



(51) International Patent Classification:

G06F 9/38 (2006.01) G06T 1/20 (2006.01)
G06F 9/30 (2006.01)

(21) International Application Number:

PCT/US2016/031679

(22) International Filing Date:

10 May 2016 (10.05.2016)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

14/738,679 12 June 2015 (12.06.2015) US

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

(72) Inventors: VALERIO, James A.; 2501 NW 229th Avenue, Hillsboro, Oregon 97124 (US). VENKATESH, Abhishek; 7052 NE Ronler Way, Apt. 2917, Hillsboro, Oregon 97124 (US). SARANGI, Satyajit; 444 Saratoga Ave Apt 2K, Santa Clara, California 95050 (US). APODACA, Michael; 1563 Ballou Circle, Folsom, California 95630 (US). RAOUX, Thomas F.; 707 Continental Circle, Apt.1210, Mountain View, California 94040 (US). HASHEMI, Hashem; 520 Heiler Court, Folsom, Califor-

nia 95630 (US). HARIHARA, Rama S.B.; 3600 Juliette Lane, Santa Clara, California 95054 (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 DYNAMIC RUNTIME TRANSFORMATION OF GRAPHICS PROCESSING COMMANDS FOR IMPROVED GRAPHICS PERFORMANCE AT COMPUTING DEVICES

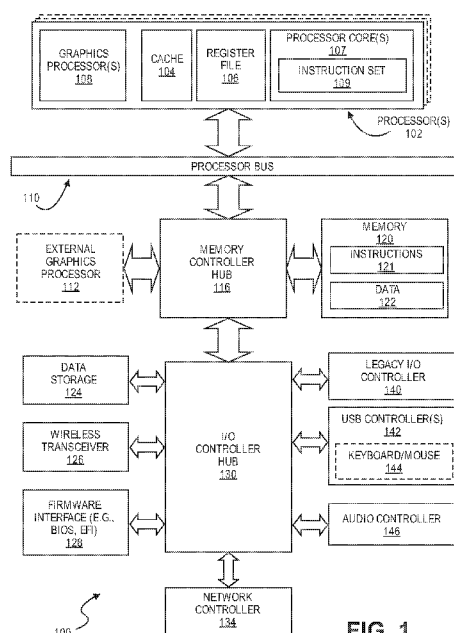


FIG. 1

(57) Abstract: A mechanism is described for facilitating dynamic runtime transformation of graphics processing commands for improved graphics performance on computing devices. A method of embodiments, as described herein, includes detecting a command stream associated with an application, where the command stream includes dispatches. The method may further include evaluating processing parameters relating to each of the dispatches, where evaluating further includes associating a first plan with one or more of the dispatches to transform the command stream into a transformed command stream. The method may further include associating, based on the first plan, a second plan to the one or more of the dispatches, where the second plan represents the transformed command stream. The method may further include executing the second plan, where execution of the second plan includes processing the transformed command stream in lieu of the command stream.



Published:

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

**FACILITATING DYNAMIC RUNTIME TRANSFORMATION OF GRAPHICS
PROCESSING COMMANDS FOR IMPROVED GRAPHICS PERFORMANCE AT
COMPUTING DEVICES**

5 CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of U.S. Patent Application 14/738,679 filed June 12, 2015, entitled FACILITATING DYNAMIC RUNTIME TRANSFORMATION OF GRAPHICS PROCESSING COMMANDS FOR IMPROVED GRAPHICS PERFORMANCE AT COMPUTING DEVICES, and which is incorporated herein by reference in its entirety for all
10 purposes.

FIELD

Embodiments described herein generally relate to computers. More particularly, embodiments relate to a mechanism for facilitating dynamic runtime transformation of graphics processing commands for improved graphics performance at computing devices.

15 BACKGROUND

Conventionally, graphics drivers are known to rely on applications to implement algorithms or perform low level optimization that is restricted to a single General-purpose computing on graphics processing unit (GPGPU) command in isolation. However, it is contemplated that an application is incapable of optimizing an algorithm for each graphics
20 processing unit (GPU), making for inefficient and costly graphics processing and rendering performances, in terms of system resources, power, etc.

BRIEF DESCRIPTION OF THE DRAWINGS

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.

25 **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.

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.

30 **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.

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.

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 dynamic runtime commands transformation mechanism according to one embodiment.

Figure 14A illustrates a dynamic runtime commands transformation mechanism according to one embodiment.

Figure 14B illustrates an architectural placement according to one embodiment.

Figure 15 illustrates a transactional sequence including compile time and runtime transactions for efficiently processing a command sequence according to one embodiment.

Figure 16A illustrates a method including compile time processes for efficiently processing a command sequence according to one embodiment.

Figure 16B illustrates a method including runtime time processes for efficiently processing a command sequence 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 framework and methodology to enable a graphics driver to perform both high and low level optimizations to reduce application development cost without sacrificing performance. For example and in one embodiment, the graphics driver may use certain underlying features of the hardware that are not exposed through an application programming interface (API) to go beyond what an application may be capable of achieving in terms of functionality and performance. Further, in one embodiment, the graphics driver may be facilitated by this novel technique to record a stream of GPGPU command and compute shader

kernels and post processes with knowledge about the underlying GPU before submitting work on it.

Embodiments provide for a novel technique for dynamic and runtime transformation of command dispatches that are put forth by an application (such as a three-dimensional (3D) game) during its execution, where, in one embodiment, such dispatches may be reduced in numbers, increased in numbers, combined together into fewer dispatches, separated into additional dispatches, routed or rerouted, held back or pushed forward, and/or the like, for better and more efficient rendering of the application as will be further discussed throughout this document.

It is contemplated and to be noted that “dispatch” may be interchangeably referred to as “work unit” or “draw” and similarly, “application” may be interchangeably referred to as “workflow”. For example, a workload, such as a 3D game, may include and issue any number and type of “frames” where each frame may represent an image (e.g., sailboat, human face). Further, each frame may include and offer any number and type of work units, where each work unit may represent a part (e.g., mast of sailboat, forehead of human face) of the image (e.g., sailboat, human face) represented by its corresponding frame. However, for the sake of consistency along with brevity, clarity, and ease of understanding, any single term, such as “dispatch”, “application”, etc., may be used throughout this document.

It is contemplated that GPGPU APIs may allow for arbitrary programming of a GPU, such as in games and high performance computing. In one embodiment, this novel technique may facilitate the graphics driver to perform various tasks to reduce the effort spent in optimizing applications in every GPU and thus improving the user (e.g., developer, end-user) experience, such as in case of 3D games and high-performance computing where the graphics performance of various algorithms maybe automatically improved through the graphics driver. For example, an application may include a game, a workstation application, etc., offered thorough an API, such as a free rendering API, such as Open Graphics Library (OpenGL®), DirectX® 11, etc.

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 for efficient performance using this novel technique may range from simple scenarios, such as desktop compositing, to complex scenarios, such as 3D games, augmented reality applications, etc.

System Overview

5 **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 one embodiment, the system 100 is a processing platform incorporated within a
10 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
15 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.

20 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 Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). Multiple processor cores 107
25 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 a Digital Signal Processor (DSP).

In some embodiments, the processor 102 includes cache memory 104. Depending on the
30 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 processor cores 107 using known cache coherency techniques. A register file 106 is additionally
35 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.

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 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 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 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 discreet 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
5 embodiments each processor core also has access to one or more shared cached 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
10 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
15 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
20 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
25 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
30 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
35 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 a 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 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 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, including three-dimensional (3D) graphics operations and media operations.

5 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 execution threads to a 3D/Media sub-system 315. While 3D pipeline 312 can be used to
10 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 units to perform one or more specialized media operations, such as video decode acceleration,
15 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 3D/Media sub-system 315.

20 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
25 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

30 **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
5 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
10 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
15 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
20 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
25 embodiments, ring interconnect 502 couples the graphics processor to other processing units, including other 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
30 pipeline front-end 504. In some embodiments, graphics processor 500 includes scalable 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
35 engine 537. In some embodiments, media engine 537 includes a Video Quality Engine (VQE)

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.

5 In some embodiments, graphics processor 500 includes scalable thread execution resources 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
10 having at least a first sub-core 550A and a second core sub-core 560A. In other embodiments, the 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
15 552A-552N and media/texture samplers 554A-554N. Each sub-core in the set of second sub-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
20 embodiments of the graphics processor.

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.

25 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
30 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
35 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 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.

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**)

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 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 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 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 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 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.

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-

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 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 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 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 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.

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 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 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 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 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 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 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 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 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 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
5 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 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
10 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 pipeline commands unless the context is to issue commands for both pipelines. In some embodiments, a pipeline flush command 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
15 operation and is used to program the 3D pipeline 922 and the media pipeline 924. In some 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
20 batch of commands.

In some embodiments, return buffer state commands 916 are used to configure a set of
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
25 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 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
30 3D pipeline 922 beginning with the 3D pipeline state 930, or the media pipeline 924 beginning at the media pipeline state 940.

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
35 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 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 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. 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 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 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.

5 **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
10 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
15 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.
20 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)
25 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
30 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
35 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 dynamic runtime commands transformation mechanism 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, clarity, 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 dynamic runtime commands transformation mechanism (“transformation mechanism”) 1310.

Transformation mechanism 1310 may include any number and type of components for facilitating runtime transformation of graphics processing commands for efficient performance of graphics contents, such as images, videos, 3D applications, games, etc., 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”.

5 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
10 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
15 software components of computing device 1300 on a single chip.

As illustrated, in one embodiment, in addition to employing transformation 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)
20 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 processors, such as processor(s) 102 of **Figure 1**, while GPU 1314 may include one or more
25 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, transformation 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 as is described through this document.

30 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”, “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
5 are 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
10 any number and type of instructions, such as instructions 121 of **Figure 1**, to perform the various novel functions of transformation 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
15 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
20 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);
25 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
30 perform 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
35 whilst 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 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 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 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 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 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 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 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 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 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 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 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 14A illustrates a dynamic runtime commands transformation mechanism 1310 according to one embodiment. In one embodiment, transformation mechanism 1310 may include any number and type of components to perform various tasks relating to intelligent, dynamic, and real-time transformation of graphics processing commands for efficient rendering of display contents at computing device 1300. For example and in one embodiment, transformation mechanism 1310 may include any number and type of components, such as (without limitation) reception/detection logic 1401, evaluation logic 1403, intermediary building/mapping logic 1405, execution/application logic 1407, and communication/compatibility logic 1409. Computing device 1300 is further in communication with database 1430 (e.g., Cloud storage, non-Cloud storage, etc.) over one or more networks (e.g., Internet, Cloud-based network, proximity network, etc.)

As an initial matter, it is contemplated and to be noted that, in one embodiment, transformation mechanism 1310 may be hosted by driver logic 1316 of GPU 1314 while, in

another embodiment, transformation mechanism 1310 may be hosted by GPU 1314, while, in yet another embodiment, transformation mechanism 1310 may be hosted by both graphics driver 1316 and GPU 1314, such as any software components of transformation mechanism 1310 may be hosted by driver logic 1316, and any hardware components of transformation mechanism 1310 may be embedded at GPU 1314. In yet another embodiment, transformation mechanism 1310 may not be hosted by driver logic 1316 or GPU 1314 and that it may be hosted by another component of computing device 1300, such as operating system 1306 of **Figure 13**, but work in communication with driver logic 1316, GPU 1314, etc. Accordingly, it is contemplated that embodiments are not limited to any particular form or manner of implementation or placement, such as illustrated with respect to **Figure 14B**.

As aforementioned, an application (e.g., game application) may include any number and types of frames having any number and type of dispatches, where each dispatch 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, any size and content of a dispatch, 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, dispatches, 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 dispatches and each dispatch may have any type of image content. For example, in some embodiments, a dispatch may indicate a number of threads per group to be used and/or a number of groups are to be executed in a command sequence, etc.

As an initial matter, as will be further described throughout this document and illustrated with reference to **Figure 15**, embodiments are not limited a particular use case or optimization scenario and that any number and type of optimization scenarios may be exercised using transformation mechanism 1310. For example, transformation mechanism 1310 may enable or facilitate graphics driver logic 1316 to perform both high and low level optimizations for GPGPU applications (e.g., game applications) for improved performance on a corresponding GPU 1314.

For example, such performance enhancement opportunities for GPU/GPGPU workloads may be achieved through any number and type of optimization scenarios as facilitated by transformation mechanism 1310, such optimization scenarios may include (without limitation): 1) redirecting global scratch memory for intermediate passes to a shared local memory to reduce memory bandwidth; 2) reducing thread latency by merging small GPGPU kernels into a single kernel; 3) breaking a large thread group into smaller ones to avoid cache thrashing; 4) executing commands out of order for better efficiency; 5) using features available on GPU 1314 not

exposed by the API for improved performance; 6) changing the walk order of GPGPU threads for more cache locality; and/or the like.

