



(51) International Patent Classification:
G06F 9/46 (2006.01)

(21) International Application Number:
PCT/CN2015/098634

(22) International Filing Date:
24 December 2015 (24.12.2015)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventors; and

(73) Applicants (for BZ only): LI, Yuanyuan [CN/CN]; Intel No.880 Zi Xing Road, Shanghai Zizhu Science Park, Shanghai 200241 (CN). JIANG, Yong [CN/CN]; Room 102, Building 201, Lane 1288, Xing Song Road, Shanghai 201612 (CN). KONG, Lingyi [CN/CN]; Zizhu2 2w062, ZIZHU2 IT Flex Lab, Shanghai 200052 (CN).

(74) Agent: NTD PATENT AND TRADEMARK AGENCY LIMITED; 10th Floor, Block A, Investment Plaza, 27 Jinrongdajie, Xicheng District, Beijing 100033 (CN).

(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,

[Continued on next page]

(54) Title: FACILITATING EFFICIENT COMMUNICATION AND DATA PROCESSING ACROSS CLUSTERS OF COMPUTING MACHINES IN HETEROGENEOUS COMPUTING ENVIRONMENT

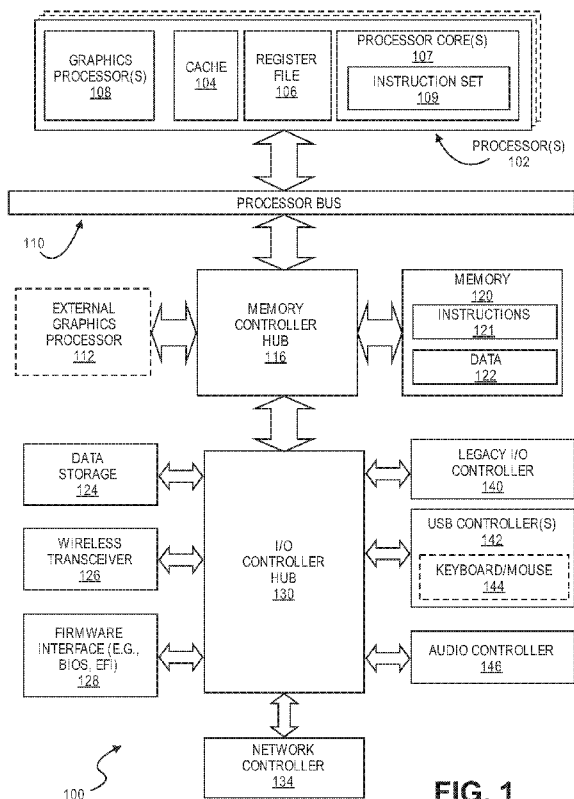


FIG. 1

(57) Abstract: A mechanism is described for facilitating efficient communication and data processing across clusters of computing machines in a heterogeneous computing environment. A method includes detecting a request for processing of data using a programming framework and a programming model; facilitating interfacing between the programming framework and the programming model, wherein interfacing includes merging the programming model into the programming framework, wherein interfacing further includes integrating the programming framework with a distribution framework hosting the programming model; and calling on the distribution framework to schedule processing of a plurality of jobs based on the request.



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

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

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

1
**FACILITATING EFFICIENT COMMUNICATION AND DATA PROCESSING
ACROSS CLUSTERS OF COMPUTING MACHINES IN
HETEROGENEOUS COMPUTING ENVIRONMENT**

FIELD

5 Embodiments described herein generally relate to computers. More particularly, embodiments are described for facilitating efficient communication and data processing across clusters of computing machines in a heterogeneous computing environment.

BACKGROUND

 Although heterogeneous computing environments attempt at working with large data sets
10 across clusters of computers, conventional systems are regarded as inefficient and complex and do not provide for efficient programmability.

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.

15 **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.

20 **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.

25 **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.

5 **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.

10 **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 scalable framework according to one embodiment.

Figure 14A illustrates a scalable framework according to one embodiment.

15 **Figure 14B** illustrates an architectural placement employing a scalable framework of according to one embodiment.

Figure 15A illustrates a transaction sequence for scalable data processing according to one embodiment.

Figure 15B illustrates a two-dimensional placement and a three-dimensional placement of work groups according to one embodiment.

