1310 may be hosted by graphics driver logic, such as driver logic 1316, of a GPU, such as GPU 1314, of **Figure 13**, while, in another embodiment, workload mechanism 1310 may not be hosted by a GPU and that it may be hosted by an operating system, such as operating system 1306 of **Figure 13**. Similarly, in yet another embodiment, various functionalities and/or components of workload mechanism 1310 may be provided as one or more hardware components of computing device 1300.

As aforementioned, an application (e.g., game application) may include any number and types of frames having any number and type of draws, where each draw may represent a portion (e.g., desk in classroom, cloud in sky, nose on face, etc.) of an image (e.g., classroom, sky, face, etc.) represented by a frame. In one embodiment, size and content of a draw, like its corresponding frame, may be predetermined by the application. It is contemplated that embodiments are not limited to any particular number or type of applications, frames, draws, etc., and similarly, embodiments are not limited to any particular number and type of image or size of image portions. It is contemplated that an application can have any number of frames and each frame may have any number of draws and each draw may have any type of image content.

Embodiments provide for a technique where metrics, such as various characteristics relating to a draw that reflect where the draw may be bottlenecking with regard to the resources of computing device 1300, relating to a draw may be detected and such metrics may then be used to predict an optimal machine configuration for that draw to accordingly configure any number and type of computing resources (e.g., hardware configuration) when the same or similar draws (e.g., shader, state, etc.) that are subsequently detected to be processed and executed in the future, which improves performance and efficiency on computing device 1300.

Computer devices, such as computing device 1300, are becoming increasingly customizable, such as in terms of how computing device 1300 is configured to utilize computing resources during workload execution. For example, various computing resources, such as cache, of computing device 1300 may be configured differently to allow or exclude certain data structures from being resident in the cache. For example, portions of the device may be configured to run at faster or slower frequencies depending on workload demands upon different computing resources, where, for example, computing elements that are awoken or asleep may be changed in order to reach peak efficiency.

In one embodiment, reception/detection logic 1401 may be used to detect, in runtime, each draw of each frame relating to an application that is being processed or running at computing device 1300. Upon detection of the draw, reception/detection logic 1401 may communicate with metrics collector 1403 to indicate the draw has been detected and then continue to detect other draws of the same and other frames of the application.

Once the indication has been made, in one embodiment, metrics collector (also referred to as “graphics characteristics collector”) 1403 may begin to collect one or more metrics regarding the draw. For example, as aforementioned, metrics may refer to or include various characteristics relating to the draw, such as any information that indicates where the draw suffers from bottlenecking during its processing in terms of using various resources, such as time, cache, hardware components, etc., and thus configurable device characteristics may include (without limitation): cache size, cache policies, frequencies, sleep state, etc. For example, such metrics, as collected by metrics collector 1403, may include (without limitation) 1) types of and frequencies at which various hardware components (e.g., CPU, GPU, GPU slice, GPU fixed function, double data rate (DDR) device, etc.) were used, 2) how much shading work it did in reading texture in terms of lighting computation is performed (such as, in some cases, there may be a great deal of computation work to figure out the lighting, but the work involved in reading of the actual texture value may be a small part of the process which may mean, upon evaluation by evaluation logic 1405, shutting down part of the chip because samplers, textures, etc., are not needed and just the computation is needed for the same or similar draw when it appears later in the process, 3) cache amount and policies, cache partitioning/distribution, 4) type of compression format applied, 5) whether depth surfaces were resolved at sample time, and/or the like.

In one embodiment, metrics collector 1403 may be used to identify various metrics relating to draws to be collected and used for subsequent draws, where these metrics are then used by other components, such as evaluation logic 1405, configuration manager 1407, execution/application logic 1409, etc., to evaluate the metrics and predictively generate a configuration setup for system resources (e.g., cache, etc.) to be applied by GPU, such as GPU

1314 of **Figure 13**, and/or other components of computing device 1300 to any subsequent draws that may be the same or similar to the previous draws for which the metrics had been collected.

Further, in one embodiment, metrics collector 1403 may not collect all the metrics data relating to a draw in a single run and thus with each run, metric collector 1403 may collect some
5 of the metrics each time the draw is executed and save those metrics at database (also referred to as “work unit optimal configuration database”) 1430 and continue to collect additional metrics on subsequent runs each time the draw is subsequently executed in the same frame or other subsequent frames. In one embodiment, each collection of metrics relating to the draw is stored at database 1430 to allow for the availability of more and more metrics for evaluation logic 1405
10 to evaluate and process.

In one embodiment, upon collecting all metrics relating to the draw by metrics collector 1403, whether they be collected in one or more runs, metrics collector 1403 may begin to collect the oldest metrics again to overwrite the stored old metrics to be sure that all metrics stay as current and updated and stored at database 1430 since any characteristics of a draw may change
15 over time, such as based on how much screen space the draw may occupy depending on the user interaction.

In one embodiment, evaluation logic 1405 may access the collected metrics at database 1430 and evaluate these metrics to help configuration manager (also referred to as “hardware configuration manager”) 1407 to determine an optimal configuration plan for future draws. For
20 example and in one embodiment, evaluation logic 1405 may consider any number and type of parameters relating to various characteristics obtained from the collected metrics to determine prior to forwarding the information to configuration manager 1407 for further processing.

For example, in some embodiments, evaluation logic 1405 may be used to determine those instances when it may not be feasible to significantly change the device configuration relating to
25 system hardware, resources, etc., such as putting certain hardware units to sleep may take more time than needed for the draw to be executed. Once determined, such information may then be forwarded from evaluation logic 1405 on to configuration manager 1407 to take this into consideration while factoring in the performance and power costs of any appropriate and feasible configuration changes along with reviewing any past/previous device configurations when
30 putting together an optimal configuration plan for subsequent draws.

For example and in one embodiment, any number and type of configuration parameters may be taken into consideration by evaluation logic 1405 and/or configuration manager 1407 in setting up appropriate, efficient, and feasible optimal configuration plans, where such configuration parameters may include (without limitation): 1) whether shader constraints are pre-
35 loaded or read by the GPU from memory at runtime; 2) types of and frequencies at which certain

hardware/firmware/software components (e.g., GPU, GPU slice, GPU fixed function, CPU, DDR devices, etc.) are access and used; 3) cache partitioning (e.g., statically allocating certain portions of a cache to certain resource types; 4) type and amount of cached accessed and used; 5) whether to resolve depth surfaces at sample time; 6) type of compression format to use for data; 5 7) whether to unroll loops in shaders; and 8) dispatch width needed to execute a shader on the GPU, and/or the like.

As with the metrics and updated metrics, in one embodiment, any optimal configuration plans, configuration parameters, analysis, evaluation results, etc., may also be stored at database 1430 for future reference and further processing. In one embodiment, configuration manager 10 1407 may access and process any metrics collected by metrics collector 1403, analysis/evaluation results as measured by evaluation logic 1405, etc., to calculate an optimal configuration for a given draw that is executed and this optimal configuration may be executed or applied by execution/application logic 1409 to one or more subsequent draws that are similar to the previously-executed draw and this process may continue as information about the draws is 15 continuously modified, in runtime, and any relevant metrics are continuously collected in runtime. Each time a new or modified optimal configuration plan relating to a draw is developed by configuration manager 1407, the configuration plan is then forwarded on to execution/application logic 1409 for its execution and application for processing any relevant subsequent draws in the same or other frames of an application. In one embodiment, 20 configuration manager 1407 may post process the command buffer and set the draw-related optimal configuration plan at the time the command buffer is submitted.

Computing device 1300 may be in communication with one or more repositories or databases, such as database 1430 having Cloud-based database, non-Cloud-based databases, etc., to store and maintain any amount and type of data (e.g., real-time sensory input data, historical 25 contents, metadata, resources, policies, criteria, rules and regulations, upgrades, etc.). Similarly, as aforementioned, computing device 1300 may be in communication with any number and type of other computing devices over a communication medium, such as one or more networks including (without limitation) Cloud network, the Internet, intranet, Internet of Things (“IoT”), proximity network, and Bluetooth, etc. It is contemplated that embodiments are not limited to 30 any particular number or type of communication medium or networks.

As previously described, in one embodiment, a work unit may be referred to as or include a single draw; however, in another embodiment, a work unit may be a smaller draw or a larger draw (e.g., a collection of small draws exhibiting the same workload characteristics, such as a geometry pass). Further, for example, it is contemplated that in case of DirectX® 12 or simply 35 DX12, a work unit may be a pipeline state object or a batch buffer. It is to be noted again that

throughout this document, “work unit” and “draw” are referenced interchangeably and similarly, “workload” and “application” are referenced interchangeably.

Communication/compatibility logic 1411 may be used to facilitate dynamic communication and compatibility between one or more computing devices, such as computing device 1300 of **Figure 13**, and any number and type of other computing devices (such as mobile computing device, desktop computer, server computing device, etc.), processing devices (such as CPUs, GPUs, etc.), capturing/sensing/detecting devices (such as capturing/sensing components including cameras, depth sensing cameras, camera sensors, RGB sensors, microphones, etc.), display devices (such as output components including display screens, display areas, display projectors, etc.), user/context-awareness components and/or identification/verification sensors/devices (such as biometric sensors/detectors, scanners, etc.), memory or storage devices, databases, and/or data sources (such as data storage devices, hard drives, solid-state drives, hard disks, memory cards or devices, memory circuits, etc.), communication channels or networks (e.g., Cloud network, the Internet, intranet, cellular network, proximity networks, such as Bluetooth, Bluetooth low energy (BLE), Bluetooth Smart, Wi-Fi proximity, Radio Frequency Identification (RFID), Near Field Communication (NFC), Body Area Network (BAN), etc.), wireless or wired communications and relevant protocols (e.g., Wi-Fi®, WiMAX, Ethernet, etc.), connectivity and location management techniques, software applications/websites, (e.g., social and/or business networking websites, etc., business applications, games and other entertainment applications, etc.), programming languages, etc., while ensuring compatibility with changing technologies, parameters, protocols, standards, etc.

Hardware Implementation

In one embodiment, one or more components of workload mechanism 1310, such as metrics collector 1403, etc., and other components, such as database 1430, etc., may be implemented as or in hardware, such as at GPU 1314 of **Figure 13**. For example and in one embodiment, hardware-based workload mechanism 1310 may perform any of the aforementioned tasks of workload mechanism 1310, such as to measure, create, maintain, etc., to generate and set any optimal configuration plans transparently to the graphics software and/or device driver, such as driver logic 1316 of **Figure 13**, running on the CPU, such as CPU 1312 of **Figure 13**. For example and in one embodiment, configuration manager 1407 of workload mechanism 1310 may be implemented as a hardware unit such that any optimal configuration plans may be set immediately prior to the GPU executing a draw, such as GPU 1314 of **Figure 13**. Further, in some embodiments, optimal configuration plans for various shader/state combinations may be measured offline, stored in cloud, and/or distributed to end-user, such as via one or more distribution mechanisms.