For example, the aforementioned and other such optimization scenarios, as facilitated by transformation mechanism 1310, allow for reducing the effort spent in optimizing applications for every GPU which, in turn, improves the viewing experience for the user, such as a software developer, a system administrator, an end-user, etc., using 3D games and other high performing computing, etc., by having transformation mechanism 1310 work with graphics driver logic, such as driver logic 1316 of **Figure 13**. For example, transformation mechanism 1310 may be implemented in and/or used to facilitate graphics driver (e.g., DirectX® 11 driver of Intel® Graphics Processor, such as driver logic 1316 of **Figure 13**, may be enabled by transformation mechanism 1310 to collapse 64 dispatches of compute shader to merely 8 dispatches while directing the scratch global unordered access view (UAV) surfaces for intermediate passes to a shared local memory, which may result in cutting of memory bandwidth and improving overall performance of the application.

In one embodiment, reception/detection logic 1401 may be used to detect or receive a kernel of an application, a command sequence placed by the kernel, and/or one or more dispatches of the command sequence, etc. For example, application may submit a stream of GPGPU commands to graphics driver logic 1316 to launch compute shade kernels on the corresponding GPU, such as GPU 1314, where reception/detection logic 1401 detects or receives the command stream and its various dispatches and other details which may then be used by other components of transformation mechanism 1310 to transform the command stream to achieve efficient performance of GPU 1314.

For example and in one embodiment, as further illustrated with respect to **Figure 15**, upon detection the command stream, reception/detection logic 1401 further detect a number of dispatches that are set to be processed as part of the command stream. As previously described, a dispatch may reveal a number groups of threads along with a number of threads per group that are to be used for processing of each dispatch and thus the command stream.

In one embodiment, upon detecting or receiving the command stream at reception/detection logic 1401, any relevant information regarding the command stream may then be forwarded on to evaluation logic 1403 for further processing. As aforementioned, any number and types of optimization scenarios may be applied, such as reducing or increasing the number of dispatches, re-scheduling of dispatches, re-ordering of dispatches, re-directing of dispatches, changing walk order of GPGPU threads for more cache locally, merging multiple kernels into fewer or a single kernel, etc. It is contemplated and to be noted that although one or

more optimization scenarios are discussed and illustrated throughout this document, such as in **Figure 15**, embodiments are not limited as such to any particular optimization scenario.

Accordingly, in one embodiment, evaluation logic 1403 may receive the command stream/sequence and any relevant information from reception/detection logic 1401 and evaluate various aspect of the command stream to determine one or more optimization scenarios that may be viable with and application to this command stream to ensure an efficient processing and rendering of the command stream in terms of any number and type of computing resources, such as GPU processing, global memory, local cache, power, speed/time, etc.

For example, if a dispatch has a big memory footprint, evaluation logic 1403 may determine to make it small by dividing it into two or more smaller dispatches that are mapped to a local cache, as opposed to a global memory. For example, a dispatch's group of threads may have 32 threads x 32 threads, equaling 1024 threads, which is considered a rather large group of threads and thus, in one embodiment, such a dispatch or, in other words, its group of threads may be broken down in half, such as down to 16 threads x 16 threads, equaling 256 threads, for efficient processing. For example, even if a single large dispatch is broken down into multiple dispatches, merely the first read of the first of the multiple dispatches may access for data at the global memory and the last write of the last of the multiple dispatches may access the global memory, but all other in-between read and writes may be done by accessing a local cache, making efficient use of the local cache and without burdening the global memory as will be further described with reference to **Figure 15**.

In some embodiments, as illustrated with respect to **Figure 15**, multiple dispatches or kernels may be merged into fewer dispatches or kernels, respectively, to reduce the number of transactions and, in turn, increase efficiency in the processing of the command stream as determined by evaluation logic 1403. For example, in some cases, a number of dispatches may be reduced because an increased number of transactions may crash the local cache as evaluated by evaluation logic 1403. Similarly, for example, a number of dispatches may be merged together because they may reference the same cache surface or portion, such as dispatch 0 (D0) and dispatch 3 (D3) may be merged together as they both are determined to reference cache surface 0 (S0), etc., as evaluated by evaluation logic 1403.

Similarly, in some embodiments, one or more dispatches may not be changed and left to remain in their original size and location/order, such as when a dispatch is regarded as being associated with the appropriate number of threads, group of threads, etc., and at the right order in the sequence within the command stream as determined by evaluation logic 1403. In some embodiments, an optimal scenario where dispatches within a command stream may be treated

with a combination of reduction in some dispatches, increase in some dispatches, and no change in some dispatches, etc., may be determined to be the best scenario by evaluation logic 1403.

In one embodiment, evaluation logic 1403 may further evaluate the sequence or order of dispatches within the command stream. For instance, continuing with the examiner above, if D0 and D3 both reference S0 in the cache, while dispatch 1 (D1) references surface 1 (S1), and dispatch 2 (D2) references surface 2 (S2), etc., evaluation logic 1403 may determine to reorder the dispatch sequence such that D0 and D3 may be processed together to reference S0, saving an additional transaction of accessing S0 of the local cache and, in turn, increasing the efficiency with respect to the cache and the common stream processing.

In one embodiment, as aforementioned, memory bandwidth may be further reduced by redirecting a scratch memory for intermediate passes to a shared local memory, such as (without limitation) reads and writes of various dispatches may be directed to a shared local cache as opposed to a global memory to efficiently reduce the number of memory transactions, saving the memory bandwidth as determined by evaluation logic 1403. In some embodiment, evaluation logic 1403 may determine to facilitate the use of various features available through GPU 1314 but not exposed by the relevant API for increased efficiency and improved performance. In some embodiments, evaluation logic 1403 may determine to change the walk order of various GPGPU threads for more cache locality.

Further, evaluation logic 1403, in some embodiments, may take into consideration the various dependencies of dispatches. For example, if D3 depends from D2, then evaluation logic 1403 ensures that D2 is processed before D3 such that the command stream is properly processed without being impacted negatively or failing entirely. Stated differently, although evaluation logic 1403 may take into account any number of optimization scenarios, certain necessary protocols, parameters, rules, standards, etc., may be honored to ensure a proper and uninterrupted processing of the command stream.

In one embodiment, this accessing of the local cache as opposed to the global memory may lead to mapping or remapping of various dispatches which may be facilitated by building/mapping logic 1405 in accordance with the evaluation results as prepared by and obtained from evaluation logic 1403.

In one embodiment, upon evaluation of the command stream and its contents (e.g., dispatches, input/output swaps, threads, groups of threads, etc.) along with other relevant information in light of the availability and limitations of computing resources (e.g., GPU processing capabilities, available memory space, local cache limitations, processor speed, power availability and resources, etc.), evaluation logic 1403 prepares results of the evaluation and forwards the evaluation results on to building/mapping logic 1405 for further processing.

Upon receiving the evaluation results, building/mapping logic 1405 prepares an intermediary optimization plan based on one or more optimization proposals in the evaluation results which may include (without limitation) charts of reduction/increase in dispatches, graphs for merging/breaking down of dispatches, mapping of dispatches to local cache, dependency charts, etc. This intermediary plan may include an intermediary format of an optimization plan that can be easily post processed and then submitted to execution/application logic 1407 to facilitate GPU 1314 for application and optimal execution of the command stream in accordance with the intermediary plan.

Further, in one embodiment, optimization plans, intermediary plans, and any other information relating to command streams, etc., may be stored at database 1430 for future reference and processing. As illustrated, 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 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 any particular number or type of communication medium or networks.

Communication/compatibility logic 1409 may be used to facilitate dynamic communication and compatibility between one or more computing devices, such as computing device 1300, 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.

5 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 "GPU", "GPU domain", "GPGPU", "CPU", "CPU domain", "kernel", "workload",
10 "application", "work unit", "draw", "dispatch", "local cache", "global memory", "command stream", "command sequence", 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.

It is contemplated that any number and type of components may be added to and/or removed from transformation mechanism 1310 to facilitate various embodiments including
15 adding, removing, and/or enhancing certain features. For brevity, clarity, and ease of understanding of transformation 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 described herein, are not limited to any particular technology, topology, system, architecture, and/or standard and are dynamic enough to adopt and
20 adapt to any future changes.

Figure 14B illustrates an architectural placement of dynamic runtime commands transformation mechanism 1310 of **Figure 14A** according to one embodiment. As an initial matter, for the sake of brevity, any components or processes of transformation mechanism 1310 previously discussed with respect to **Figures 13-14A** are not discussed hereafter. In the
25 illustrated embodiment, transformation mechanism 1310 shown to be hosted as part of graphics driver packages, including (without limitation) graphics/GPUPU driver 1316, shader compiler 1453, and transformation mechanism 1310. As further illustrated, transformation mechanism 1310 remains in communication with GPU 1314 and, as aforementioned with respect to **Figure 14A**, in some embodiments, any software components of transformation mechanism 1310 may
30 be hosted by graphics driver 1316 while any hardware components of transformation mechanism 1310 may be embedded at GPU 1314. It is contemplated that embodiments are not limited to a particular form or manner of implementation and that any variety of placement, such as the one illustrated here, may be executed to achieve the various functionalities of transformation mechanism 1310.

Figure 15 illustrates a transactional sequence 1500 including compile time and runtime transactions for efficiently processing a command sequence according to one embodiment. Transactional sequence 1500 may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof. In one embodiment, transactional sequence 1500 may be performed by transformation mechanism 1310 of **Figures 13-14**. The processes of transactional sequence 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 **Figures 1-14** may not be discussed or repeated hereafter.

As illustrated, in one embodiment, an application running at a computing device, such as computing device 1300 of **Figure 13**, may issue original kernel (also referred to as “compute kernel”) 1501 for processing of original command sequence 1511 by the computing device’s GPU, such as GPU 1314 of **Figure 13**, using other computing resources, such as cache, memory, etc. In the illustrated embodiment, original command sequence 1511 is shown to include any number and type of dispatches, such as dispatch 0 1513A, dispatch 1 1513B, dispatch 2 1513C, dispatch 3 1513D, dispatch 4 (D4) 1513E, etc., corresponding to their input/output swaps 1515A, 1515B, 1515C, 1515D, etc. As further illustrated, input/output swaps 1515A-1515D may perform swapping of inputs and outputs are represented by read/write command sequence 1503 having a read command (e.g., reading from input buffer) and a write command (e.g., writing to output buffer) in original kernel 1501.

In one embodiment, reception/detection logic 1401 of **Figure 14** may detect original command sequence 1511 including dispatches 1513A-1513E and provides this information to evaluation logic 1403. At evaluation logic 1403 of **Figure 14**, original command sequence 1511 and any relevant information is evaluation and, in one embodiment, an optimization plan for merging the five back-to-back dispatches 1513A-1513E into a single dispatch, such as dispatch 1533, may be determined to be the best plan for this original command sequence 1511. This optimization plan may then be proposed by evaluation logic 1403 to building/mapping logic 1405 of **Figure 14** for further processing.

Upon receiving the optimization plan, this building/mapping logic 1405 may receive the optimization plan and in reviewing it, prepare an intermediary plan based on the optimization plan. In one embodiment, this intermediary plan by building/mapping logic 1405 of **Figure 14** may propose to transform original kernel 1501 into fused kernel (also referred to as “transformed kernel”) 1521 that now includes a programming *for*-loop, such as *for*-loop 1523, including a total number of original dispatches, such as 5, to merge the five original dispatches 1513A-1513E into

a single transformed dispatch 1533 provided in new command sequence 1531 that replaces original command sequence 1511.

In one embodiment, to further transform original command sequence 1511 into new command sequence 1531, building/mapping logic 1405 further proposes including a transformed
5 swapping command, such as swap input/output 1525, into fused kernel 1521 as opposed to leaving it in new command sequence 1531 like a series of swap input/output 1515A-1515D in original command sequence 1511. This intermediary plan may then forwarded on to execution/application logic 1407 of **Figure 14** to be applied and executed for efficient processing of the application by the GPU and other computing resources.

10 For example, as aforementioned with respect to **Figure 14**, having a single transformed dispatch 1533 in new command sequence 1531 along with *for*-loop 1523 and swap input/output 1525 in fused kernel 1521 allows for minimal use of memory resources, such as a first read from an input buffer (such as the first read associated with original D0 1513A) may be performed by accessing a global memory and similarly, a last write from an output buffer (such as the last
15 write associated with original D4 1513E) may be performed by accessing the global memory, where all other reads and writes are performed locally by accessing a shared local cache, significantly reducing the use of memory resources. These other reads and writes may include, for example, a write command associated with original D0 1513A, reads and writes associated with original D1 1513B, D2 1513C, and D3 1513D, and a read command associated with
20 original D4 1513E.

Figure 16A illustrates a method 1600 including compile time processes for efficiently processing a command sequence according to one embodiment. Method 1600 may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof.
25 In one embodiment, method 1600 may be performed by transformation mechanism 1310 of **Figures 13-14**. The processes of method 1600 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 **Figures 1-15** may not be discussed or repeated hereafter.