20 **Figure 15C** illustrates a conventional vector kernel and a scalable vector kernel according to one embodiment.

Figure 16A illustrates a method for facilitating scalable runtime according to one embodiment.

25 **Figure 16B** illustrates a method for facilitating scalable data processing according to one embodiment.

DETAILED DESCRIPTION

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

Embodiments provide for a novel technique for facilitating efficient communication and processing of data across clusters of computing machines in a heterogeneous, parallel computing environment. Embodiments provide for integration of multiple programming frameworks and high-level programming models to facilitate enhanced and efficient programmability of graphics processing units (GPUs) (also referenced as “graphics processors”).

Emerging cloud-computing technologies provide for distributed processing of large data sets across clusters of computers; however, complexity and programmability are two of several major obstacles in adopting combinations of platforms, frameworks, and programming models, etc., for more facilitating high level GPU programming, such as adopting a programming framework, such as Open Computing Language (OpenCLTM), with a distribution framework, such as ApacheTM HadoopTM distribution framework, providing high-level programming models, like MapReduceTM, SparkTM, etc. For example, when using OpenCLTM with MapReduceTM, software programmers or developers not only need to have proper knowledge of OpenCLTM, but also they are required to be thoroughly familiar with HadoopTM and MapReduceTM. Further, although using a high-level programming model, such as MapReduceTM, can help with enhancing GPU programming, the developers would still have to be concerned with managing data transfers between host device and client device and *vice versa*, execution of kernels, optimization of the kernel code, and/or the like.

Embodiments provide for a scalable framework for facilitating integration of programming frameworks, distribution frameworks, high-level programming models, etc., such

that programming framework kernels, such as OpenCL™, can run across a cluster of computing machines in a heterogeneous computing environment without having or needing any awareness of the backend cloud computing infrastructure (e.g., distribution frameworks, such as Hadoop™, or high-level programming models, such as MapReduce™, and/or the like).

5 It is contemplated that embodiments are not merely limited to any of OpenCL™ (also referenced as “programming framework”), Hadoop™ (also referred to as “distribution framework”), and MapReduce™ (also referenced as “high-level programming model” or simply “programming model”), and/or the like, and that any number and type of programming frameworks, distribution frameworks, high-level programming models, and/or the like,
10 respectively, may be employed and used with embodiments offering a novel scalable framework.

It is contemplated that terms like “request”, “query”, “job”, “work”, “work item”, and “workload” may be referenced interchangeably throughout this document. Similarly, an “application” or “agent” may refer to or include a computer program, a software application, a game, a workstation application, etc., offered thorough an API, such as a free rendering API,
15 such as Open Graphics Library (OpenGL®), DirectX® 11, DirectX® 12, etc., where “dispatch” may be interchangeably referred to as “work unit” or “draw” and similarly, “application” may be interchangeably referred to as “workflow” or simply “agent”. For example, a workload, such as that of a three-dimensional (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
20 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, each item may be referenced by a single term (e.g., “dispatch”, “agent”, etc.) throughout this document.

In some embodiments, terms like “display screen” and “display surface” may be used
25 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

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 system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

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

In some embodiments, the one or more processors 102 each include one or more processor cores 107 to process instructions which, when executed, perform operations for system

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

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

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 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 cache, where the highest level of cache before external memory is classified as the LLC. In some embodiments, cache coherency logic maintains coherency between the various cache units 206 and 204A-204N.

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

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

In some embodiments, processor 200 additionally includes graphics processor 208 to execute graphics processing operations. In some embodiments, the graphics processor 208 couples with the set of shared cache units 206, and the system agent core 210, including the one or more integrated memory controllers 214. In some embodiments, a display controller 211 is

coupled with the graphics processor 208 to drive graphics processor output to one or more coupled displays. In some embodiments, display controller 211 may be a separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 208 or system agent core 210.

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

10 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.

15 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
20 cores 202A-202N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. Additionally, processor 200 can be implemented on one or more chips or as an SoC integrated circuit having the illustrated components, in addition to other components.