Throughout this document, terms like "logic", "component", "module", "framework", "engine", and the like, may be referenced interchangeably and include, by way of example, software, hardware, and/or any combination of software and hardware, such as firmware. Further, any use of a particular brand, word, term, phrase, name, and/or acronym, such as

5 "GPU", "GPU domain", "CPU", "CPU domain", "metrics", "characteristics" or "optimal configuration plan/state", "graphics characteristic collector", "work unit optimal configuration database", "hardware configuration manager", "workload", "application", "work unit", "draw", etc., should not be read to limit embodiments to software or devices that carry that label in products or in literature external to this document.

10 It is contemplated that any number and type of components may be added to and/or removed from workload mechanism 1310 to facilitate various embodiments including adding, removing, and/or enhancing certain features. For brevity, clarity, and ease of understanding of workload mechanism 1310, many of the standard and/or known components, such as those of a computing device, are not shown or discussed here. It is contemplated that embodiments, as

15 described herein, are not limited to any particular technology, topology, system, architecture, and/or standard and are dynamic enough to adopt and adapt to any future changes.

Figure 15 illustrates a method 1500 for facilitating metrics-based workload management according to one embodiment. Method 1500 may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as

20 instructions run on a processing device), or a combination thereof. In one embodiment, method 1500 may be performed by workload mechanism 1310 of **Figures 13-14**. The processes of method 1500 are illustrated in linear sequences for brevity and clarity in presentation; however, it is contemplated that any number of them can be performed in parallel, asynchronously, or in different orders. For brevity, many of the details discussed with reference to the preceding

25 **Figures 1-14** may not be discussed or repeated hereafter.

Method 1500 begins at block 1501 with an application/workload issuing a draw/work unit or rendering operation to graphics driver, where the draw is detected by reception/detection logic 1401 of workload mechanism 1310 of **Figure 14**. At block 1503, in one embodiment, metrics collector 1403 of **Figure 14** identifies metrics relating to the draw along with checking a

30 database, such as database 1430 of **Figure 14**, to access and identify any metrics already stored at the database. At block 1505, metrics collector 1403 of **Figure 14** collects the identified metrics during the execution of the draw.

In one embodiment, at block 1507, evaluation logic 1405 and configuration manager 1407 of **Figure 14** may evaluate the metrics relating to the draw and prepare an optimal configuration

35 plan for the same or similar subsequent draws, respectively, and further, a determination is made

as to whether to proceed with the optimal configuration plan or disregard any configuration plans due to, for example, insufficient metrics or unknown optimal configuration. At block 1509, in one embodiment, in case of an unknown optimal configuration plan or lack of any optimal configuration plan (such as because of insufficient metrics), method 1500 simply continues with the conventional or existing configuration setup of hardware and other resources, such as without having to perform any hardware configuration.

At block 1511, in one embodiment, if the optimal configuration plan of block 1507 is valid and applicable, it is selected and setup by configuration manager 1407 of **Figure 14** to be the plan for this particular draw and any same or similar draws that are detected in the future. At block 1513, in one embodiment, this optimal configuration plan is applied and executed for one or more of the same or similar subsequent draws as they are executed and issued to the GPU as facilitated by execution/application logic 1409 of **Figure 14**. At block 1515, upon each execution of the optimal configuration plan at the execution of each subsequent draw, metrics collector 1403 may be triggered to collect additional metrics to be stored at the database, where such additional metrics may include new metrics and/or updates to existing metrics stored at the database. Further, at block 1517, in one embodiment, any existing optimal configuration plans may be updated by configuration manager 1407 of **Figure 14** based on the newly collected additional metrics for future references and processing.

References to “one embodiment”, “an embodiment”, “example embodiment”, “various embodiments”, etc., indicate that the embodiment(s) so described may include particular features, structures, or characteristics, but not every embodiment necessarily includes the particular features, structures, or characteristics. Further, some embodiments may have some, all, or none of the features described for other embodiments.

In the foregoing specification, embodiments have been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of embodiments as set forth in the appended claims. The Specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

In the following description and claims, the term “coupled” along with its derivatives, may be used. “Coupled” is used to indicate that two or more elements co-operate or interact with each other, but they may or may not have intervening physical or electrical components between them.

As used in the claims, unless otherwise specified the use of the ordinal adjectives “first”, “second”, “third”, etc., to describe a common element, merely indicate that different instances of like elements are being referred to, and are not intended to imply that the elements so described

must be in a given sequence, either temporally, spatially, in ranking, or in any other manner.

The following clauses and/or examples pertain to further embodiments or examples. Specifics in the examples may be used anywhere in one or more embodiments. The various features of the different embodiments or examples may be variously combined with some features included and others excluded to suit a variety of different applications. Examples may include subject matter such as a method, means for performing acts of the method, at least one machine-readable medium including instructions that, when performed by a machine cause the machine to performs acts of the method, or of an apparatus or system for facilitating hybrid communication according to embodiments and examples described herein.

Some embodiments pertain to Example 1 that includes an apparatus to facilitate configuration of computing engines based on runtime workload measurements at computing devices, comprising: reception/detection logic to detect a work unit corresponding to a workload; metrics collector to collect metrics relating to the work unit, wherein the metrics to indicate one or more characteristics of the work unit; evaluation logic to evaluate the one or more characteristics based on one or more configuration parameters relating to computing resources; configuration manager to generate, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit; and execution/application logic to apply and execute the configuration plan at the apparatus upon execution of the one or more subsequent work units.

Example 2 includes the subject matter of Example 1, wherein the workload comprises a software application including a game application, wherein the workload further includes a plurality of frames representing a plurality of images.

Example 3 includes the subject matter of Example 1 or 2, wherein each frame comprises a plurality of work units, wherein each work unit includes a draw representing a portion of an image represented by a frame corresponding to the draw.

Example 4 includes the subject matter of Example 1, wherein the one or more characteristics relate to one or more processing traits relating to execution of the work unit, wherein the one or more characteristics relate to at least one of bottlenecking locations, bottlenecking reasons, cache usage, cache partitioning criteria, shader constraints, and amounts and types and frequencies of hardware allocation, and wherein the hardware allocation comprises allocation of hardware components including one or more of graphics processing unit (GPU), GPU slice, GPU fixed function, computer processing unit (CPU), and memory devices include double data rate (DDR) devices.

Example 5 includes the subject matter of Example 1 or 4, wherein the one or more configuration parameters are based on considerations associated with configuration or

reconfiguration of the computing resources comprising the hardware components, wherein the considerations include time consumption and power costs relating to one or more of the computing resources.

Example 6 includes the subject matter of Example 1, wherein the metrics collector is further to collect subsequent metrics relating to and upon execution of the one or more subsequent work units, wherein the subsequent metrics include new metrics or updated measurement of the metrics, and wherein the configuration manager is further to generate one or more subsequent configuration plans relating to the one or more subsequent work units based on the subsequent metrics.

Example 7 includes the subject matter of Example 1 or 7, wherein the metrics collector is further to store the metrics and the subsequent metrics at a database, wherein the metrics collector is further to retrieve the metrics and the subsequent metrics from the database, wherein the database is further to store the configuration plan and the one or more subsequent configuration plans, wherein the database includes a Cloud-based storage accessible over one or more networks.

Example 8 includes the subject matter of Example 1, wherein one or more of the reception/detection logic, metrics collector, evaluation logic, configuration manager, and execution/application logic are employed as one or more hardware units at the apparatus.

Some embodiments pertain to Example 9 that includes a method for facilitating configuration of computing engines based on runtime workload measurements at computing devices, comprising: detecting a work unit corresponding to a workload; collecting metrics relating to the work unit, wherein the metrics to indicate one or more characteristics of the work unit; evaluating the one or more characteristics based on one or more configuration parameters relating to computing resources; generating, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit; and applying and executing the configuration plan at a computing device upon execution of the one or more subsequent work units.

Example 10 includes the subject matter of Example 9, wherein the workload comprises a software application including a game application, wherein the workload further includes a plurality of frames representing a plurality of images.

Example 11 includes the subject matter of Example 10 or 11, wherein each frame comprises a plurality of work units, wherein each work unit includes a draw representing a portion of an image represented by a frame corresponding to the draw.

Example 12 includes the subject matter of Example 9, wherein the one or more

characteristics relate to one or more processing traits relating to execution of the work unit, wherein the one or more characteristics relate to at least one of bottlenecking locations, bottlenecking reasons, cache usage, cache partitioning criteria, shader constraints, and amounts and types and frequencies of hardware allocation, and wherein the hardware allocation comprises allocation of hardware components including one or more of graphics processing unit (GPU), GPU slice, GPU fixed function, computer processing unit (CPU), and memory devices include double data rate (DDR) devices.

Example 13 includes the subject matter of Example 9 or 12, wherein the one or more configuration parameters are based on considerations associated with configuration or reconfiguration of the computing resources comprising the hardware components, wherein the considerations include time consumption and power costs relating to one or more of the computing resources.

Example 14 includes the subject matter of Example 9, wherein collecting further comprises collecting subsequent metrics relating to and upon execution of the one or more subsequent work units, wherein the subsequent metrics include new metrics or updated measurement of the metrics, and wherein generating further comprises generating one or more subsequent configuration plans relating to the one or more subsequent work units based on the subsequent metrics.

Example 15 includes the subject matter of Example 9 or 14, further comprising storing the metrics and the subsequent metrics at a database, and retrieving the metrics and the subsequent metrics from the database, wherein the database is further to store the configuration plan and the one or more subsequent configuration plans, wherein the database includes a Cloud-based storage accessible over one or more networks.

Example 16 includes the subject matter of Example 9, wherein one or more of detecting, collecting, evaluating, generating, applying, storing, and receiving are performed via one or more hardware units at the computing device.

Example 17 includes at least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method or realize an apparatus as claimed in any preceding claims.

Example 18 includes at least one non-transitory or tangible machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method or realize an apparatus as claimed in any preceding claims.

Example 19 includes a system comprising a mechanism to implement or perform a method or realize an apparatus as claimed in any preceding claims.

Example 20 includes an apparatus comprising means to perform a method as claimed in

any preceding claims.

Example 21 includes a computing device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

Example 22 includes a communications device arranged to implement or perform a method
5 or realize an apparatus as claimed in any preceding claims.