30 Method 1600 offers a compile time flow of processes beginning at block 1601 with compiling of a compute shader at a computing device and continues, at block 1603, with parsing of the shader. At block 1605, in one embodiment, a determination is made as to whether the shader is transformable for later runtime efficient processing of an application as facilitated by transformation mechanism 1310 of **Figures 13-14**. For example, a transformable shader may be
35 used later, at runtime, as illustrated with respect to method 1650 of **Figure 16B**, where one or

more dispatches of one or more command streams issued through one or more kernels of an application be transformed at runtime for efficient processing at a GPU of the computing device, such as GPU 1314 of computing device 1300 of **Figure 13**. If yes, at block 1607, the shader is transformed into a transformed shader for later runtime processing and, at block 1609, the compilation process of method 1600 ends. Further, if the shader is not to be transformed or capable of being transformed, the compilation process ends at block 1609.

Figure 16B illustrates a method 1650 including runtime processes for efficiently processing a command sequence according to one embodiment. Method 1650 may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof. In one embodiment, method 1650 may be performed by transformation mechanism 1310 of **Figures 13-14**. The processes of method 1650 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 **Figures 1-16A** may not be discussed or repeated hereafter.

Method 1650 offers a runtime flow of processes beginning at block 1651 with the application submitting a GPGPU command to a graphics driver, such as driver logic 1316 of **Figure 13**, for processing. As previously discussed, the command may represent a command stream as defined by a corresponding kernel of the application, where the command stream may include any number and type of dispatches to be processed by the GPU for eventually rendering the contents of the application for the user to view on a display device associated with the computing device.

At block 1653, in one embodiment, the command is saved at a database, such as database 1430, and subsequently, the command is evaluated by evaluation logic 1403 to create and forward an optimization plan that is best suited for the command stream to building/mapping logic 1405 of **Figure 14**. At block 1655, in one embodiment, the optimization plan is received and used by building/mapping logic 1405 to generate an intermediary plan that is based on the optimization plan, as indicated by a continuous, but optional, optimization pass 1661, where the intermediary plan represents a transformed processing plan for the command stream.

At block 1657, in one embodiment, the transformed shader of compile time method 1600 of **Figure 16A** is used with the intermediary plan, where the intermediary plan is applied to form the transformed command stream, based on the transformed shader, and then executed by execution/application logic 1407 of **Figure 14**. At block 1659, upon application and execution at block 1657, the transformed command stream along with any other relevant data, such as a

transformed kernel as illustrated in **Figure 15**, may be submitted to be processed by one or more components of the GPU at the computing device.

References to “one embodiment”, “an embodiment”, “example embodiment”, “various embodiments”, etc., indicate that the embodiment(s) so described may include particular
5 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
10 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
15 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
20 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
25 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 dynamic
30 runtime transformation of graphics processing commands for improved graphics performance on computing devices, comprising: reception/detection logic to detect a command stream associated with an application, wherein the command stream includes dispatches; evaluation logic to evaluate processing parameters relating to each of the dispatches, wherein the evaluation logic is further to associate a first plan with one or more of the dispatches to transform the command
35 stream into a transformed command stream; building/mapping logic to associate, based on the

first plan, a second plan to the one or more of the dispatches, wherein the second plan represents the transformed command stream; and execution/application logic to execute the second plan, wherein execution of the second plan includes processing the transformed command stream in lieu of the command stream.

5 Example 2 includes the subject matter of Example 1, wherein the first plan comprises an optimization plan, and wherein the second plan comprises an intermediary plan.

 Example 3 includes the subject matter of Example 1, wherein the command stream issued by the application is received at a graphics driver of a graphics processing unit (GPU), and wherein the transformed command stream is executed at the GPU via the graphics driver,
10 wherein the application represents a workload having a plurality of frames, and wherein a dispatch represents a work unit of the workload such that the work unit represents a portion of an image represented a corresponding frame.

 Example 4 includes the subject matter of Example 1 or 3, wherein the command stream corresponds to a kernel of the application, wherein the first plan is further to transform the kernel
15 into a transformed kernel having instructions corresponding to the transformed command stream.

 Example 5 includes the subject matter of Example 1, wherein the processing parameters comprise one or more characteristics relating to execution of the command stream, wherein the one or more characteristics include at least one of size of the dispatches, processing order of the dispatches, number of threads in a group of threads associated with each dispatch, number of
20 groups of threads associated with each dispatch, memory usage, cache usage, bottlenecking locations or reasons, amounts and types and frequencies of hardware allocation, compression format types, shader constraints, and shader execution dispatch width.

 Example 6 includes the subject matter of Example 1 or 5, wherein transforming the command stream into the transformed command stream comprises at least one of changing
25 processing orders of the dispatches, resizing the dispatches, building dependency graphs relating to the dispatches, and mapping one or more of the dispatches from a main memory to a local cache, wherein mapping includes remapping read commands or write commands associated with the one or more of the dispatches from the main memory to the local cache.

 Example 7 includes the subject matter of Example 6, wherein resizing the dispatches
30 comprises increasing a number of the dispatches by merging smaller of the dispatches into bigger dispatches, or reducing the number of the dispatches by breaking bigger of the dispatches into smaller dispatches.

 Example 8 includes the subject matter of Example 1, further comprising a database to store one or more of the command stream, the transformed command stream, the first plan, and

the second plan, wherein the database includes a Cloud-based storage accessible over one or more networks.

Some embodiments pertain to Example 9 that includes a method for facilitating dynamic runtime transformation of graphics processing commands for improved graphics performance on computing devices, comprising: detecting a command stream associated with an application, wherein the command stream includes dispatches; evaluating processing parameters relating to each of the dispatches, wherein evaluating further includes associating a first plan with one or more of the dispatches to transform the command stream into a transformed command stream; associating, based on the first plan, a second plan to the one or more of the dispatches, wherein the second plan represents the transformed command stream; and executing the second plan, wherein execution of the second plan includes processing the transformed command stream in lieu of the command stream.

Example 10 includes the subject matter of Example 9, wherein the first plan comprises an optimization plan, and wherein the second plan comprises an intermediary plan.

Example 11 includes the subject matter of Example 9, wherein the command stream issued by the application is received at a graphics driver of a graphics processing unit (GPU), and wherein the transformed command stream is executed at the GPU via the graphics driver, wherein the application represents a workload having a plurality of frames, and wherein a dispatch represents a work unit of the workload such that the work unit represents a portion of an image represented a corresponding frame.

Example 12 includes the subject matter of Example 9 or 11, wherein the command stream corresponds to a kernel of the application, wherein the first plan is further to transform the kernel into a transformed kernel having instructions corresponding to the transformed command stream.

Example 13 includes the subject matter of Example 9, wherein the processing parameters comprise one or more characteristics relating to execution of the command stream, wherein the one or more characteristics include at least one of size of the dispatches, processing order of the dispatches, number of threads in a group of threads associated with each dispatch, number of groups of threads associated with each dispatch, memory usage, cache usage, bottlenecking locations or reasons, amounts and types and frequencies of hardware allocation, compression format types, shader constraints, and shader execution dispatch width.

Example 14 includes the subject matter of Example 9 or 13, wherein transforming the command stream into the transformed command stream comprises at least one of changing processing orders of the dispatches, resizing the dispatches, building dependency graphs relating to the dispatches, and mapping one or more of the dispatches from a main memory to a local

cache, wherein mapping includes remapping read commands or write commands associated with the one or more of the dispatches from the main memory to the local cache.

Example 15 includes the subject matter of Example 14, wherein resizing the dispatches comprises increasing a number of the dispatches by merging smaller of the dispatches into bigger dispatches, or reducing the number of the dispatches by breaking bigger of the dispatches into smaller dispatches.

Example 16 includes the subject matter of Example 9, further comprising storing, at a database, one or more of the command stream, the transformed command stream, the first plan, and the second plan, wherein the database includes a Cloud-based storage accessible over one or more networks.

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 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 command stream associated with an application, wherein the command stream includes dispatches; evaluating processing parameters relating to each of the dispatches, wherein evaluating further includes associating a first plan with one or more of the dispatches to transform the command stream into a transformed command stream; associating, based on the first plan, a second plan to the one or more of the dispatches, wherein the second plan represents the transformed command stream; and executing the second plan, wherein execution of the second plan includes processing the transformed command stream in lieu of the command stream.

Example 24 includes the subject matter of Example 23, wherein the first plan comprises an optimization plan, and wherein the second plan comprises an intermediary plan.

Example 25 includes the subject matter of Example 23, wherein the command stream issued by the application is received at a graphics driver of a graphics processing unit (GPU),
5 and wherein the transformed command stream is executed at the GPU via the graphics driver, wherein the application represents a workload having a plurality of frames, and wherein a dispatch represents a work unit of the workload such that the work unit represents a portion of an image represented a corresponding frame.

Example 26 includes the subject matter of Example 23, wherein the command stream
10 corresponds to a kernel of the application, wherein the first plan is further to transform the kernel into a transformed kernel having instructions corresponding to the transformed command stream.

Example 27 includes the subject matter of Example 23, wherein the processing parameters comprise one or more characteristics relating to execution of the command stream, wherein the one or more characteristics include at least one of size of the dispatches, processing
15 order of the dispatches, number of threads in a group of threads associated with each dispatch, number of groups of threads associated with each dispatch, memory usage, cache usage, bottlenecking locations or reasons, amounts and types and frequencies of hardware allocation, compression format types, shader constraints, and shader execution dispatch width.

Example 28 includes the subject matter of Example 23, wherein transforming the
20 command stream into the transformed command stream comprises at least one of changing processing orders of the dispatches, resizing the dispatches, building dependency graphs relating to the dispatches, and mapping one or more of the dispatches from a main memory to a local cache, wherein mapping includes remapping read commands or write commands associated with the one or more of the dispatches from the main memory to the local cache.

Example 29 includes the subject matter of Example 28, wherein resizing the dispatches
25 comprises increasing a number of the dispatches by merging smaller of the dispatches into bigger dispatches, or reducing the number of the dispatches by breaking bigger of the dispatches into smaller dispatches.

Example 30 includes the subject matter of Example 23, wherein the one or more
30 operations further comprise storing, at a database, one or more of the command stream, the transformed command stream, the first plan, and the second plan, wherein the database includes a Cloud-based storage accessible over one or more networks.

Some embodiments pertain to Example 31 includes an apparatus comprising: means for
35 detecting a command stream associated with an application, wherein the command stream includes dispatches; means for evaluating processing parameters relating to each of the

dispatches, wherein evaluating further includes associating a first plan with one or more of the dispatches to transform the command stream into a transformed command stream; means for associating, based on the first plan, a second plan to the one or more of the dispatches, wherein the second plan represents the transformed command stream; and means for executing the
5 second plan, wherein execution of the second plan includes processing the transformed command stream in lieu of the command stream.

Example 32 includes the subject matter of Example 31, wherein the first plan comprises an optimization plan, and wherein the second plan comprises an intermediary plan.

Example 33 includes the subject matter of Example 31, wherein the command stream
10 issued by the application is received at a graphics driver of a graphics processing unit (GPU), and wherein the transformed command stream is executed at the GPU via the graphics driver, wherein the application represents a workload having a plurality of frames, and wherein a dispatch represents a work unit of the workload such that the work unit represents a portion of an image represented a corresponding frame.

Example 34 includes the subject matter of Example 31, wherein the command stream
15 corresponds to a kernel of the application, wherein the first plan is further to transform the kernel into a transformed kernel having instructions corresponding to the transformed command stream.

Example 35 includes the subject matter of Example 31, wherein the processing
parameters comprise one or more characteristics relating to execution of the command stream,
20 wherein the one or more characteristics include at least one of size of the dispatches, processing order of the dispatches, number of threads in a group of threads associated with each dispatch, number of groups of threads associated with each dispatch, memory usage, cache usage, bottlenecking locations or reasons, amounts and types and frequencies of hardware allocation, compression format types, shader constraints, and shader execution dispatch width.

Example 36 includes the subject matter of Example 31, wherein means for transforming
25 the command stream into the transformed command stream comprises at least one of means for changing processing orders of the dispatches, means for resizing the dispatches, means for building dependency graphs relating to the dispatches, and means for mapping one or more of the dispatches from a main memory to a local cache, wherein mapping includes remapping read
30 commands or write commands associated with the one or more of the dispatches from the main memory to the local cache.

Example 37 includes the subject matter of Example 36, wherein means for resizing the
dispatches comprises means for increasing a number of the dispatches by merging smaller of the dispatches into bigger dispatches, or means for reducing the number of the dispatches by
35 breaking bigger of the dispatches into smaller dispatches.

Example 38 includes the subject matter of Example 31, further comprising means for storing, at a database, one or more of the command stream, the transformed command stream, the first plan, and the second plan, wherein the database includes a Cloud-based storage accessible over one or more networks.

5 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.

10 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.

15 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.

20 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, 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;
25 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 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.