25 **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
5 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
10 306 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

In some embodiments, graphics processor 300 includes a block image transfer (BLIT) engine 304 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in one embodiment, 2D graphics operations are performed using one or more components of graphics processing engine (GPE) 310. In some embodiments, graphics processing engine 310 is a compute engine for performing graphics operations,
15 including three-dimensional (3D) graphics operations and media operations.

In some embodiments, GPE 310 includes a 3D pipeline 312 for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). The 3D pipeline 312 includes programmable and fixed function elements that perform various tasks within the element and/or
25 spawn execution threads to a 3D/Media sub-system 315. While 3D pipeline 312 can be used to perform media operations, an embodiment of GPE 310 also includes a media pipeline 316 that is

specifically used to perform media operations, such as video post-processing and image enhancement.

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

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

3D/Media Processing

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

In some embodiments, GPE 410 couples with a command streamer 403, which provides a command stream to the GPE 3D and media pipelines 412, 416. In some embodiments,

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

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

In some embodiments, the specialized media sampling logic in sampling engine 430 includes a de-noise/de-interlace module 432, a motion estimation module 434, and an image scaling and filtering module 436. In some embodiments, de-noise/de-interlace module 432 includes logic to perform one or more of a de-noise or a de-interlace algorithm on decoded video data. The de-interlace logic combines alternating fields of interlaced video content into a single frame of video. The de-noise logic reduces or removes data noise from video and image data. In some embodiments, the de-noise logic and de-interlace logic are motion adaptive and use spatial or temporal filtering based on the amount of motion detected in the video data. In some embodiments, the de-noise/de-interlace module 432 includes dedicated motion detection logic (e.g., within the motion estimation engine 434).

In some embodiments, motion estimation engine 434 provides hardware acceleration for video operations by performing video acceleration functions such as motion vector estimation and prediction on video data. The motion estimation engine determines motion vectors that describe the transformation of image data between successive video frames. In some

5 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

10 motion within video data.

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

15 In some embodiments, the GPE 410 includes a data port 444, which provides an additional mechanism for graphics subsystems to access memory. In some embodiments, data port 444 facilitates memory access for operations including render target writes, constant buffer reads, scratch memory space reads/writes, and media surface accesses. In some embodiments, data port 444 includes cache memory space to cache accesses to memory. The cache memory

20 can be a single data cache or separated into multiple caches for the multiple subsystems that access memory via the data port (e.g., a render buffer cache, a constant buffer cache, etc.). In some embodiments, threads executing on an execution unit in execution unit array 414 communicate with the data port by exchanging messages via a data distribution interconnect that couples each of the sub-systems of GPE 410.

Execution Units

Figure 5 is a block diagram of another embodiment of a graphics processor 500.

Elements of **Fig. 5** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

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

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

In some embodiments, graphics processor 500 includes scalable thread execution resources featuring modular cores 580A-580N (sometimes referred to as core slices), each having multiple sub-cores 550A-550N, 560A-560N (sometimes referred to as core sub-slices).

In some embodiments, graphics processor 500 can have any number of graphics cores 580A through 580N. In some embodiments, graphics processor 500 includes a graphics core 580A having at least a first sub-core 550A and a second core sub-core 560A. In other embodiments, the graphics processor is a low power processor with a single sub-core (e.g., 550A). In some
5 embodiments, graphics processor 500 includes multiple graphics cores 580A-580N, each including a set of first sub-cores 550A-550N and a set of second sub-cores 560A-560N. Each sub-core in the set of first sub-cores 550A-550N includes at least a first set of execution units 552A-552N and media/texture samplers 554A-554N. Each sub-core in the set of second sub-cores 560A-560N includes at least a second set of execution units 562A-562N and samplers
10 564A-564N. In some embodiments, each sub-core 550A-550N, 560A-560N shares a set of shared resources 570A-570N. In some embodiments, the shared resources include shared cache memory and pixel operation logic. Other shared resources may also be included in the various embodiments of the graphics processor.

Figure 6 illustrates thread execution logic 600 including an array of processing elements
15 employed in some embodiments of a GPE. Elements of **Fig. 6** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

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

each thread. In some embodiments, execution unit array 608A-608N includes any number individual execution units.