Some embodiments pertain to Example 23 includes a system comprising a storage device having instructions, and a processor to execute the instructions to facilitate a mechanism to perform one or more operations comprising: detecting a work unit corresponding to a workload; collecting metrics relating to the work unit, wherein the metrics to indicate one or more
10 characteristics of the work unit; evaluating the one or more characteristics based on one or more configuration parameters relating to computing resources; generating, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit; and applying and executing the configuration plan at a computing device upon execution of the one or more subsequent work
15 units.

Example 24 includes the subject matter of Example 23, wherein the workload comprises a software application including a game application, wherein the workload further includes a plurality of frames representing a plurality of images.

Example 25 includes the subject matter of Example 24, wherein each frame comprises a
20 plurality of work units, wherein each work unit includes a draw representing a portion of an image represented by a frame corresponding to the draw.

Example 26 includes the subject matter of Example 23, wherein the one or more characteristics relate to one or more processing traits relating to execution of the work unit, wherein the one or more characteristics relate to at least one of bottlenecking locations,
25 bottlenecking reasons, cache usage, cache partitioning criteria, shader constraints, and amounts and types and frequencies of hardware allocation, and wherein the hardware allocation comprises allocation of hardware components including one or more of graphics processing unit (GPU), GPU slice, GPU fixed function, computer processing unit (CPU), and memory devices include double data rate (DDR) devices.

Example 27 includes the subject matter of Example 23, wherein the one or more configuration parameters are based on considerations associated with configuration or reconfiguration of the computing resources comprising the hardware components, wherein the considerations include time consumption and power costs relating to one or more of the computing resources.

Example 28 includes the subject matter of Example 23, wherein collecting further

comprises collecting subsequent metrics relating to and upon execution of the one or more subsequent work units, wherein the subsequent metrics include new metrics or updated measurement of the metrics, and wherein generating further comprises generating one or more subsequent configuration plans relating to the one or more subsequent work units based on the subsequent metrics.

Example 29 includes the subject matter of Example 23, wherein the one or more operations comprise storing the metrics and the subsequent metrics at a database, and retrieving the metrics and the subsequent metrics from the database, wherein the database is further to store the configuration plan and the one or more subsequent configuration plans, wherein the database includes a Cloud-based storage accessible over one or more networks.

Example 30 includes the subject matter of Example 23, wherein one or more of detecting, collecting, evaluating, generating, applying, storing, and receiving are performed via one or more hardware units at the computing device.

Some embodiments pertain to Example 31 includes an apparatus comprising: means for detecting a work unit corresponding to a workload; means for collecting metrics relating to the work unit, wherein the metrics to indicate one or more characteristics of the work unit; means for evaluating the one or more characteristics based on one or more configuration parameters relating to computing resources; means for generating, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit; and means for applying and executing the configuration plan at a computing device upon execution of the one or more subsequent work units.

Example 32 includes the subject matter of Example 31, wherein the workload comprises a software application including a game application, wherein the workload further includes a plurality of frames representing a plurality of images.

Example 33 includes the subject matter of Example 32, wherein each frame comprises a plurality of work units, wherein each work unit includes a draw representing a portion of an image represented by a frame corresponding to the draw.

Example 34 includes the subject matter of Example 31, wherein the one or more characteristics relate to one or more processing traits relating to execution of the work unit, wherein the one or more characteristics relate to at least one of bottlenecking locations, bottlenecking reasons, cache usage, cache partitioning criteria, shader constraints, and amounts and types and frequencies of hardware allocation, and wherein the hardware allocation comprises allocation of hardware components including one or more of graphics processing unit (GPU), GPU slice, GPU fixed function, computer processing unit (CPU), and memory devices include

double data rate (DDR) devices.

Example 35 includes the subject matter of Example 31, wherein the one or more configuration parameters are based on considerations associated with configuration or reconfiguration of the computing resources comprising the hardware components, wherein the considerations include time consumption and power costs relating to one or more of the computing resources.

Example 36 includes the subject matter of Example 31, wherein means for collecting further comprises means for collecting subsequent metrics relating to and upon execution of the one or more subsequent work units, wherein the subsequent metrics include new metrics or updated measurement of the metrics, and wherein means for generating further comprises means for generating one or more subsequent configuration plans relating to the one or more subsequent work units based on the subsequent metrics.

Example 37 includes the subject matter of Example 31, further comprising means for storing the metrics and the subsequent metrics at a database, and means for retrieving the metrics and the subsequent metrics from the database, wherein the database is further to store the configuration plan and the one or more subsequent configuration plans, wherein the database includes a Cloud-based storage accessible over one or more networks.

Example 38 includes the subject matter of Example 31, wherein one or more of means for detecting, means for collecting, means for evaluating, means for generating, means for applying, storing, and means for receiving are performed via one or more hardware units at the computing device.

Example 39 includes at least one non-transitory or tangible machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims or examples 9-16.

Example 40 includes at least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims or examples 9-16.

Example 41 includes a system comprising a mechanism to implement or perform a method as claimed in any of claims or examples 9-16.

Example 42 includes an apparatus comprising means for performing a method as claimed in any of claims or examples 9-16.

Example 43 includes a computing device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

Example 44 includes a communications device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

The drawings and the forgoing description give examples of embodiments. Those skilled in the art will appreciate that one or more of the described elements may well be combined into a single functional element. Alternatively, certain elements may be split into multiple functional elements. Elements from one embodiment may be added to another embodiment. For example, 5 orders of processes described herein may be changed and are not limited to the manner described herein. Moreover, the actions any flow diagram need not be implemented in the order shown; nor do all of the acts necessarily need to be performed. Also, those acts that are not dependent on other acts may be performed in parallel with the other acts. The scope of embodiments is by no means limited by these specific examples. Numerous variations, whether explicitly given in 10 the specification or not, such as differences in structure, dimension, and use of material, are possible. The scope of embodiments is at least as broad as given by the following claims.

CLAIMS

What is claimed is:

1. An apparatus to facilitate configuration of computing engines based on runtime workload measurements, comprising:

5 reception/detection logic to detect a work unit corresponding to a workload;
metrics collector to collect metrics relating to the work unit, wherein the metrics to indicate one or more characteristics of the work unit;

evaluation logic to evaluate the one or more characteristics based on one or more configuration parameters relating to computing resources;

10 configuration manager to generate, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit; and

execution/application logic to apply and execute the configuration plan at the apparatus upon execution of the one or more subsequent work units.

15 2. The apparatus of claim 1, wherein the workload comprises a software application including a game application, wherein the workload further includes a plurality of frames representing a plurality of images.

3. The apparatus of claim 1 or 2, wherein each frame comprises a plurality of work units, wherein each work unit includes a draw representing a portion of an image represented by
20 a frame corresponding to the draw.

4. The apparatus of claim 1, wherein the one or more characteristics relate to one or more processing traits relating to execution of the work unit, wherein the one or more characteristics relate to at least one of bottlenecking locations, bottlenecking reasons, cache usage, cache partitioning criteria, shader constraints, and amounts and types and frequencies of
25 hardware allocation, and

wherein the hardware allocation comprises allocation of hardware components including one or more of graphics processing unit (GPU), GPU slice, GPU fixed function, computer processing unit (CPU), and memory devices include double data rate (DDR) devices.

5. The apparatus of claim 1 or 4, wherein the one or more configuration parameters
30 are based on considerations associated with configuration or reconfiguration of the computing resources comprising the hardware components, wherein the considerations include time consumption and power costs relating to one or more of the computing resources.

6. The apparatus of claim 1, wherein the metrics collector is further to collect
35 subsequent metrics relating to and upon execution of the one or more subsequent work units, wherein the subsequent metrics include new metrics or updated measurement of the metrics, and

wherein the configuration manager is further to generate one or more subsequent configuration plans relating to the one or more subsequent work units based on the subsequent metrics.

7. The apparatus of claim 1 or 6, wherein the metrics collector is further to store the metrics and the subsequent metrics at a database, wherein the metrics collector is further to retrieve the metrics and the subsequent metrics from the database, wherein the database is further to store the configuration plan and the one or more subsequent configuration plans, wherein the database includes a Cloud-based storage accessible over one or more networks.

8. The apparatus of claim 1, wherein one or more of the reception/detection logic, metrics collector, evaluation logic, configuration manager, and execution/application logic are employed as one or more hardware units at the apparatus.

9. A method for facilitating configuration of computing engines based on runtime workload measurements, comprising:

detecting a work unit corresponding to a workload;

collecting metrics relating to the work unit, wherein the metrics to indicate one or more characteristics of the work unit;

evaluating the one or more characteristics based on one or more configuration parameters relating to computing resources;

generating, based on evaluating of the one or more characteristics, a configuration plan specific to the work unit and applicable to one or more subsequent work units that are similar to the work unit; and

applying and executing the configuration plan at a computing device upon execution of the one or more subsequent work units.

10. The method of claim 9, wherein the workload comprises a software application including a game application, wherein the workload further includes a plurality of frames representing a plurality of images.

11. The method of claim 10, wherein each frame comprises a plurality of work units, wherein each work unit includes a draw representing a portion of an image represented by a frame corresponding to the draw.

12. The method of claim 9, wherein the one or more characteristics relate to one or more processing traits relating to execution of the work unit, wherein the one or more characteristics relate to at least one of bottlenecking locations, bottlenecking reasons, cache usage, cache partitioning criteria, shader constraints, and amounts and types and frequencies of hardware allocation, and

wherein the hardware allocation comprises allocation of hardware components including one or more of graphics processing unit (GPU), GPU slice, GPU fixed function, computer processing unit (CPU), and memory devices include double data rate (DDR) devices.

13. The method of claim 9, wherein the one or more configuration parameters are based on considerations associated with configuration or reconfiguration of the computing resources comprising the hardware components, wherein the considerations include time consumption and power costs relating to one or more of the computing resources.

14. The method of claim 9, wherein collecting further comprises collecting subsequent metrics relating to and upon execution of the one or more subsequent work units, wherein the subsequent metrics include new metrics or updated measurement of the metrics, and wherein generating further comprises generating one or more subsequent configuration plans relating to the one or more subsequent work units based on the subsequent metrics.

15. The method of claim 9, further comprising storing the metrics and the subsequent metrics at a database, and retrieving the metrics and the subsequent metrics from the database, wherein the database is further to store the configuration plan and the one or more subsequent configuration plans, wherein the database includes a Cloud-based storage accessible over one or more networks.

16. The method of claim 9, wherein one or more of detecting, collecting, evaluating, generating, applying, storing, and receiving are performed via one or more hardware units at the computing device.