30

CLAIMS

What is claimed is:

1. An apparatus to facilitate dynamic runtime transformation of graphics processing commands, comprising:

5 reception/detection logic to detect a command stream associated with an application, wherein the command stream includes dispatches;

evaluation logic to evaluate processing parameters relating to each of the dispatches, wherein the evaluation logic is further to associate a first plan with one or more of the dispatches to transform the command stream into a transformed command stream;

10 building/mapping logic to associate, based on the first plan, a second plan to the one or more of the dispatches, wherein the second plan represents the transformed command stream; and

execution/application logic to execute the second plan, wherein execution of the second plan includes processing the transformed command stream in lieu of the command stream.

15 2. The apparatus of claim 1, wherein the first plan comprises an optimization plan, and wherein the second plan comprises an intermediary plan.

3. The apparatus of claim 1, wherein the command stream issued by the application is received at a graphics driver of a graphics processing unit (GPU), and wherein the transformed command stream is executed at the GPU via the graphics driver, wherein the application
20 represents a workload having a plurality of frames, and wherein a dispatch represents a work unit of the workload such that the work unit represents a portion of an image represented a corresponding frame.

4. The apparatus of claim 1 or 3, wherein the command stream corresponds to a kernel of the application, wherein the first plan is further to transform the kernel into a
25 transformed kernel having instructions corresponding to the transformed command stream.

5. The apparatus of claim 1, wherein the processing parameters comprise one or more characteristics relating to execution of the command stream, wherein the one or more characteristics include at least one of size of the dispatches, processing order of the dispatches, number of threads in a group of threads associated with each dispatch, number of groups of
30 threads associated with each dispatch, memory usage, cache usage, bottlenecking locations or reasons, amounts and types and frequencies of hardware allocation, compression format types, shader constraints, and shader execution dispatch width.

6. The apparatus of claim 1 or 5, wherein transforming the command stream into the transformed command stream comprises at least one of changing processing orders of the
35 dispatches, resizing the dispatches, building dependency graphs relating to the dispatches, and

mapping one or more of the dispatches from a main memory to a local cache, wherein mapping includes remapping read commands or write commands associated with the one or more of the dispatches from the main memory to the local cache.

7. The apparatus of claim 6, wherein resizing the dispatches comprises increasing a number of the dispatches by merging smaller of the dispatches into bigger dispatches, or reducing the number of the dispatches by breaking bigger of the dispatches into smaller dispatches.

8. The apparatus of claim 1, further comprising a database to store one or more of the command stream, the transformed command stream, the first plan, and the second plan, wherein the database includes a Cloud-based storage accessible over one or more networks.

9. A method for facilitating dynamic runtime transformation of graphics processing commands, comprising:

detecting a command stream associated with an application, wherein the command stream includes dispatches;

evaluating processing parameters relating to each of the dispatches, wherein evaluating further includes associating a first plan with one or more of the dispatches to transform the command stream into a transformed command stream;

associating, based on the first plan, a second plan to the one or more of the dispatches, wherein the second plan represents the transformed command stream; and

executing the second plan, wherein execution of the second plan includes processing the transformed command stream in lieu of the command stream.

10. The method of claim 9, wherein the first plan comprises an optimization plan, and wherein the second plan comprises an intermediary plan.

11. The method of claim 9, wherein the command stream issued by the application is received at a graphics driver of a graphics processing unit (GPU), and wherein the transformed command stream is executed at the GPU via the graphics driver, wherein the application represents a workload having a plurality of frames, and wherein a dispatch represents a work unit of the workload such that the work unit represents a portion of an image represented a corresponding frame.

12. The method of claim 9, wherein the command stream corresponds to a kernel of the application, wherein the first plan is further to transform the kernel into a transformed kernel having instructions corresponding to the transformed command stream.

13. The method of claim 9, wherein the processing parameters comprise one or more characteristics relating to execution of the command stream, wherein the one or more characteristics include at least one of size of the dispatches, processing order of the dispatches,

number of threads in a group of threads associated with each dispatch, number of groups of threads associated with each dispatch, memory usage, cache usage, bottlenecking locations or reasons, amounts and types and frequencies of hardware allocation, compression format types, shader constraints, and shader execution dispatch width.

5 14. The method of claim 9, wherein transforming the command stream into the transformed command stream comprises at least one of changing processing orders of the dispatches, resizing the dispatches, building dependency graphs relating to the dispatches, and mapping one or more of the dispatches from a main memory to a local cache, wherein mapping includes remapping read commands or write commands associated with the one or more of the
10 dispatches from the main memory to the local cache.

 15. The method of claim 14, wherein resizing the dispatches comprises increasing a number of the dispatches by merging smaller of the dispatches into bigger dispatches, or reducing the number of the dispatches by breaking bigger of the dispatches into smaller dispatches.

15 16. The method of claim 9, further comprising storing, at a database, one or more of the command stream, the transformed command stream, the first plan, and the second plan, wherein the database includes a Cloud-based storage accessible over one or more networks.

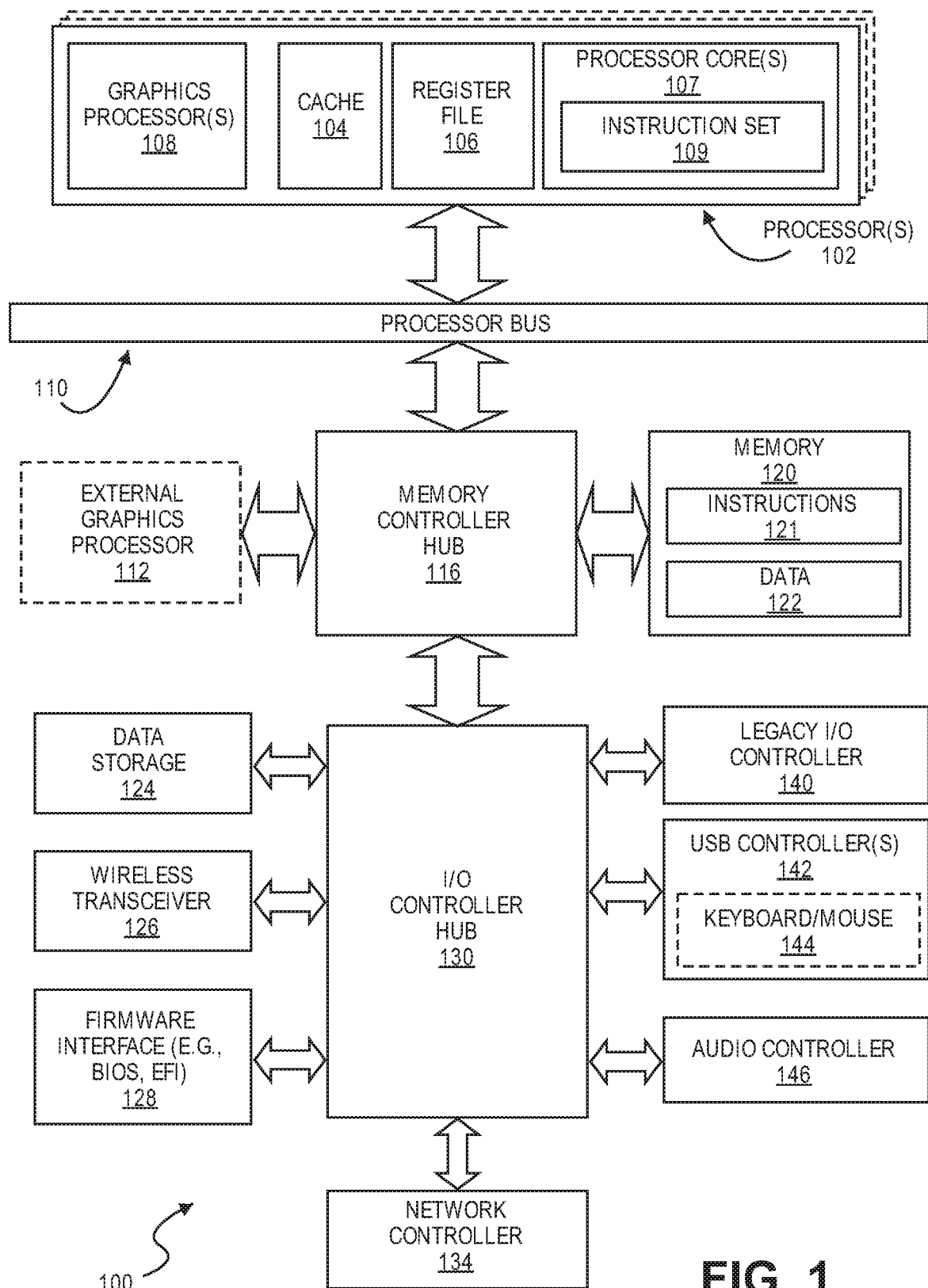
 17. At least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of
20 claims 9-16.

 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.

25 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.