In some embodiments, execution unit array 608A-608N is primarily used to execute “shader” programs. In some embodiments, the execution units in array 608A-608N execute an instruction set that includes native support for many standard 3D graphics shader instructions, such that shader programs from graphics libraries (e.g., Direct 3D and OpenGL) are executed with a minimal translation. The execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, vertex shaders), pixel processing (e.g., pixel shaders, fragment shaders) and general-purpose processing (e.g., compute and media shaders).

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

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

However, different vector widths and register sizes are possible.

One or more internal instruction caches (e.g., 606) are included in the thread execution logic 600 to cache thread instructions for the execution units. In some embodiments, one or 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.

5 **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
10 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
15 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
20 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
25 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
5 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
10 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
15 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

20 **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
25 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 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 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 is 912 is required immediately before a pipeline switch via the pipeline select command 913.

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

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

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

The commands for the 3D pipeline state 930 include 3D state setting commands for vertex buffer state, vertex element state, constant color state, depth buffer state, and other state variables that are to be configured before 3D primitive commands are processed. The values of these commands are determined at least in part based the particular 3D API in use. In some embodiments, 3D pipeline state 930 commands are also able to selectively disable or bypass 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
5 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
10 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
15 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
20 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
25 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. The operating system 1020 can support a graphics API 1022 such as the Direct3D API or the OpenGL API. When the Direct3D API is in use, the operating system 1020 uses a front-end shader compiler 1024 to compile any shader instructions 1012 in HLSL into a lower-level shader language. The compilation may be a just-in-time (JIT) compilation or the application can perform shader pre-compilation. In some embodiments, high-level shaders are compiled into low-level shaders during the compilation of the 3D graphics application 1010.

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

IP Core Implementations

One or more aspects of at least one embodiment may be implemented by representative code stored on a machine-readable medium which represents and/or defines logic within an integrated circuit such as a processor. For example, the machine-readable medium may include instructions which represent various logic within the processor. When read by a machine, the instructions may cause the machine to fabricate the logic to perform the techniques described herein. Such representations, known as “IP cores,” are reusable units of logic for an integrated circuit that may be stored on a tangible, machine-readable medium as a hardware model that

describes the structure of the integrated circuit. The hardware model may be supplied to various customers or manufacturing facilities, which load the hardware model on fabrication machines that manufacture the integrated circuit. The integrated circuit may be fabricated such that the circuit performs operations described in association with any of the embodiments described
5 herein.

Figure 11 is a block diagram illustrating an IP core development system 1100 that may be used to manufacture an integrated circuit to perform operations according to an embodiment. The IP core development system 1100 may be used to generate modular, re-usable designs that can be incorporated into a larger design or used to construct an entire integrated circuit (e.g., an
10 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 using a simulation model 1112. The simulation model 1112 may include functional, behavioral, and/or timing simulations. A register transfer level (RTL) design can then be created or synthesized from the simulation
15 model 1112. The RTL design 1115 is an abstraction of the behavior of the integrated circuit that models the flow of digital signals between hardware registers, including the associated logic performed using the modeled digital signals. In addition to an RTL design 1115, lower-level designs at the logic level or transistor level may also be created, designed, or synthesized. Thus, the particular details of the initial design and simulation may vary.

20 The RTL design 1115 or equivalent may be further synthesized by the design facility into a hardware model 1120, which may be in a hardware description language (HDL), or some other representation of physical design data. The HDL may be further simulated or tested to verify the IP core design. The IP core design can be stored for delivery to a 3rd party fabrication facility 1165 using non-volatile memory 1140 (e.g., hard disk, flash memory, or any non-volatile storage
25 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 scalable framework 1310 according to one embodiment. Computing device 1300 (e.g., mobile computer, laptop computer, desktop computer, server 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. As

illustrated, in one embodiment, computing device 1300 is shown as hosting scalable framework 1310.

In the illustrated embodiment, scalable framework 1310 is shown as being hosted by operating system 1316; however, it is contemplated that embodiments are not limited as such.

5 For example, in another embodiment, one or more components 1401-1405, 1411-1417 of scalable framework 1310 may be part of firmware of GPU 1314. In yet another embodiment, one or more components 1401-1405, 1411-1417 of scalable framework 1310 may be one or more hardware components hosted by GPU 1314. In yet another embodiment, scalable framework 1310 may be partially and simultaneously hosted by multiple components of
10 computing device 1300, such as one or more of graphics driver 1316, GPU 1314, GPU firmware, operating system 1316, and/or the like.