17. At least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims 9-16.

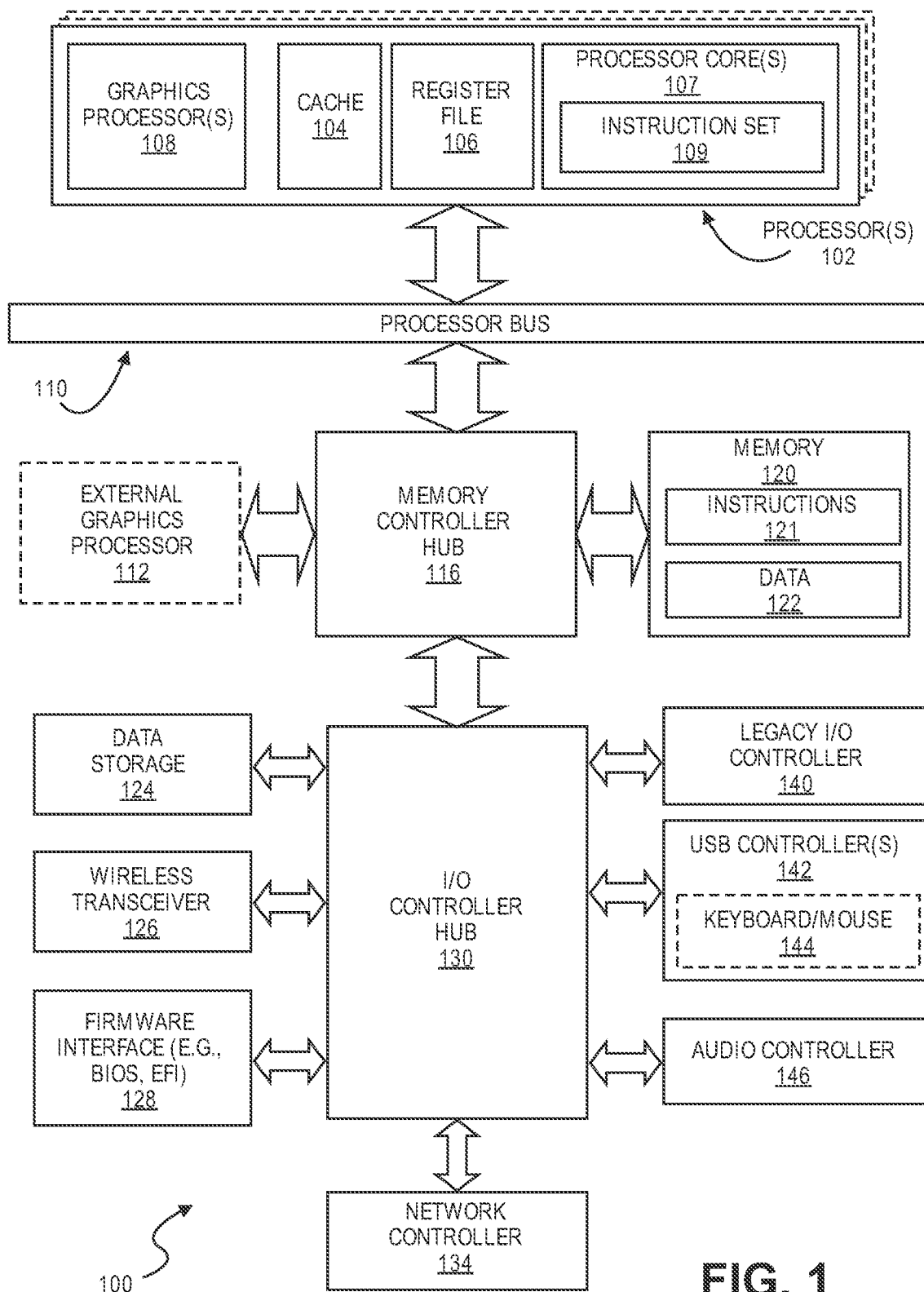
18. A system comprising a mechanism to implement or perform a method as claimed in any of claims 9-16.