**FIG. 1**

PROCESSOR
200

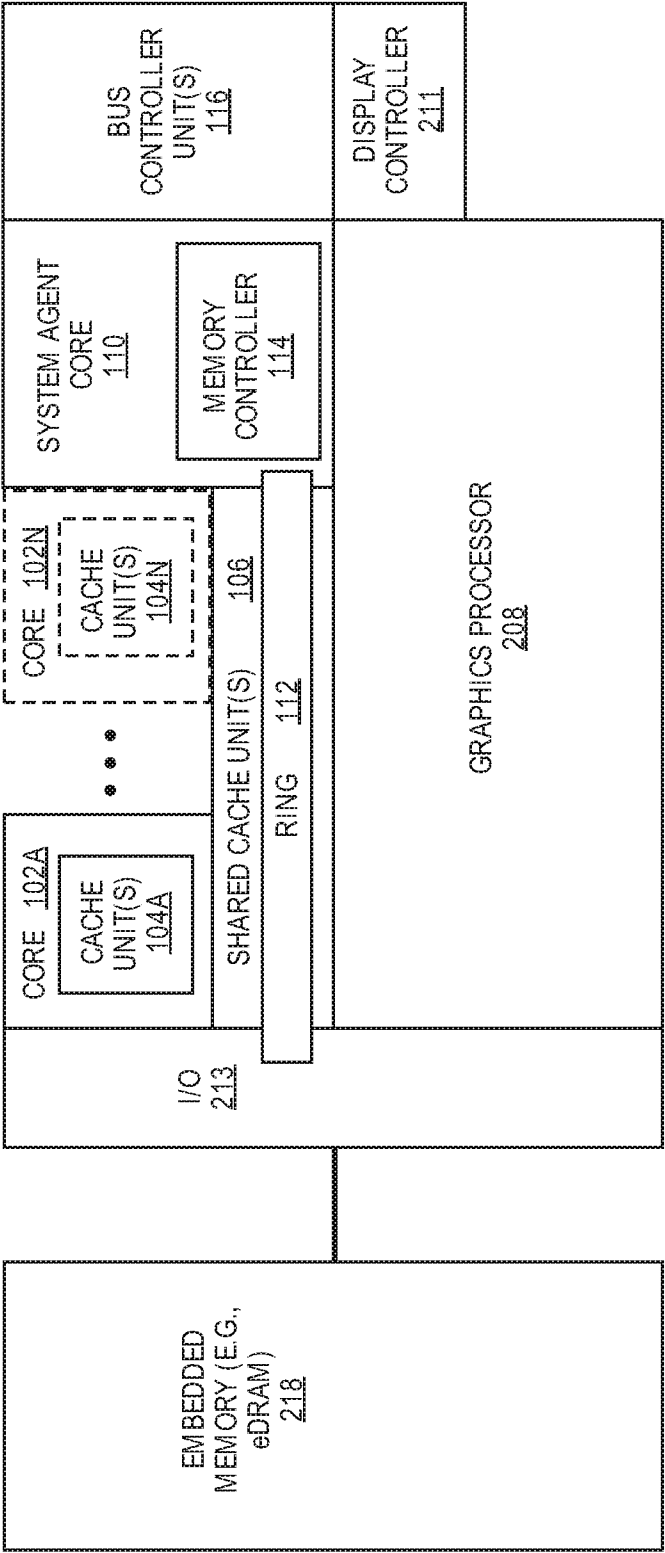


FIG. 2

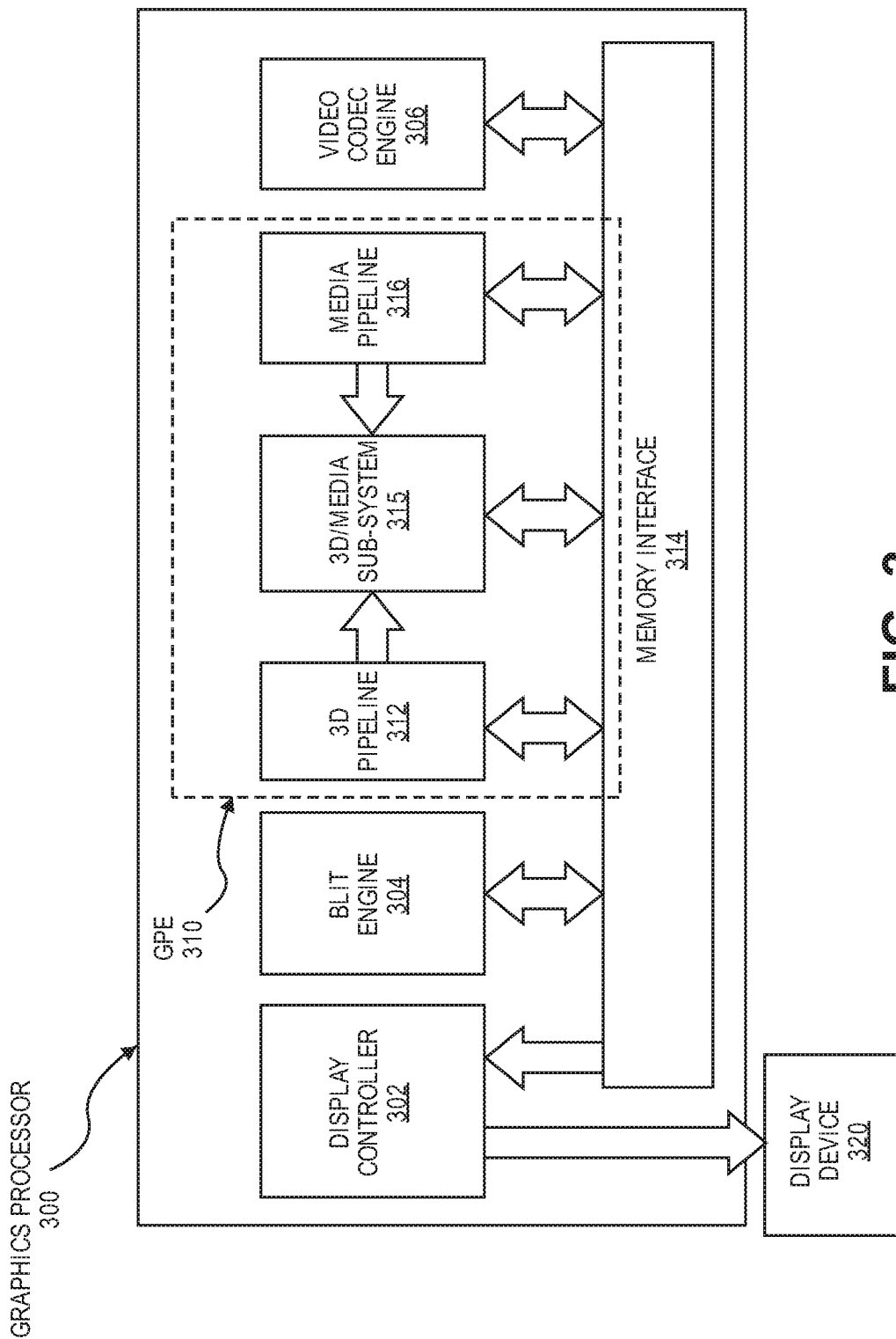


FIG. 3

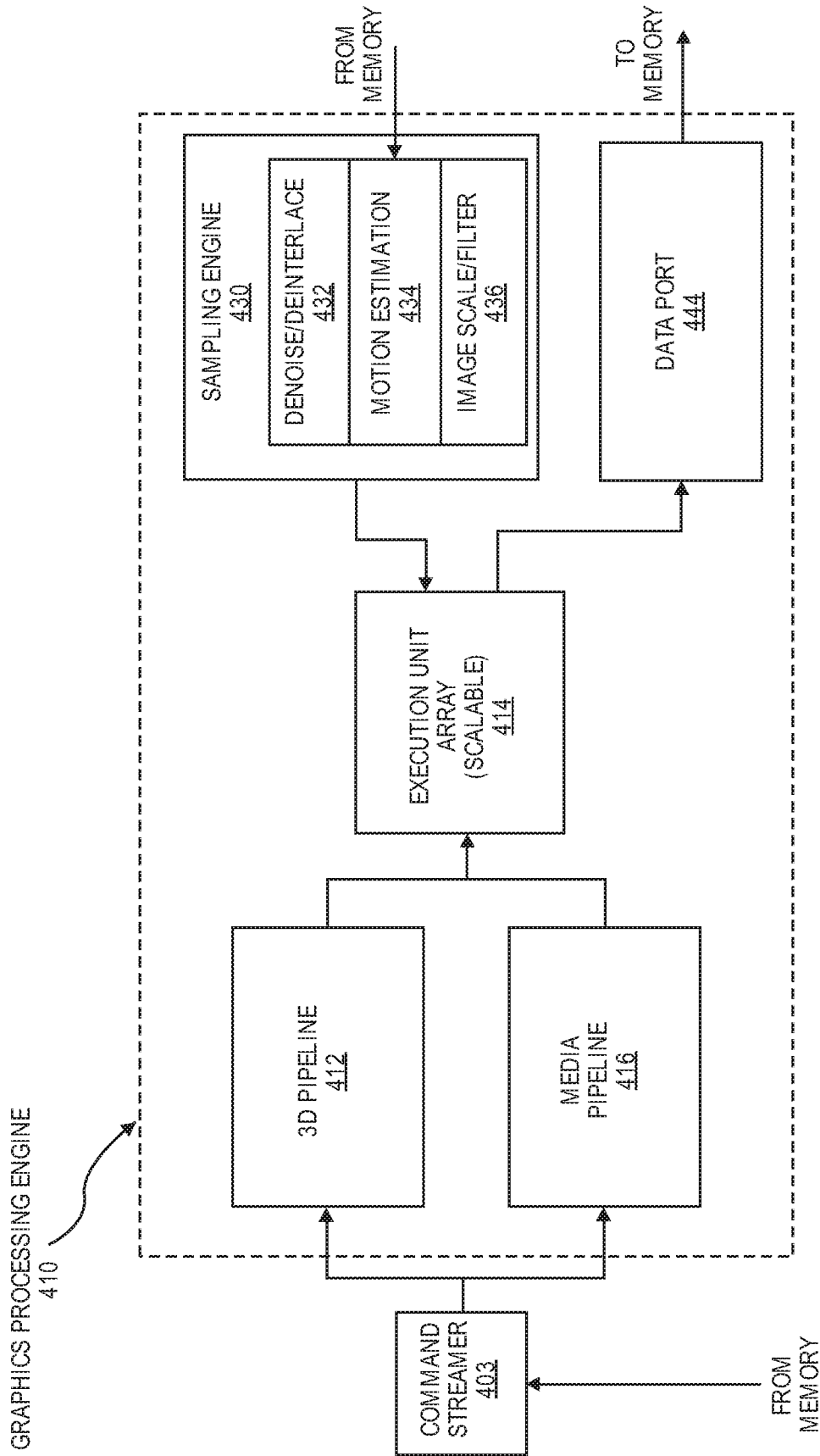


FIG. 4

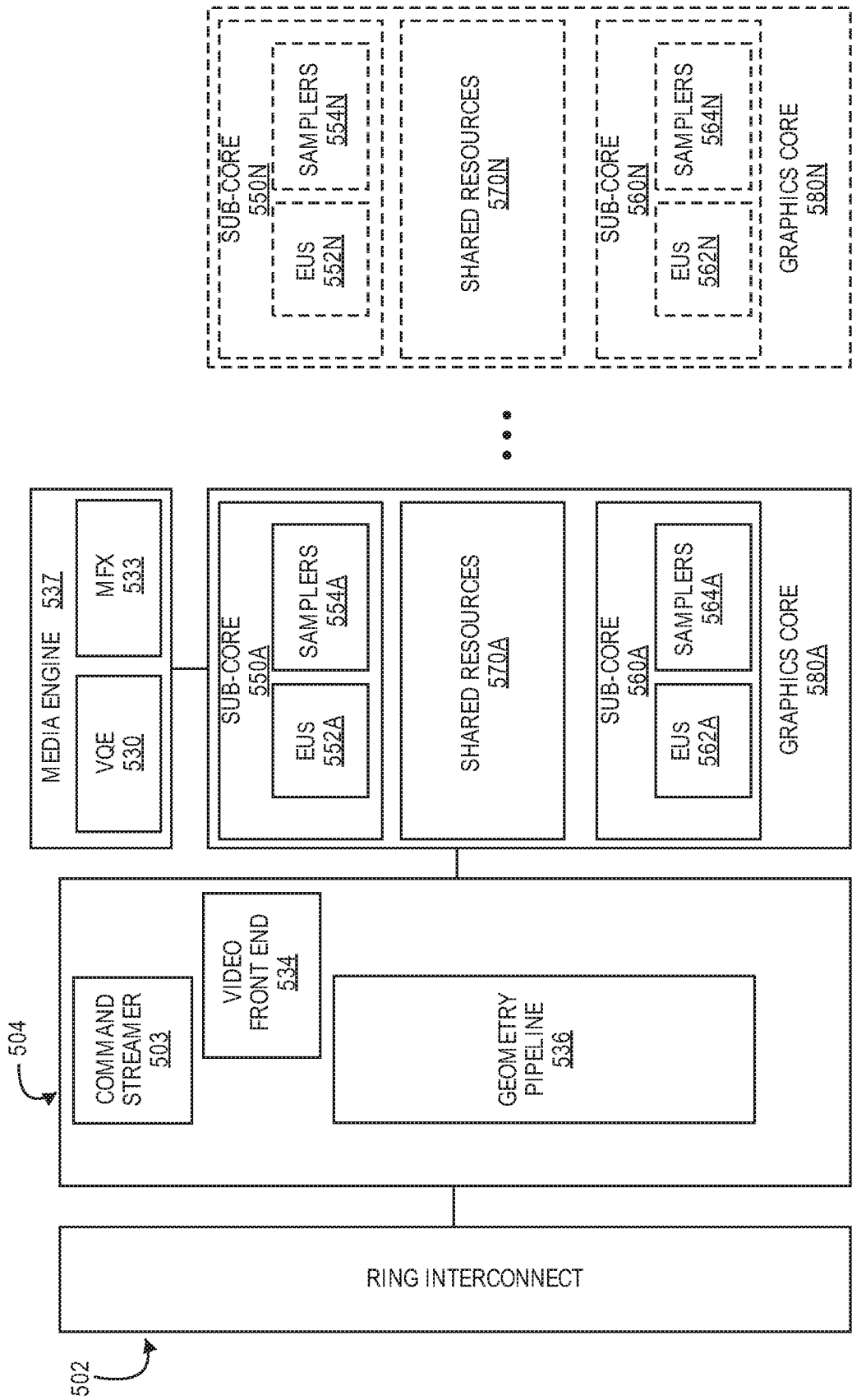


FIG. 5

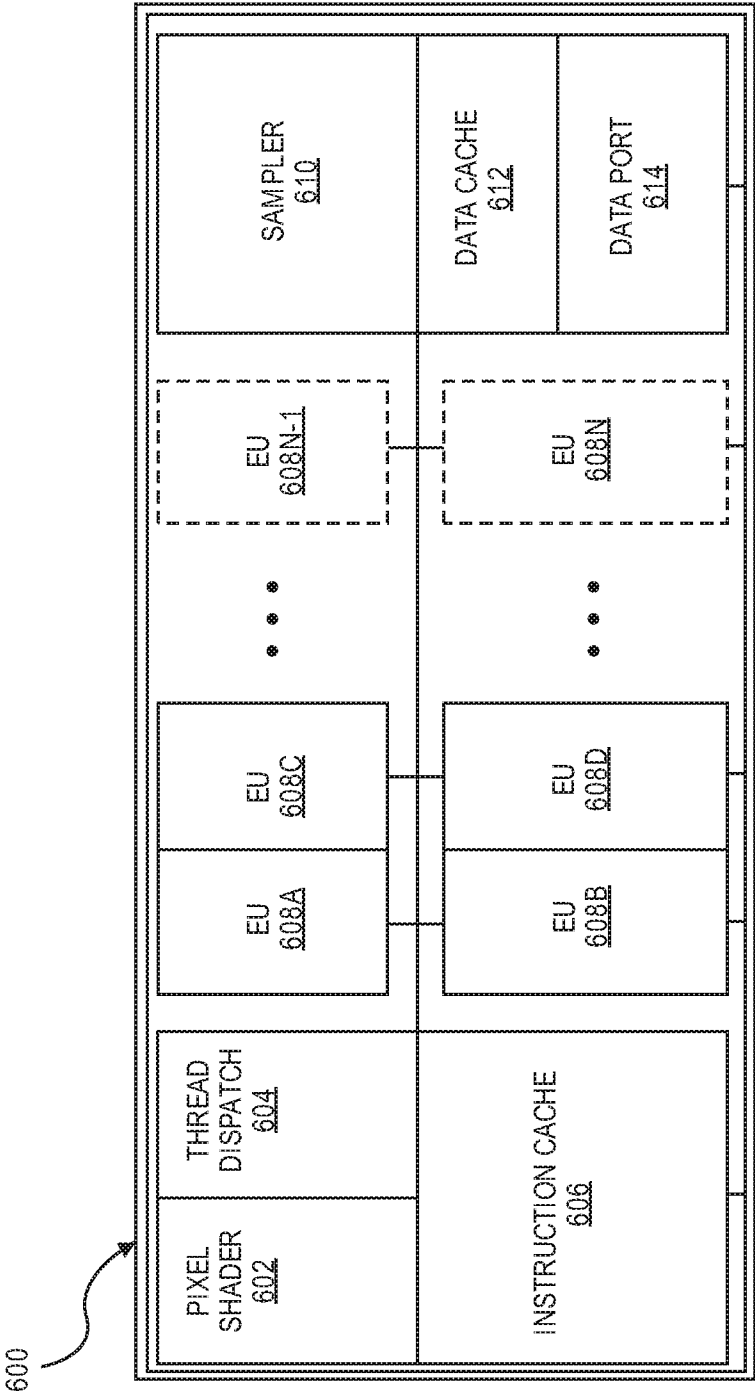
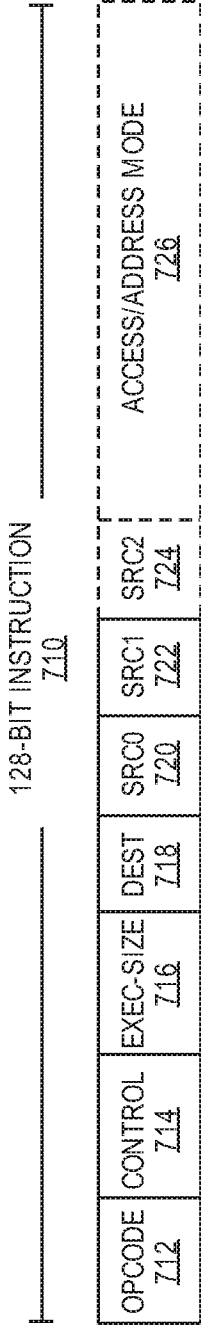