Throughout the document, the term “user” may be interchangeably referred to as “viewer”, “observer”, “person”, “individual”, “end-user”, and/or the like; further, “programmer”, “developer”, and “user” may be interchangeably referenced throughout this document. It is to be
15 noted that throughout this document, terms like “graphics domain” may be referenced interchangeably with “graphics processor”, “graphics processing unit” or simply “GPU” and similarly, “CPU domain” or “host domain” may be referenced interchangeably with “application processor”, “computer processing unit” or simply “CPU”.

Computing device 1300 may include any number and type of communication devices,
20 such as large computing systems, such as server computers, desktop computers, etc., and may further include set-top boxes (e.g., Internet-based cable television set-top boxes, etc.), global positioning system (GPS)-based devices, etc. Computing device 1300 may include mobile computing devices serving as communication devices, such as cellular phones including smartphones, personal digital assistants (PDAs), tablet computers, laptop computers, e-readers,
25 smart televisions, television platforms, wearable devices (e.g., glasses, watches, bracelets, smartcards, jewelry, clothing items, etc.), media players, etc. For example, in one embodiment,

computing device 1300 may include a mobile computing device employing an integrated circuit (“IC”), such as system on a chip (“SoC” or “SOC”), integrating various hardware and/or software components of computing device 1300 on a single chip.

As illustrated, in one embodiment, computing device 1300 may include any number and type of hardware and/or software components, such as (without limitation) graphics processing unit 1314, graphics driver (also referred to as “GPU driver”, “graphics driver logic”, “driver logic”, user-mode driver (UMD), UMD, user-mode driver framework (UMDF), UMDF, or simply “driver”) 1316, central processing unit 1312, memory 1308, network devices, drivers, or the like, as well as input/output (I/O) sources 1304, such as touchscreens, touch panels, touch pads, virtual or regular keyboards, virtual or regular mice, ports, connectors, etc. Computing device 1300 may include operating system (OS) 1306 serving as an interface between hardware and/or physical resources of the computer device 1300 and a user. It is contemplated that CPU 1312 may include one or processors, such as processor(s) 102 of **Figure 1**, while GPU 1314 may include one or more graphics processors, such as graphics processor(s) 108 of **Figure 1**.

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 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 any number and type of instructions, such as instructions 121 of **Figure 1**, to perform the various novel functions of scalable framework 1310 as disclosed throughout this document.

5 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 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
10 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 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),
15 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); 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.

20 Computing device 1300 may further include input/output (I/O) control hub (ICH) 130 as referenced in **Figure 1**, one or more I/O sources 1304, etc.

CPU 1312 may include one or more processors to execute instructions in order to perform whatever software routines the computing system implements. The instructions frequently involve some sort of operation performed upon data. Both data and instructions may
25 be stored in system memory 1308 and any associated cache. Cache is typically designed to have shorter latency times than system memory 1308; for example, cache might be integrated onto the

same silicon chip(s) as the processor(s) and/or constructed with faster static RAM (SRAM) cells whilst 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), 4th Generation (4G), 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 scalable framework 1310 according to one embodiment. For brevity, many of the details already discussed with reference to **Figures 1-13** are not repeated or discussed hereafter. In one embodiment, scalable framework 1310 may include any number and type of components, such as (without limitation): detection/reception logic 1401; integration and processing engine 1403; and communication/compatibility logic 1405. As illustrated, in one embodiment, integration and processing engine 1403 may include one or more components, such as (without limitation): interfacing logic 1411, partition/vector logic 1413, processing logic 1415, and execution/calling logic 1417.

Computing device 1300 is further shown to be in communication with one or more repositories, datasets, and/or databases, such as database(s) 1430 (e.g., cloud storage, non-cloud storage, etc.), where the communication may be direct and/or over communication medium 1425,

such as one or more networks (e.g., a cloud network, a proximity network, a mobile network, an intranet, the Internet, etc.).