19. An apparatus comprising means for performing a method as claimed in any of claims 9-16.

20. A computing device arranged to implement or perform a method as claimed in any of claims 9-16.

21. A communications device arranged to implement or perform a method as claimed in any of claims 9-16.

1/15

**FIG. 1**

PROCESSOR
200

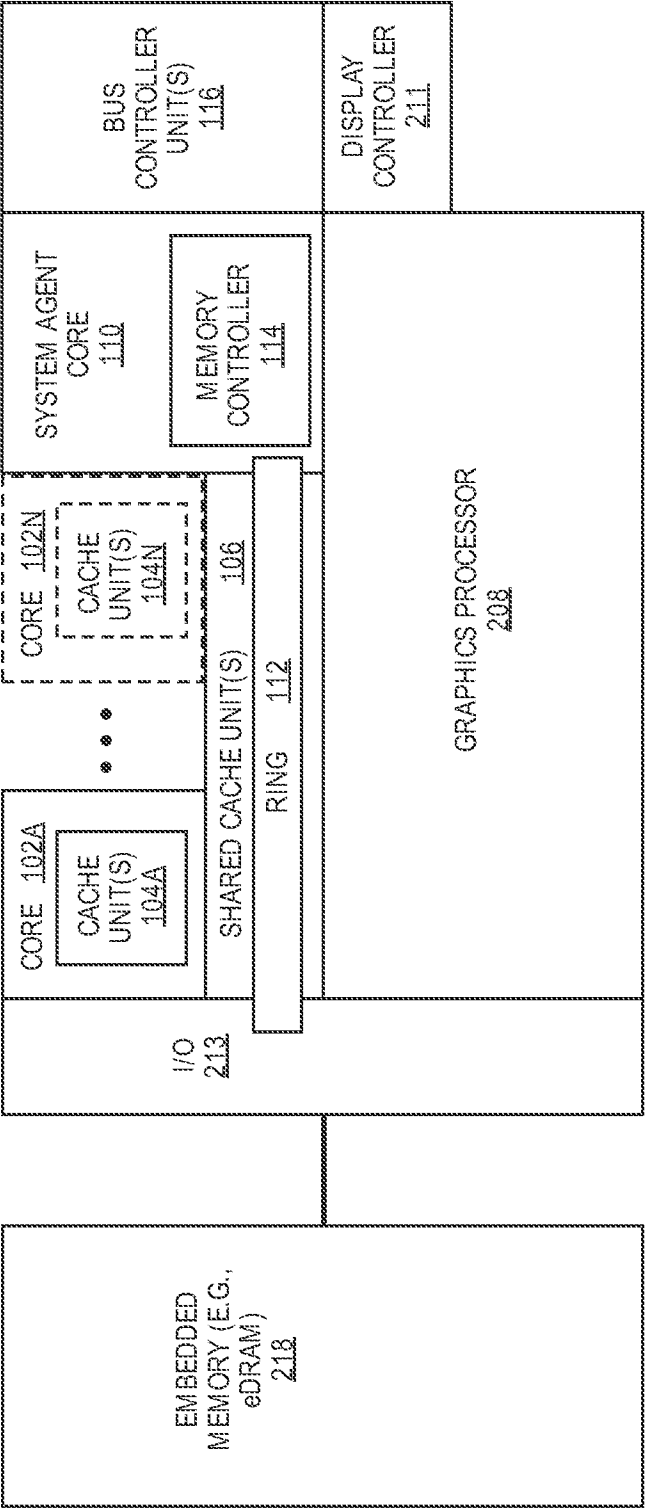


FIG. 2

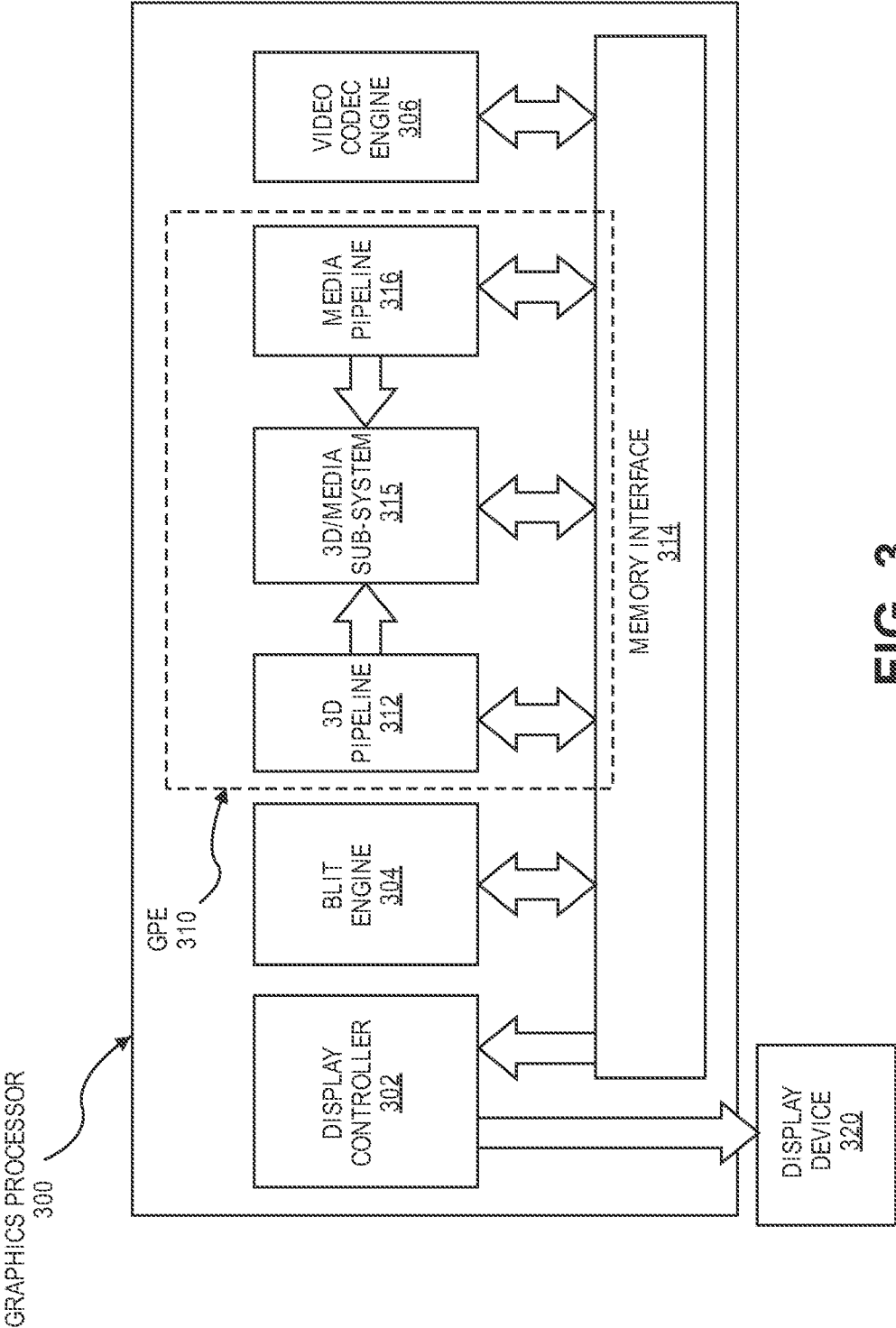


FIG. 3

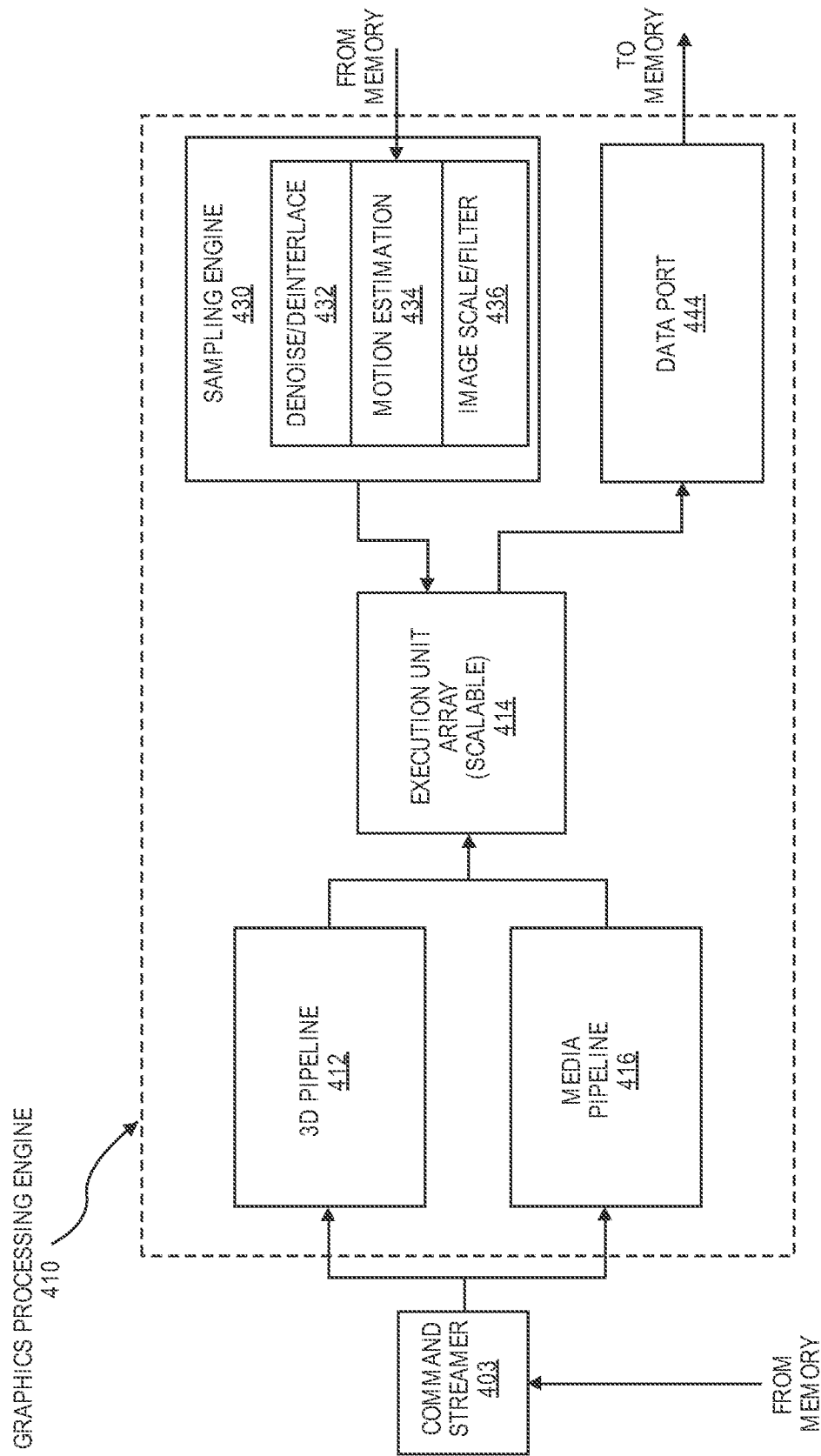


FIG. 4

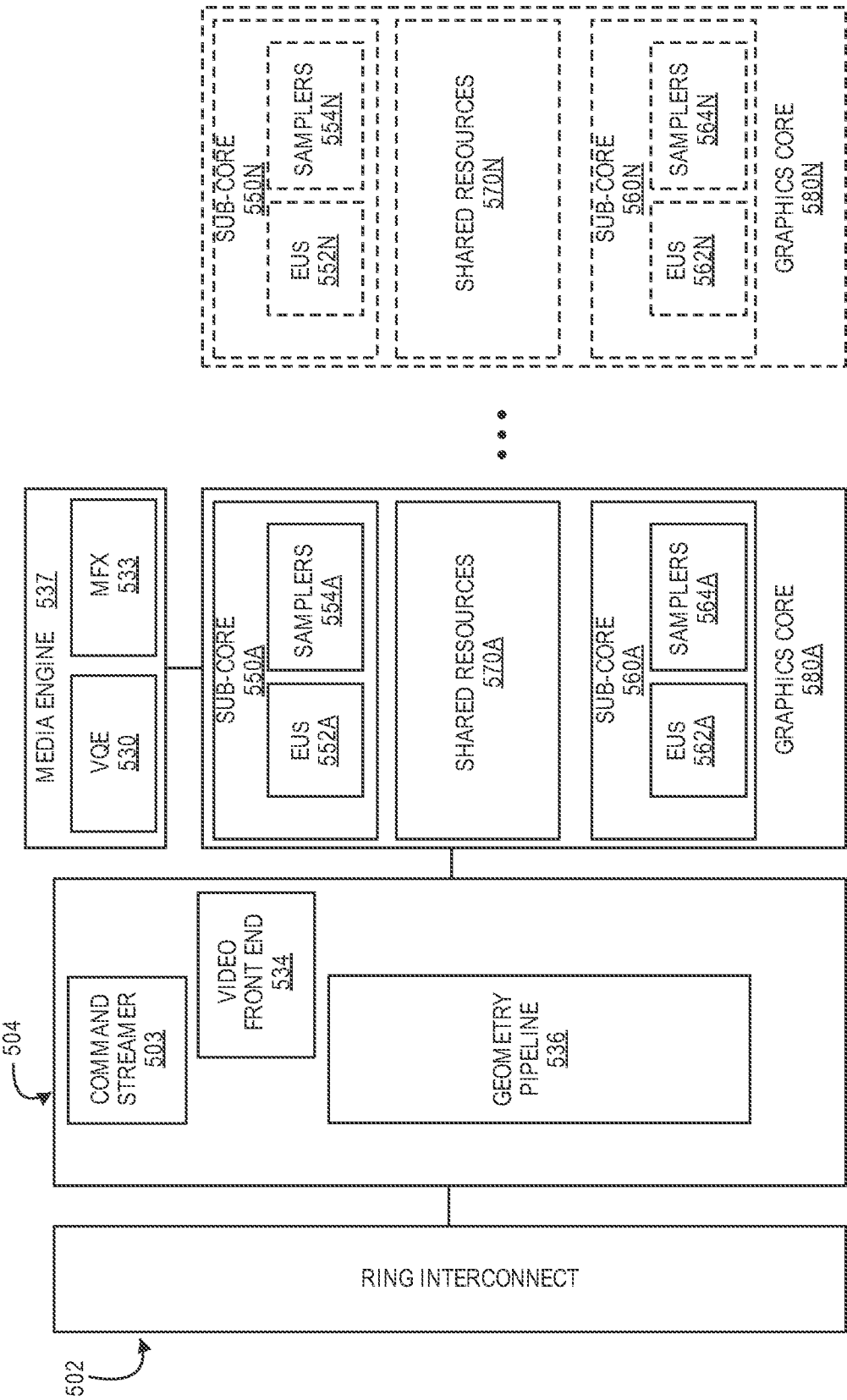


FIG. 5

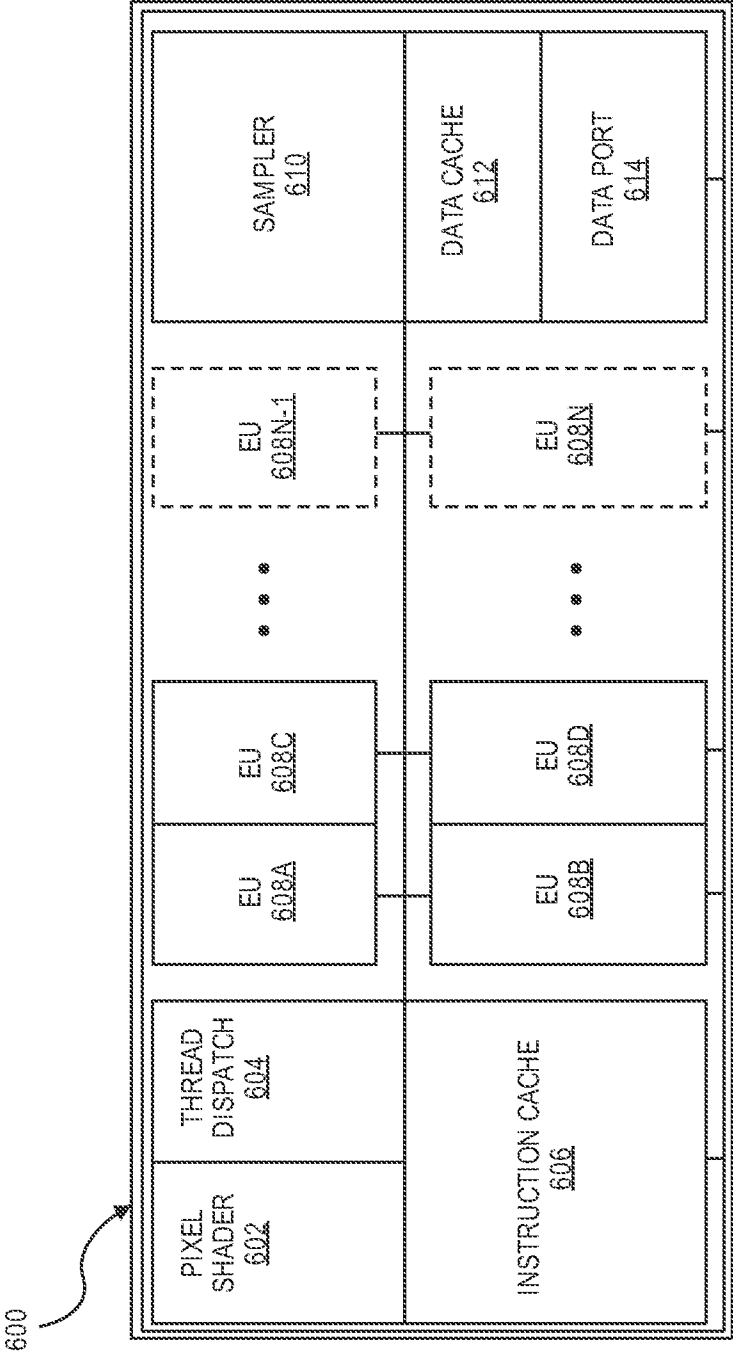
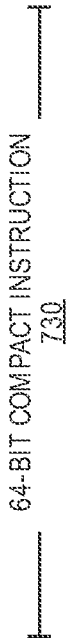
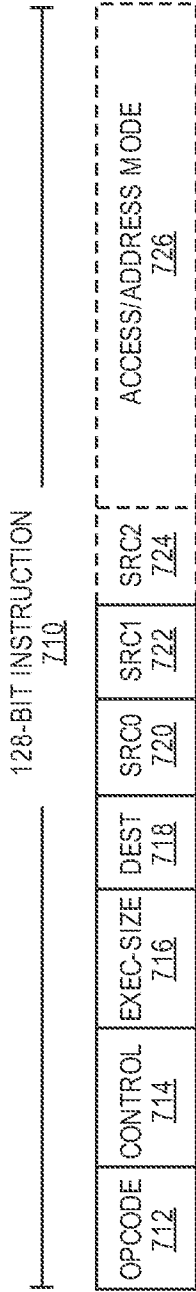