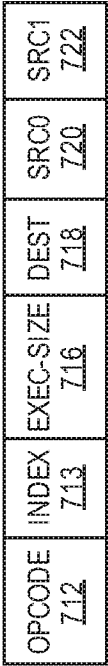
FIG. 6

GRAPHICS CORE INSTRUCTION FORMATS

700



64-BIT COMPACT INSTRUCTION 730



OPCODE DECODE 740

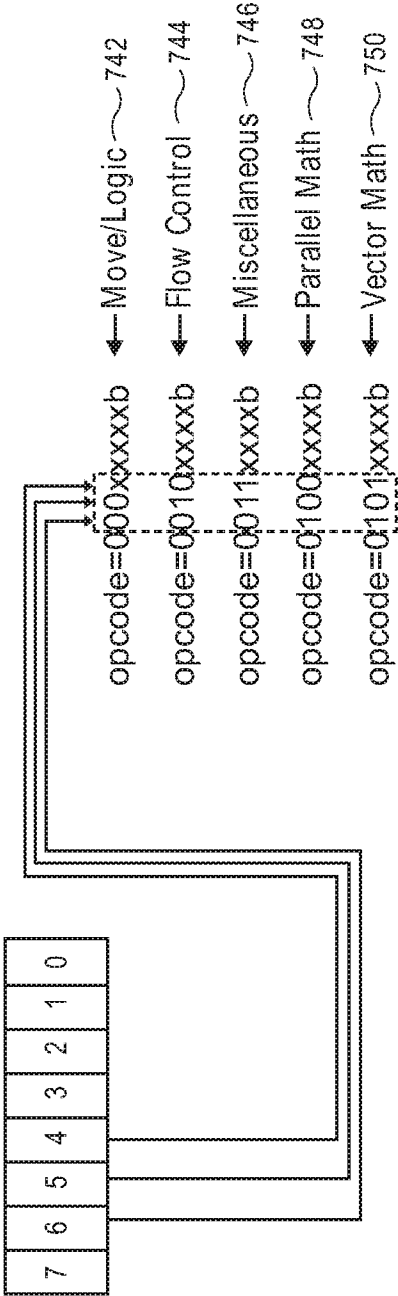


FIG. 7

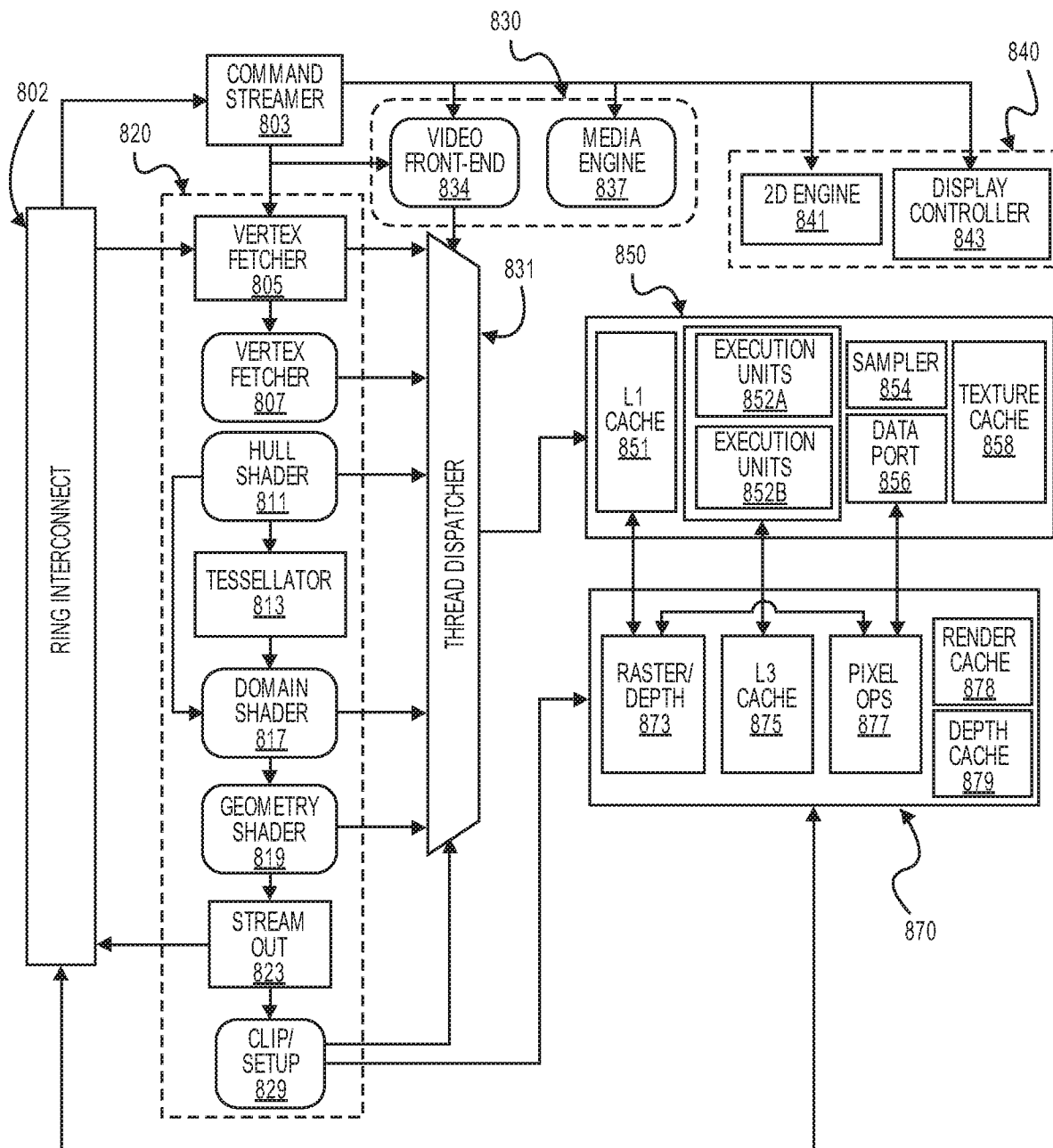


FIG. 8

FIG. 9A

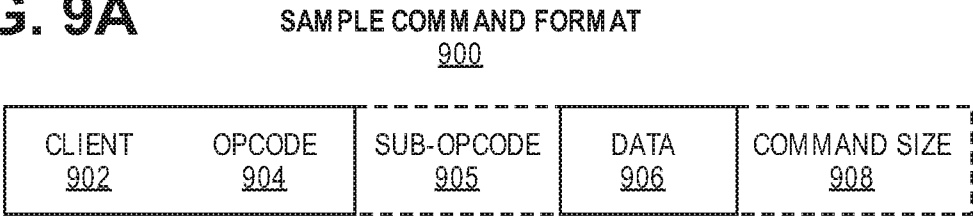
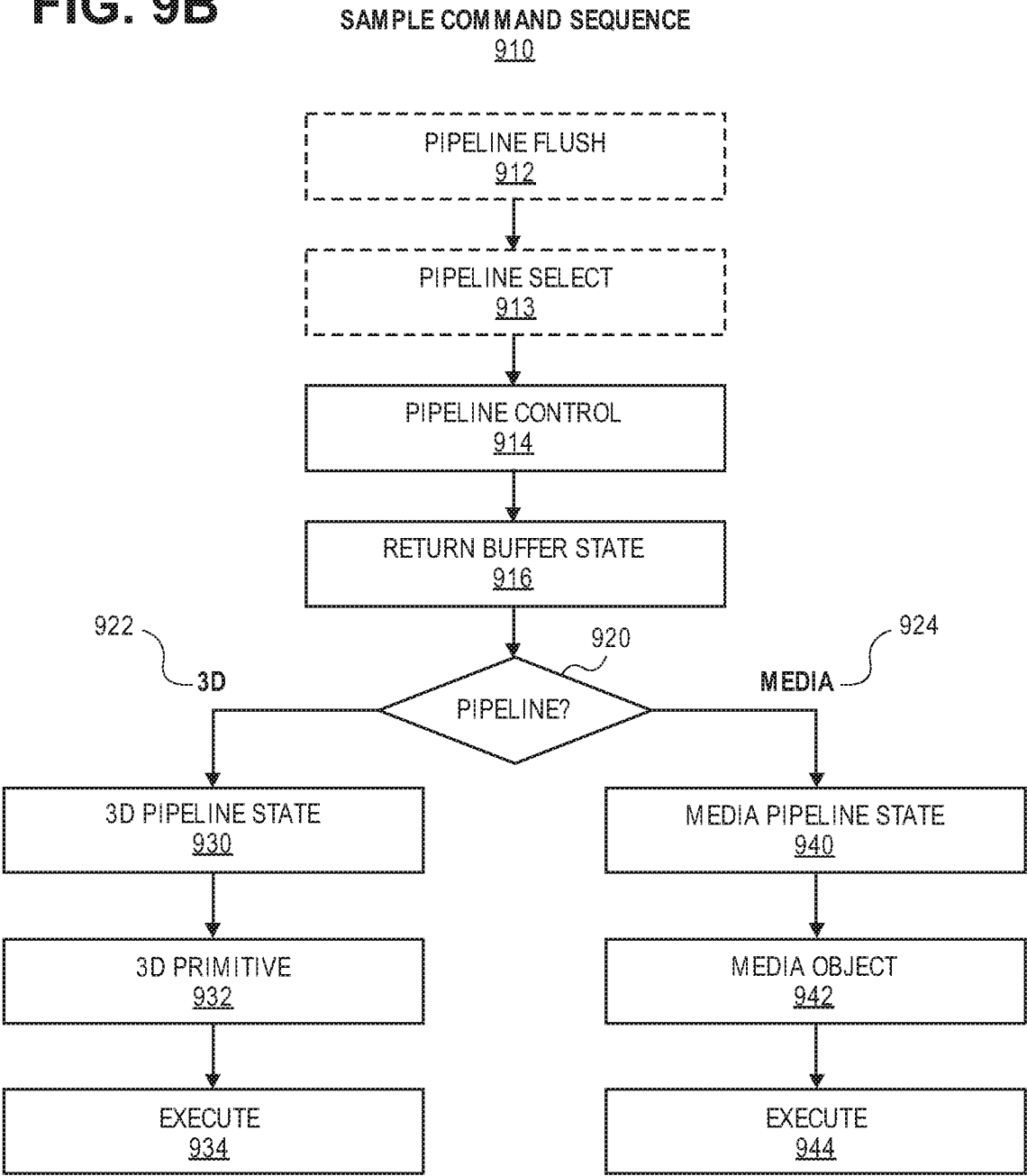
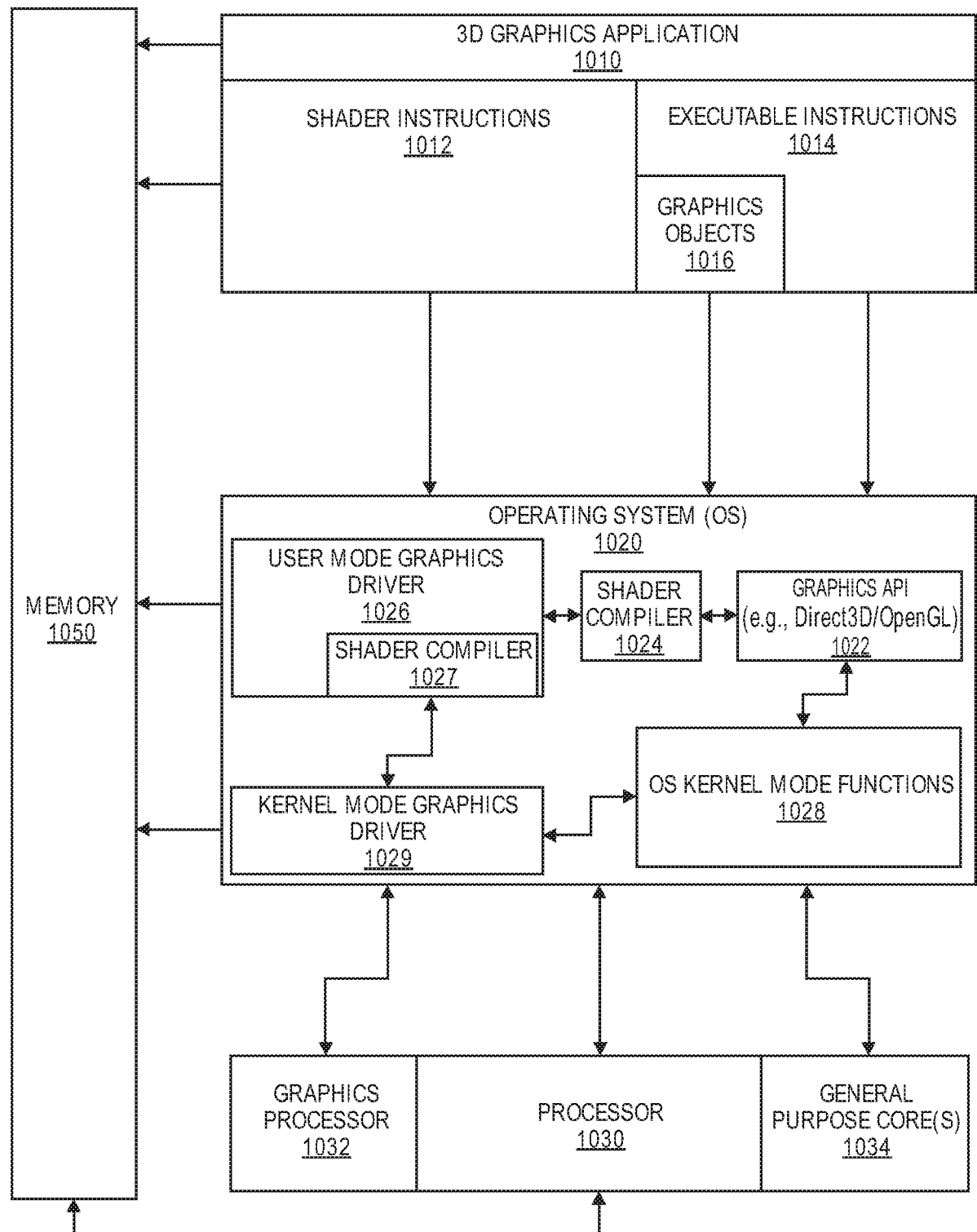