FIG. 6

GRAPHICS CORE INSTRUCTION FORMATS

700



OPCODE DECODE 740

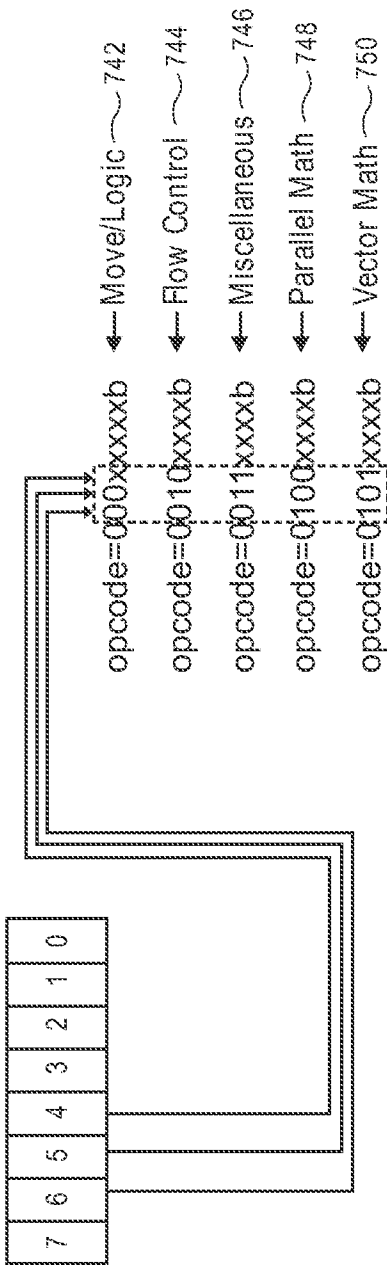


FIG. 7

8/15

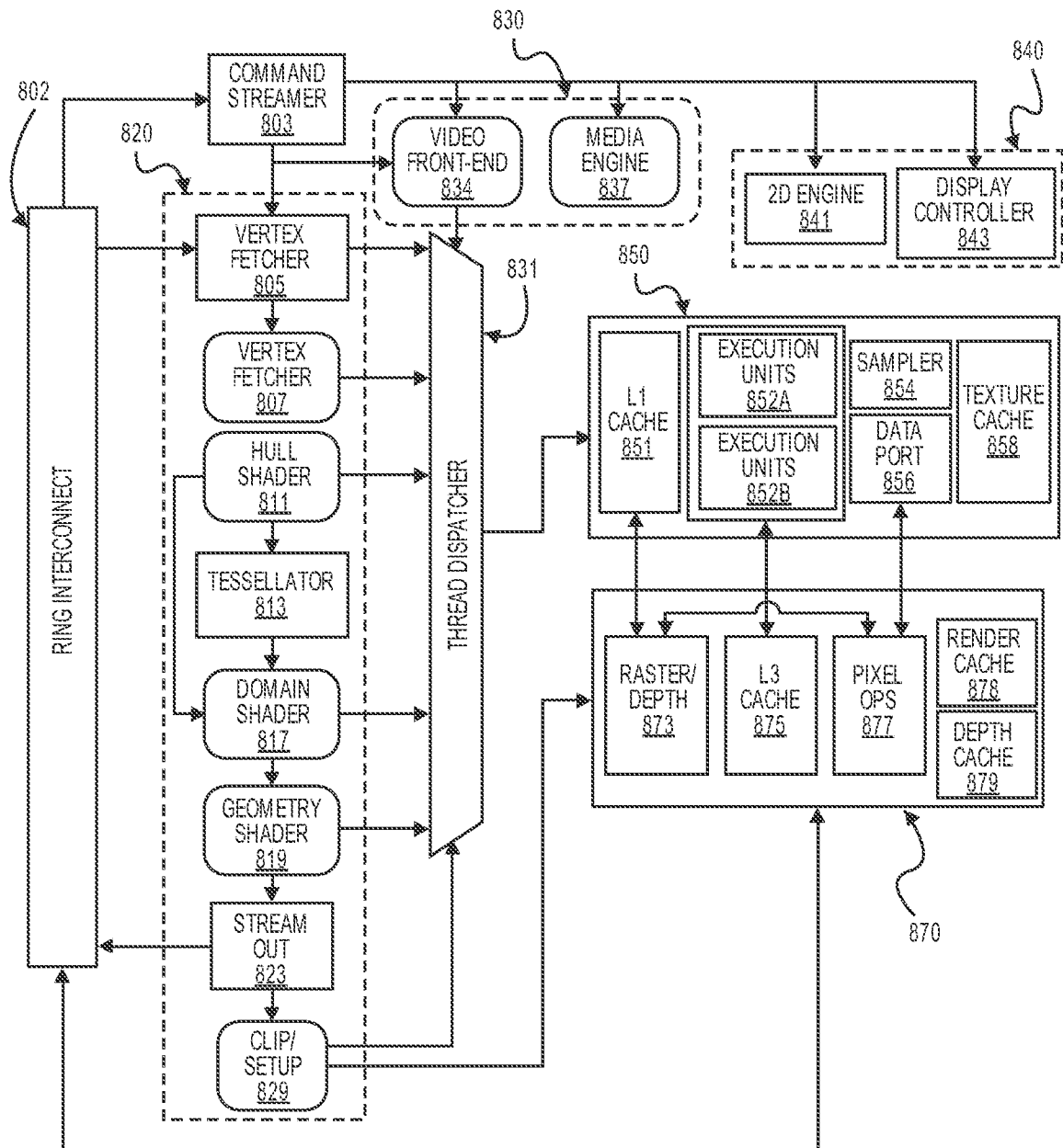


FIG. 8

FIG. 9A

SAMPLE COMMAND FORMAT
900

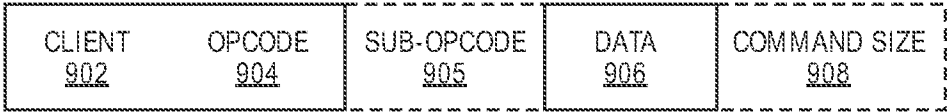
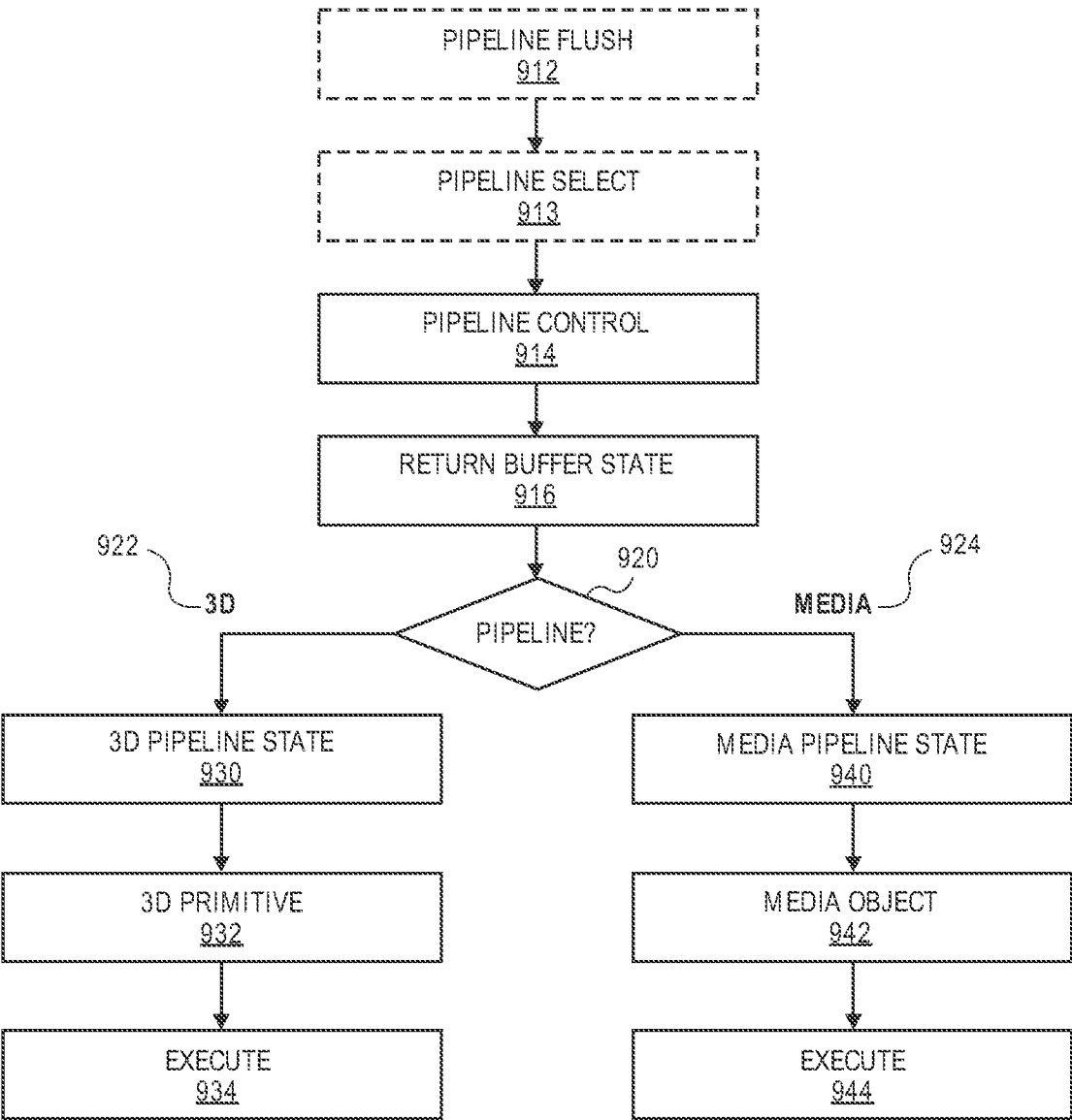
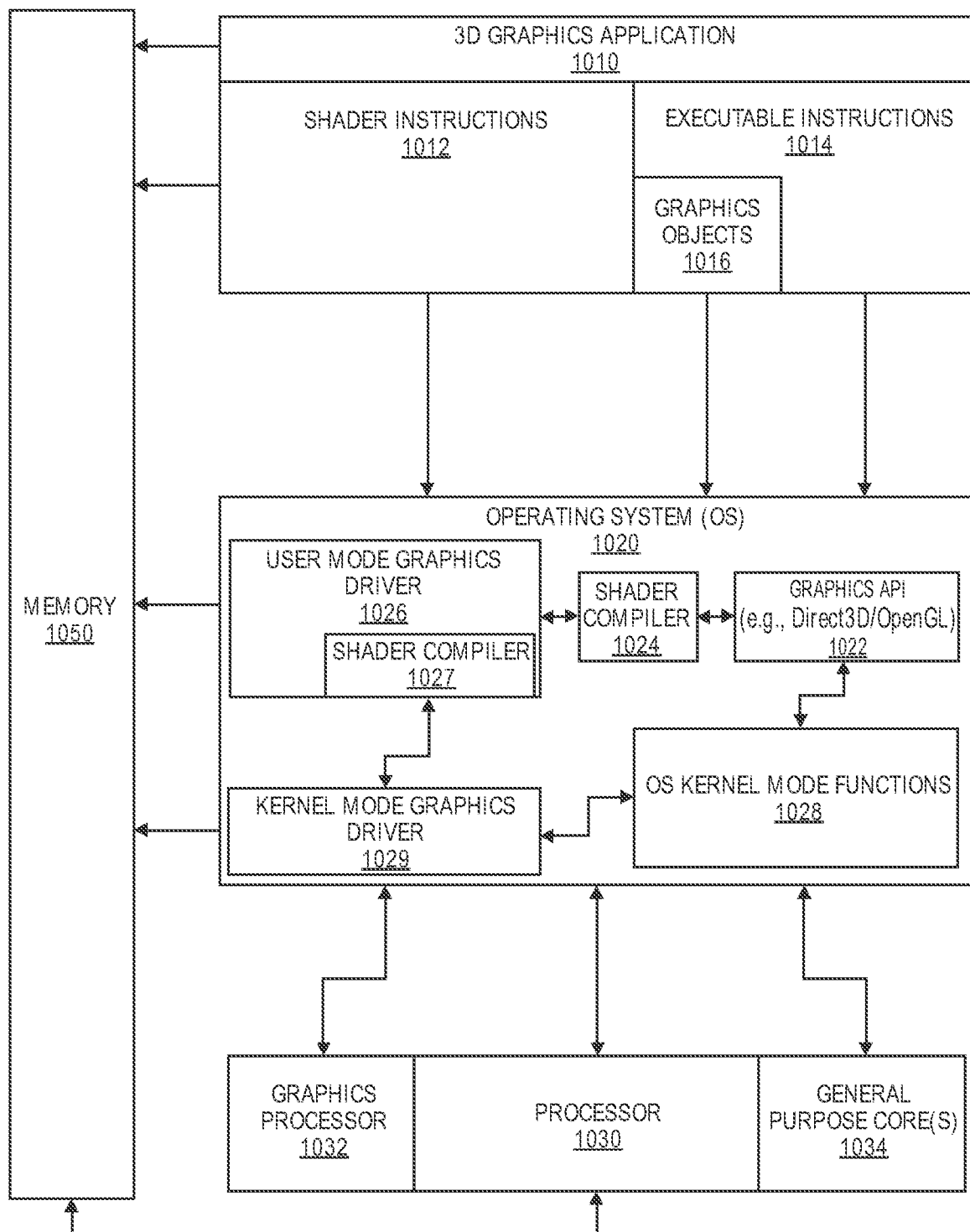


FIG. 9B

SAMPLE COMMAND SEQUENCE
910



10/15

**FIG. 10**

IP CORE DEVELOPMENT - 1100

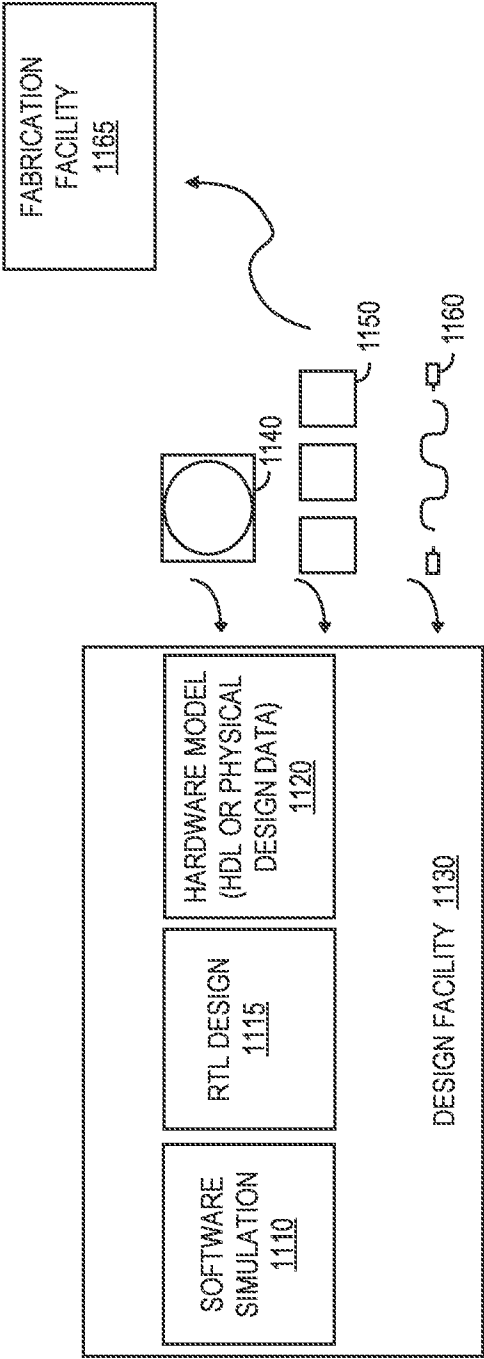
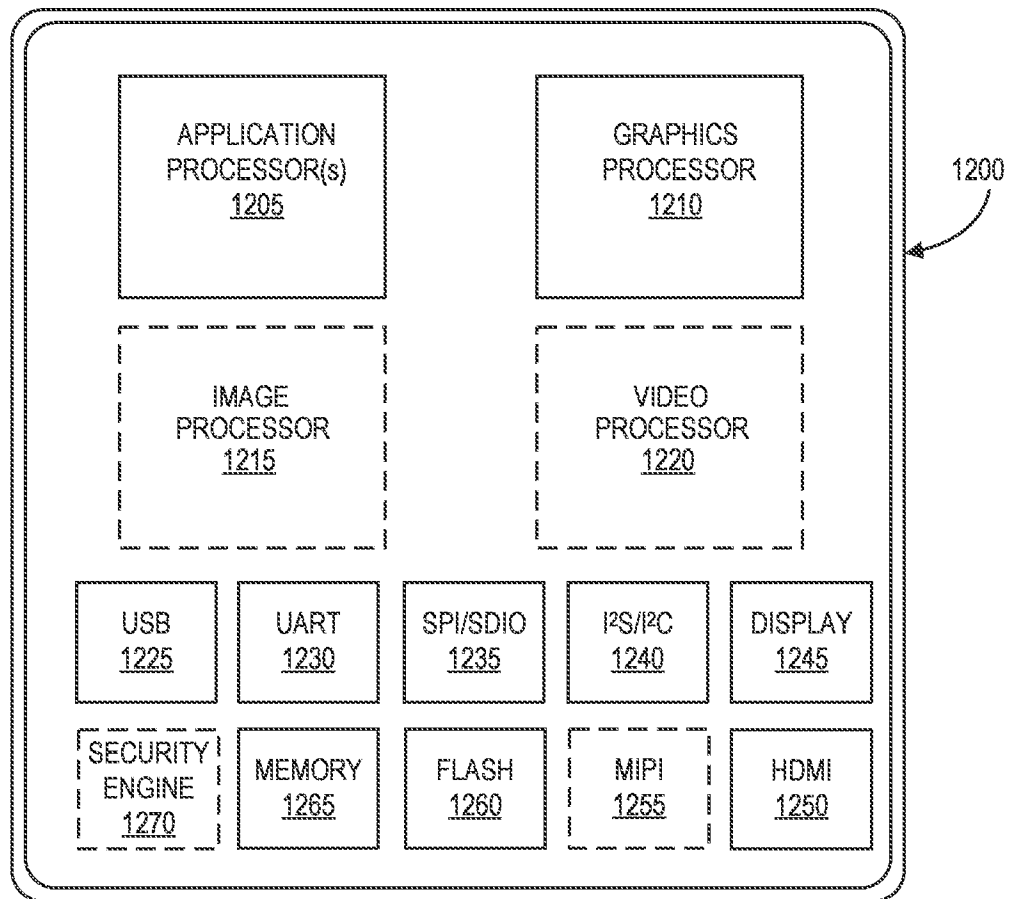


FIG. 11

12/15

**FIG. 12**

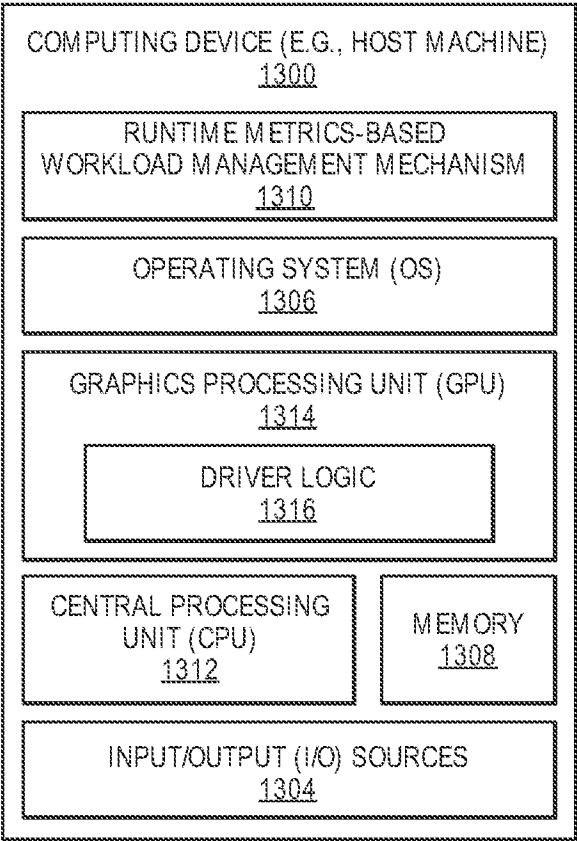
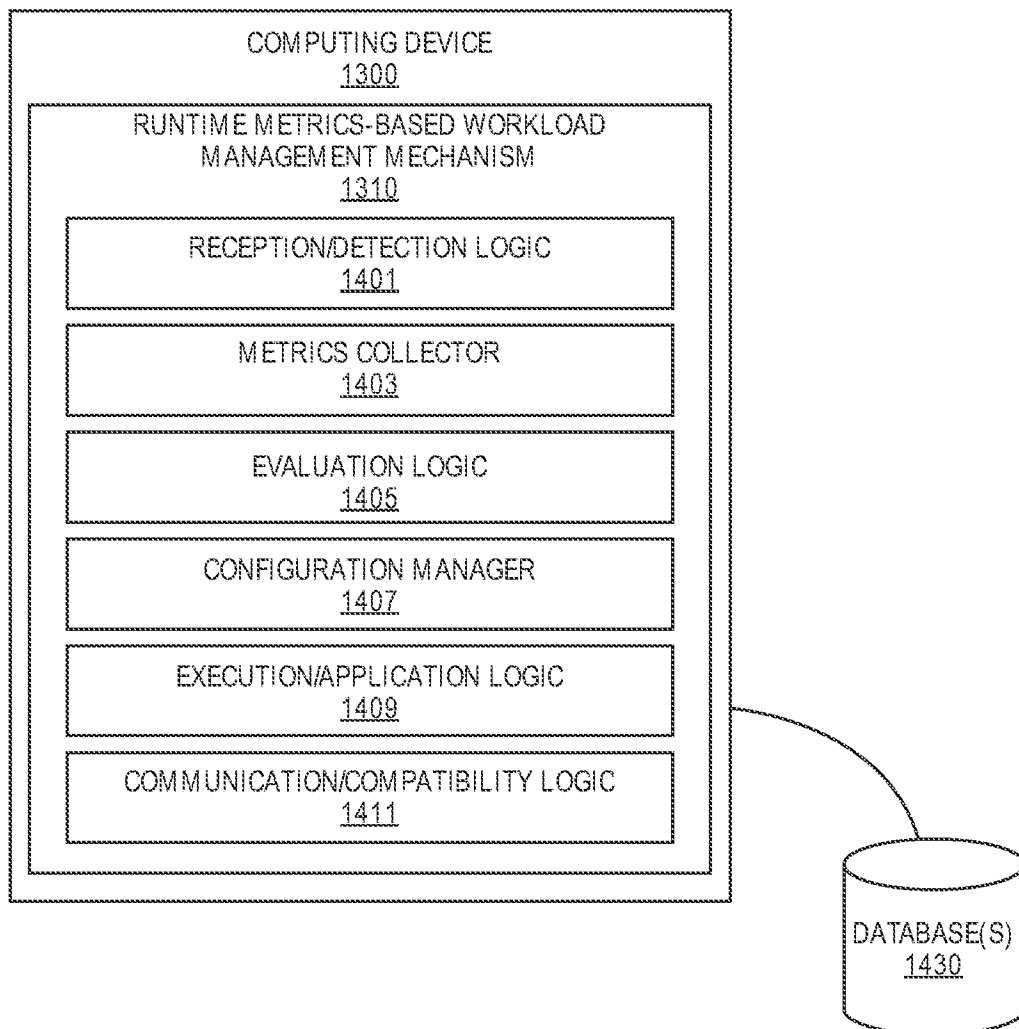
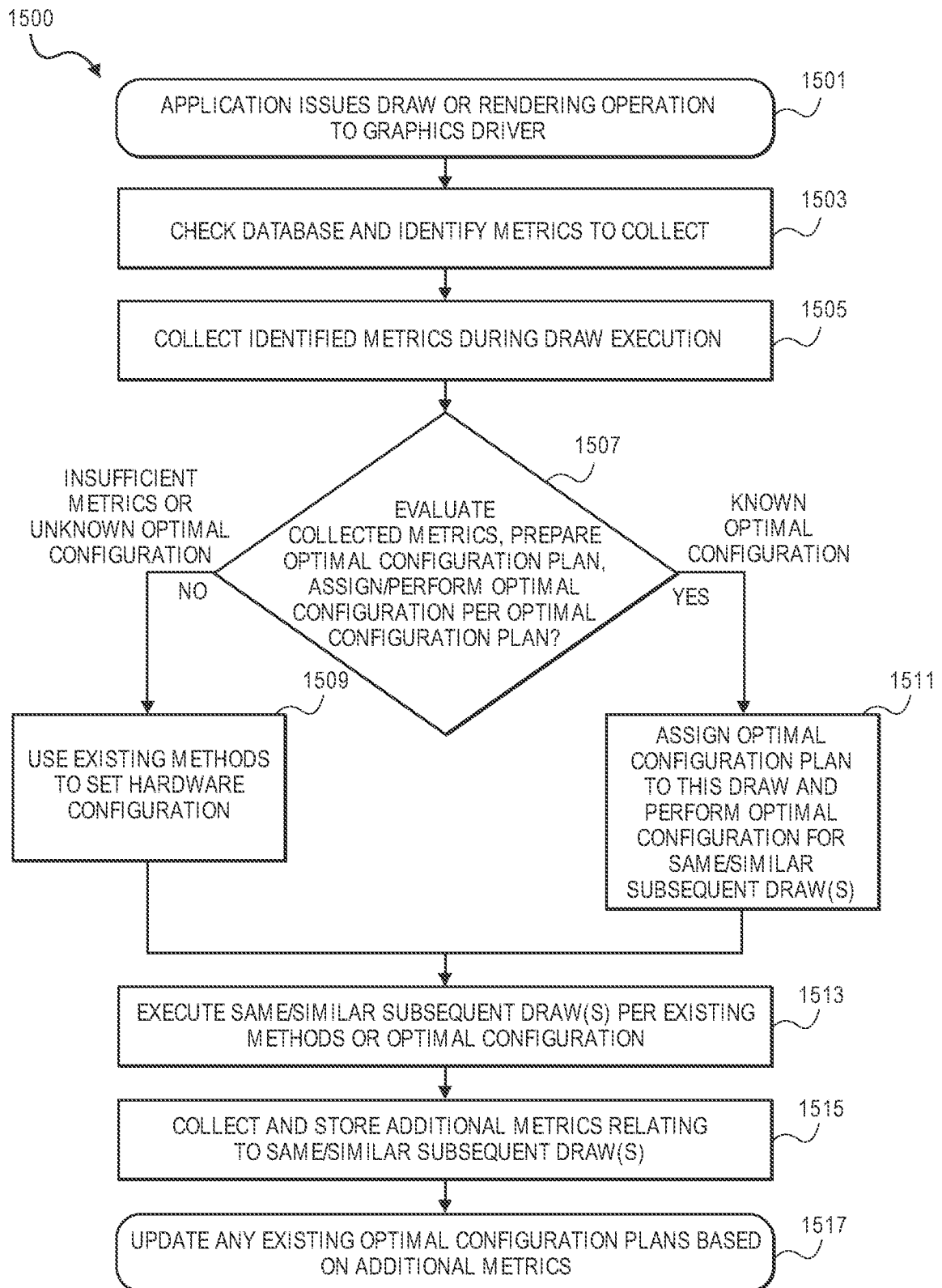


FIG. 13

14/15

**FIG. 14**

15/15

**FIG. 15**

A. CLASSIFICATION OF SUBJECT MATTER**G06F 9/50(2006.01)I, G06T 1/20(2006.01)I, G06F 9/38(2006.01)I, G06F 3/06(2006.01)I, G06F 11/34(2006.01)I**

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/50; G06F 9/46; G06F 13/14; G06F 700; G06T 1/00; G06F 1700; G06T 1/20; G06F 9/38; G06F 3/06; G06F 11/34

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & Keywords: work unit, workload, subsequent, configuration, GPU

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2013-0106881 A1 (APPLE INC.) 02 May 2013 See paragraphs [0029]-[0055]; and claims 1, 3, 9.	1-21
Y	US 6823346 B2 (SCOT MACLELLAN et al.) 23 November 2004 See claim 1; and figure 1.	1-21
A	US 2011-0050713 A1 (REX MCCRARY et al.) 03 March 2011 See paragraphs [0054]-[0056]; and figure 5.	1-21
A	US 2012-0192200 A1 (JAYANTH N. RAO et al.) 26 July 2012 See paragraphs [0020]-[0032]; and figure 1.	1-21
A	US 2014-0035938 A1 (CHANGLIANG CHARLIE WANG et al.) 06 February 2014 See paragraphs [0044]-[0047]; and figure 4A.	1-21



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search

19 August 2016 (19.08.2016)

Date of mailing of the international search report

19 August 2016 (19.08.2016)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

CHIN, Sang Bum

Telephone No. +82-42-481-8398



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/031870

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013-0106881 A1	02/05/2013	US 2016-042489 A1 US 9196013 B2	11/02/2016 24/11/2015
US 6823346 B2	23/11/2004	US 2003-0208470 A1 US 6591262 B1	06/11/2003 08/07/2003
US 2011-0050713 A1	03/03/2011	CN 102597950 A CN 102597950 B EP 2473914 A1 IN 2567 DEN2012 A JP 05722327 B2 JP 2013-504127 A KR 10-1587201 B1 KR 10-2012-0058605 A US 8310492 B2 WO 2011-028896 A1	18/07/2012 29/10/2014 11/07/2012 28/08/2015 20/05/2015 04/02/2013 20/01/2016 07/06/2012 13/11/2012 10/03/2011
US 2012-0192200 A1	26/07/2012	CN 103329100 A EP 2666085 A2 TW 201243618 A WO 2012-099693 A2 WO 2012-099693 A3	25/09/2013 27/11/2013 01/11/2012 26/07/2012 27/12/2012
US 2014-0035938 A1	06/02/2014	CN 103577269 A EP 2693761 A2 EP 2693761 A3 KR 10-2014-0018157 A KR 10-2015-0096639 A US 8928678 B2	12/02/2014 05/02/2014 16/04/2014 12/02/2014 25/08/2015 06/01/2015