FIG. 9B



**FIG. 10**

IP CORE DEVELOPMENT - 1100

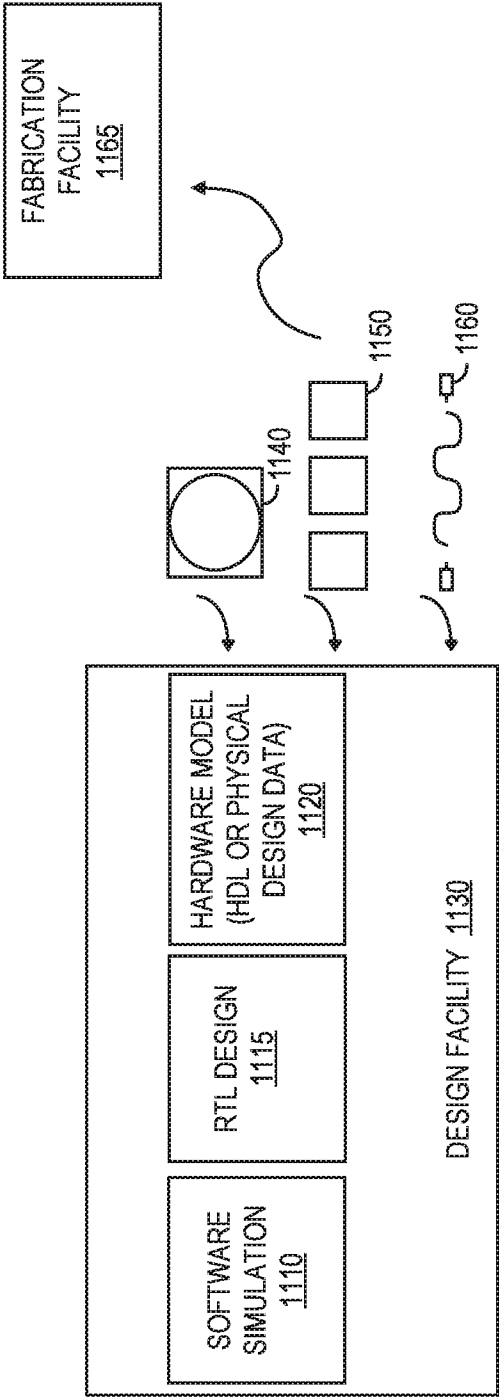
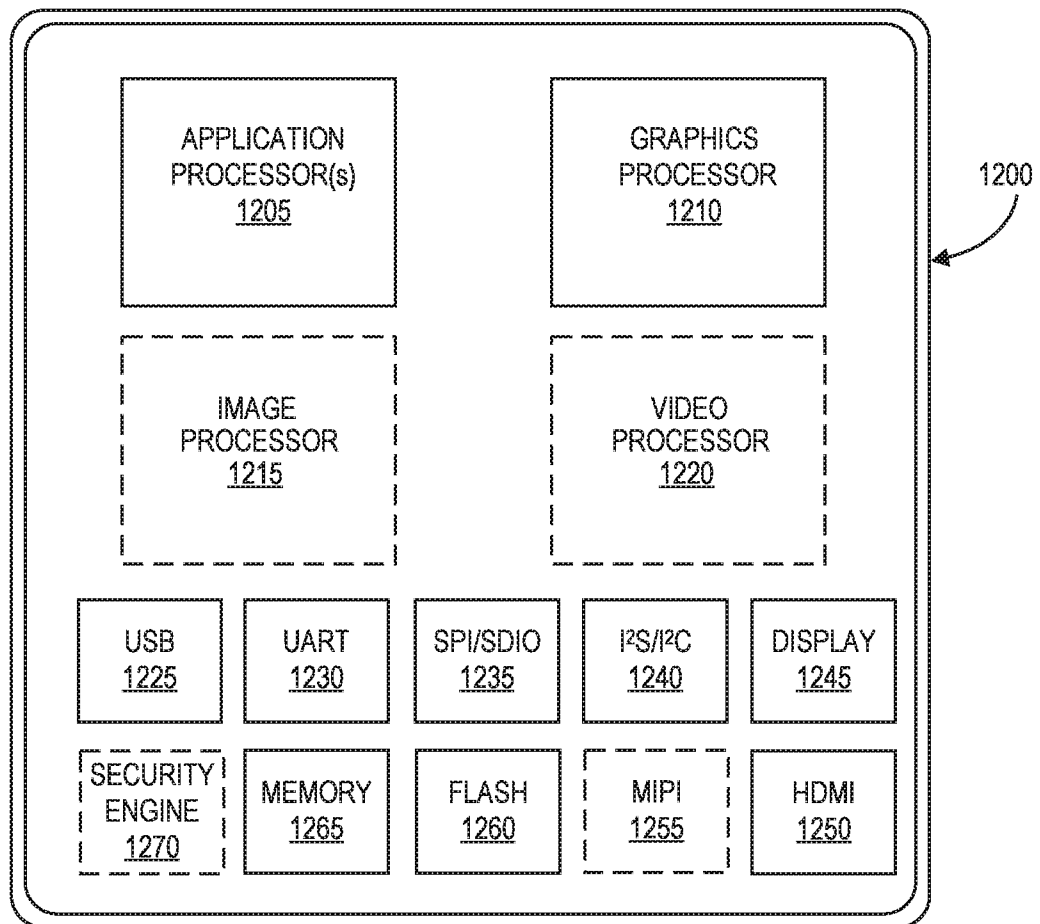


FIG. 11

**FIG. 12**

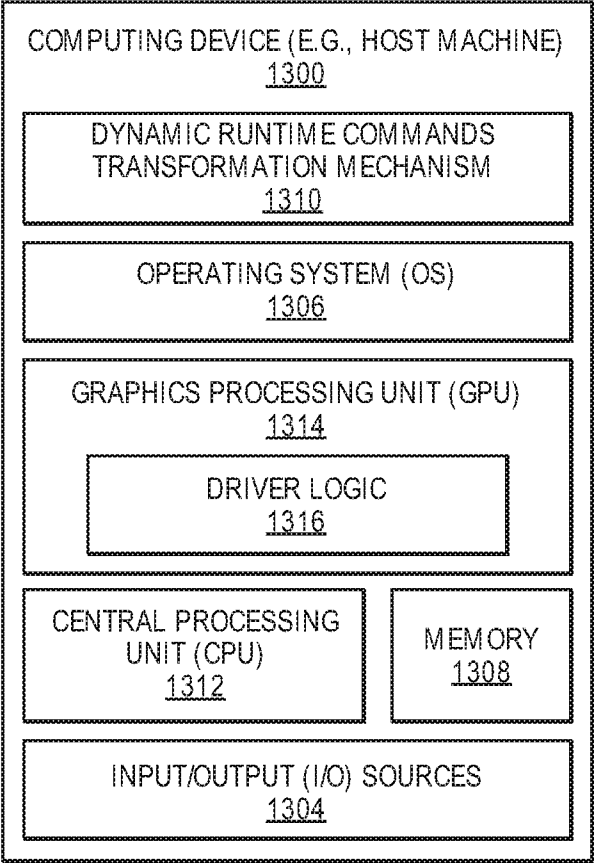
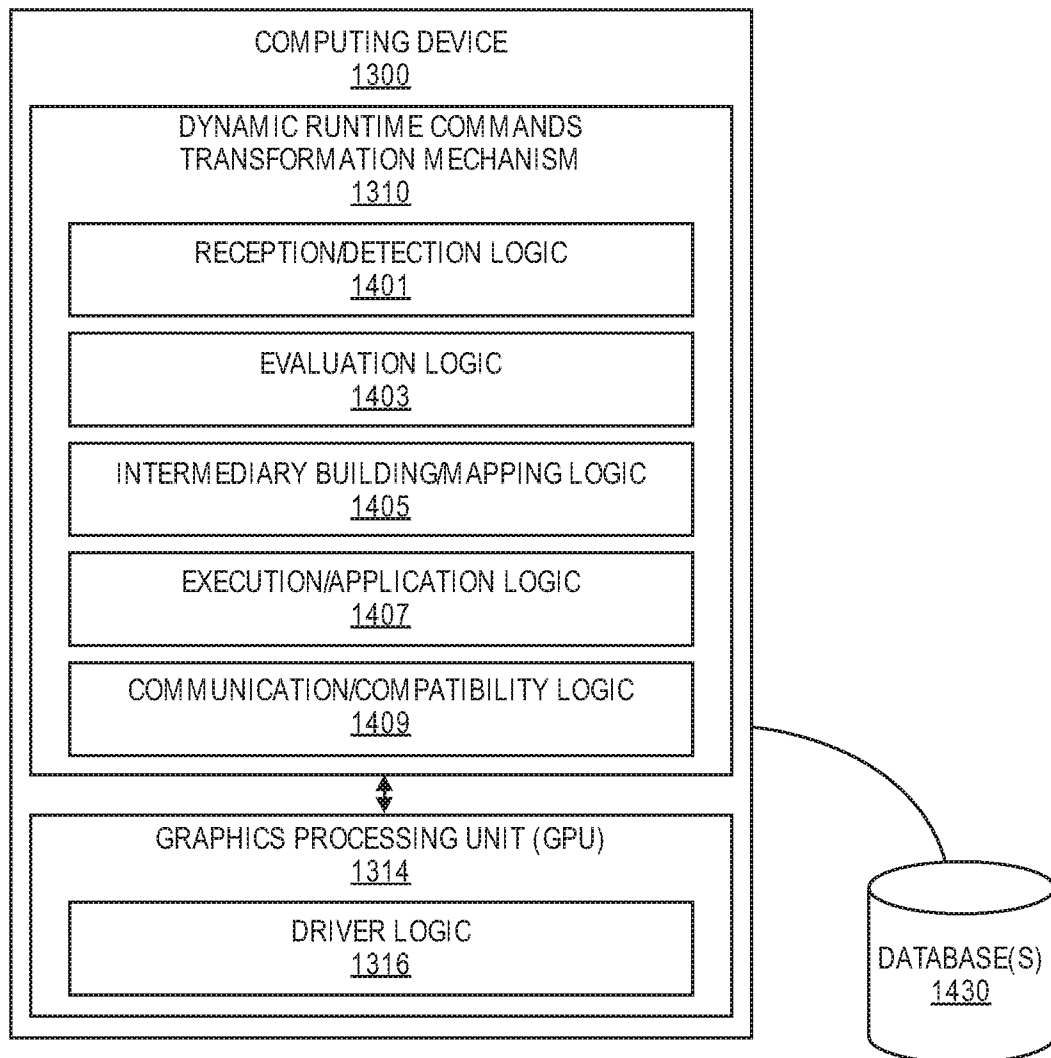


FIG. 13

**FIG. 14A**

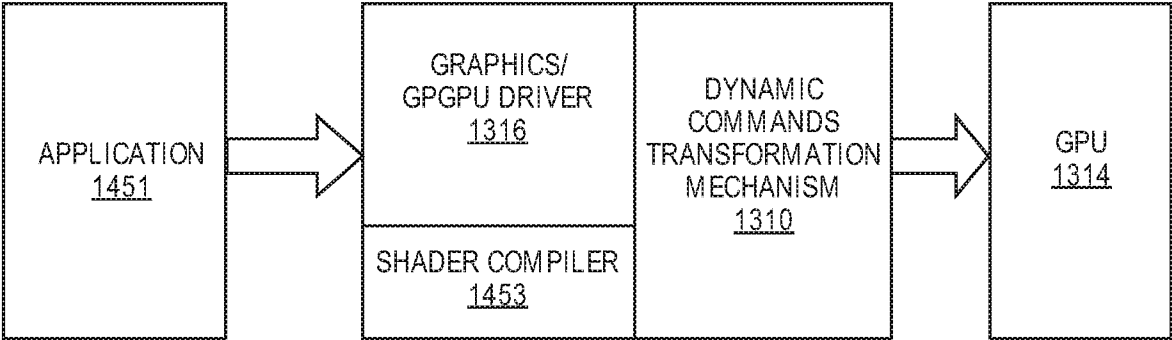


FIG. 14B

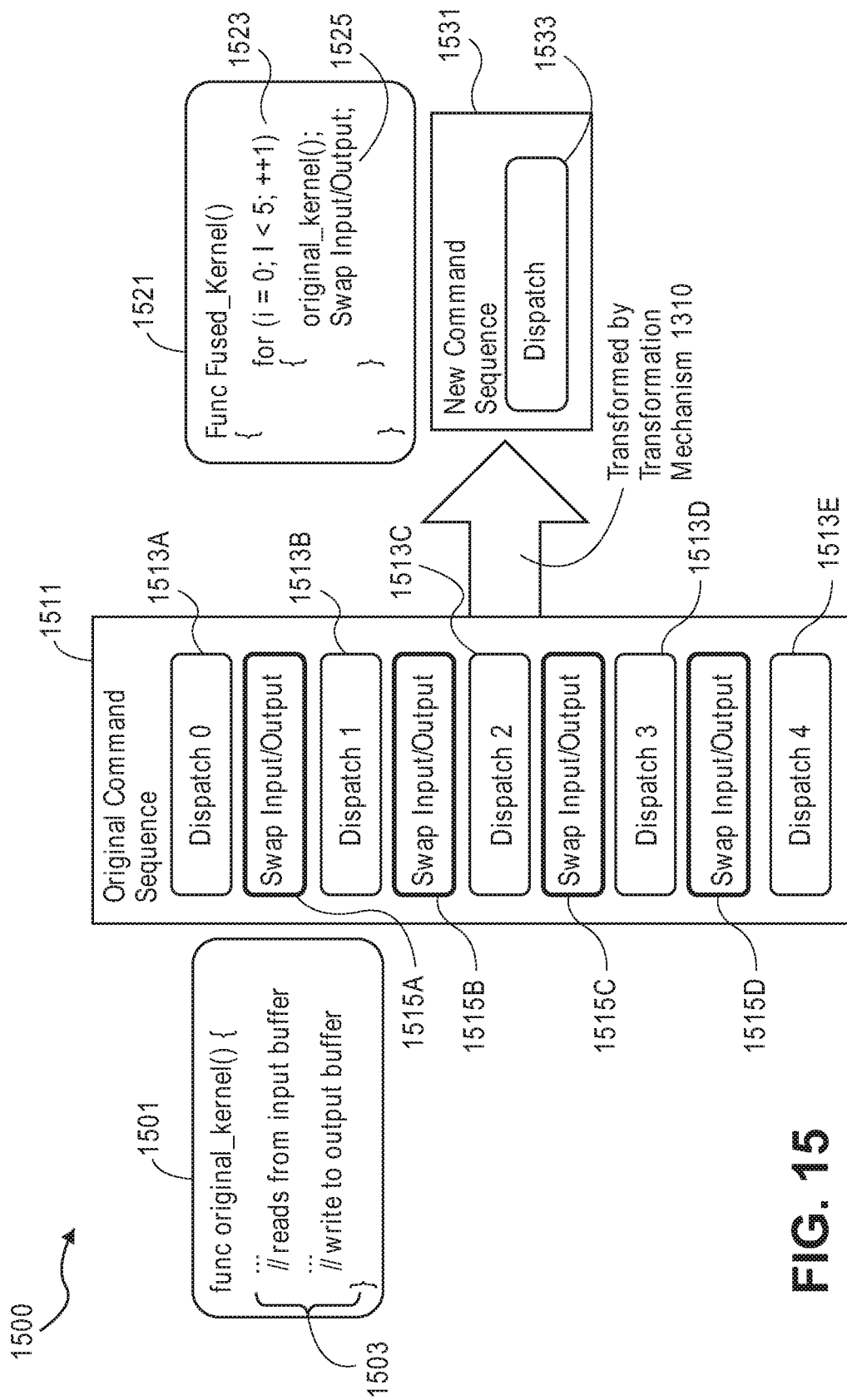
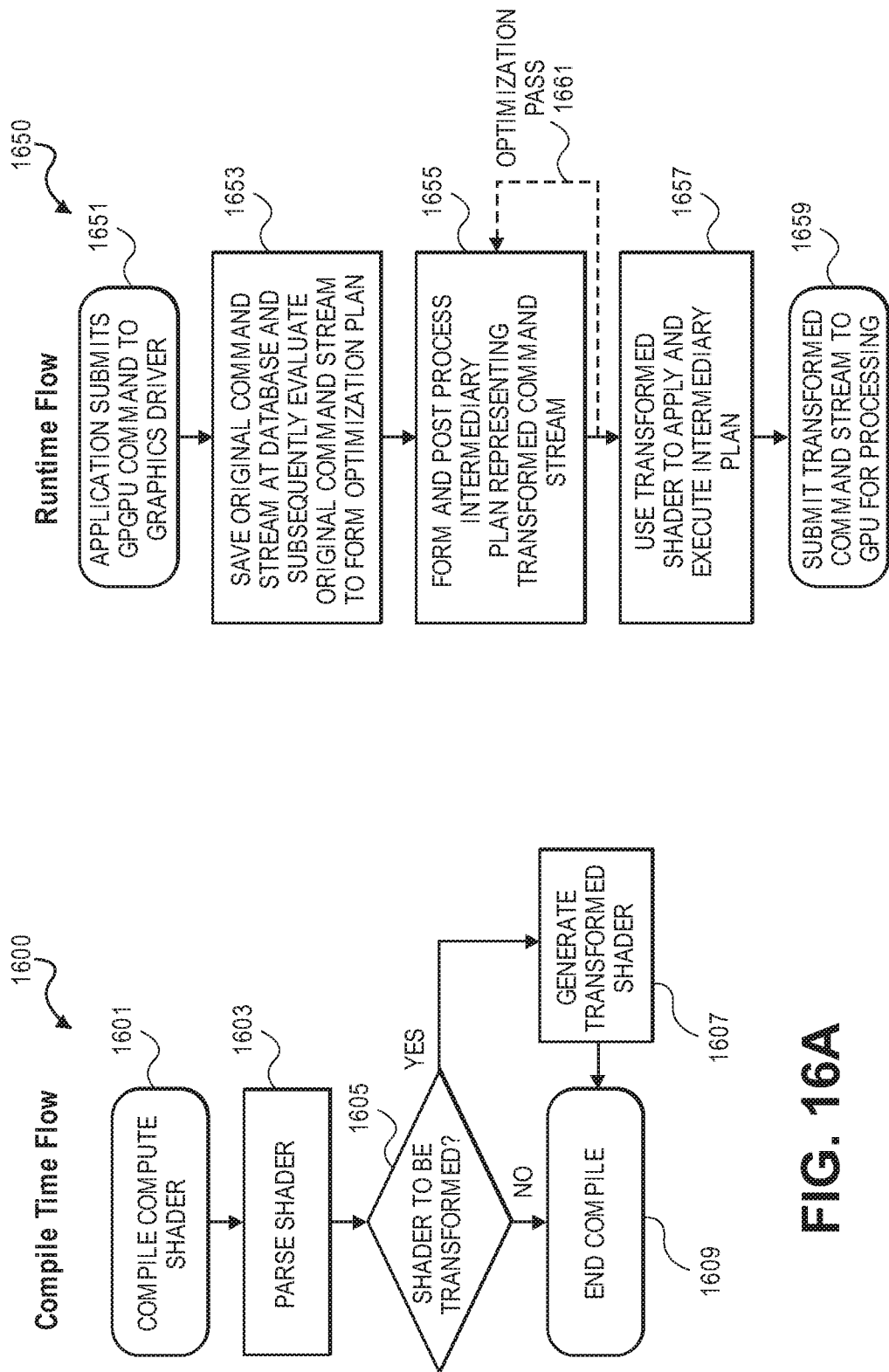


FIG. 15

**FIG. 16A****FIG. 16B**

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/031679**A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/38(2006.01)i, G06F 9/30(2006.01)i, G06T 1/20(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 9/38; G06T 1/20; G06F 13/14; G09G 5/00; G06F 9/30

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: graphics processing command, parameter, dispatch, transform, GPU, frame, graphics driver, resize, command stream

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2007-0052723 A1 (SRIRAM SUBRAMANIAN et al.) 08 March 2007 See paragraphs [0018]-[0048]; claims 1-8; and figures 1-6.	1-21
Y	US 2014-0184623 A1 (QUALCOMM INCORPORATED) 03 July 2014 See paragraphs [0015]-[0067]; claims 11-20; and figures 1-3	1-21
A	US 2014-0035937 A1 (THE MATHWORKS, INC.) 06 February 2014 See paragraphs [0175]-[0182]; claim 28; and figures 14-18.	1-21
A	US 2012-0182318 A1 (PHILIP ANDREW MANSFIELD et al.) 19 July 2012 See paragraphs [0103]-[0112]; and figure 14.	1-21
A	US 2009-0322768 A1 (LALONDE PAUL A. et al.) 31 December 2009 See paragraphs [0018]-[0021]; claim 1; and figures 2-4.	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

11 August 2016 (11.08.2016)

Date of mailing of the international search report

16 August 2016 (16.08.2016)

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/US2016/031679

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007-0052723 A1	08/03/2007	AU 2006-287781 A1 BR PI0615723 A2 CA 2621076 A1 CN 101258478 A CN 101258478 B EP 1922634 A1 EP 1922634 A4 JP 2009-507306 A JP 4827923 B2 KR 10-2008-0052574 A NO 20080928 A NO 20080928 B RU 2008108840 A RU 2431185 C2 US 7609280 B2 WO 2007-030283 A1	15/03/2007 10/04/2012 15/03/2007 03/09/2008 27/06/2012 21/05/2008 26/09/2012 19/02/2009 30/11/2011 11/06/2008 03/04/2008 03/04/2008 20/09/2009 10/10/2011 27/10/2009 15/03/2007
US 2014-0184623 A1	03/07/2014	US 9384522 B2 WO 2014-105268 A1	05/07/2016 03/07/2014
US 2014-0035937 A1	06/02/2014	US 2011-0035737 A1 US 8549500 B2 US 8949807 B2	10/02/2011 01/10/2013 03/02/2015
US 2012-0182318 A1	19/07/2012	US 2012-0182317 A1 US 8963959 B2 US 9111327 B2	19/07/2012 24/02/2015 18/08/2015
US 2009-0322768 A1	31/12/2009	None	