Similarly, in one embodiment, computing device 1300 (e.g., server computer) may be communicatively part of a network including one or more computing devices, such as computing device 1440 (e.g., server computer) and computing devices 1460A, 1460B, 1460C (e.g., client computers). For example, as illustrated, computing device 1440 may serve to host a distribution framework 1441, such as HadoopTM, etc., that offers or provides access to one or more high-level programming models (“programming models” or simply “models”) 1443, such as MapReduceTM, etc. Further, in one embodiment, computing device 1440 is further shown to host communication logic 1451 along with distribution-side processing engine (“distribution engine”) 1445 including one or more components, such as (without limitation) dispatch logic 1447 and scheduling/mapping logic 1449.

Similarly, for example, each of computing devices 1460A-N may serve client computers hosting a programming framework, such as OpenCLTM, etc., along with programming-side processing engine (“programming engine”) 1463A, 1463B, 1463N having one or more components, such as (without limitation): partition/assignment logic 1465A, 1465B, 1465N and data processing logic 1467A, 1467B, 1467N. Further, each of computing devices 1460A-N may further provide user interface 1469A, 1469B, 1469N while hosting communication logic 1471A, 1471B, 1471N and one or more I/O components 1473A, 1473B, 1473B (e.g., sensors, detectors, receptors, cameras, microphones, speakers, display screens/devices, etc.).

As an initial matter, embodiments are not limited to merely any particular numbers and types of programming frameworks, distribution frameworks, high-level programming models, etc.; however, for the sake of brevity, clarity, and ease of understanding, OpenCLTM, HadoopTM, and MapReduceTM, respectively, are referenced throughout this document.

For example, OpenCLTM refers to a programming framework for writing software programs that are capable of being executed across heterogeneous platforms including GPUs,

central processing units (CPUs), field-programmable gate array (FPGAs), digital signal processors (DSPs), and other processors, etc. Similarly, ApacheTM HadoopTM refers to a distribution framework including a Java®-based open-source software framework for distributed storage and processing of rather large data sets on and across clusters of computing machines.

5 Further, HadoopTM MapReduceTM refers to a high-level programming model/framework for generating and processing data sets using parallel, distributed algorithm, in a cluster of computing machines, such as computing devices 1460A-N, where a MapReduceTM program typically includes: 1) a Map() procedure or method for performing filtering and sorting of data (e.g., sorting individuals in alphabetical order, age, geographic locations, etc.); and 2) a Reduce()
10 procedure or method for performing a summary operation (e.g., counting the number of individuals, etc.). Similarly, for example, ApacheTM SparkTM refers to an open source computing/software framework that provides multi-stage in-memory primitives and allows developers to load data into a cluster's memory and query it repeatedly.

In one embodiment, scalable framework 1310 offers a mechanism to run programming
15 framework kernels (e.g., OpenCLTM kernels) across a cluster of computing machines, such as computing devices 1460A-N, without needing for computing devices 1460A-N and/or developers having access to computing devices 1460A-N to be aware of any backend computing infrastructure (e.g., backend cloud computing infrastructure), such as computing device 1440, etc. For example, scalable framework 1310 facilitates integration or merging of programming model
20 1443 (e.g., MapReduceTM <Key-Value> model) offered at computing device 1440 with programming models 1461A-N (e.g., OpenCLTM) offered at computing devices 1460A-N to obtain and work as a singular and unified programming infrastructure representing a two-level parallelism (e.g., node-level and execution unit (EU)-level parallelism) of distribution and programming frameworks 1441, 1461A-N (e.g., HadoopTM+OpenCLTM) leading to programming
25 model 1443 and programming framework 1461A-N (e.g., MapReduceTM+OpenCLTM), etc.

This novel integration of programming model 1443 with programming framework 1461A-N reduces conventional complexities and improves programmability since it does not require developers to have knowledge of both programming model 1443 and programming framework 1461A-N. This integration further provides for dynamic scalability, such as

5 continuous processing and compatibility, even with an increased number of nodes, data sets, etc., such as by having distribution framework 1441 to spill data and schedule jobs using distribution-side processing engine 1445.

In one embodiment, detection/reception logic 1401 of scalable framework 1310 is used to receive or detect any request to use programming framework 1461A-N with programming model

10 1443, such as receive a request from computing device 1461A, such as where a developer having access to computing device 1461A is attempting to use programming framework 1461A in combination with programming model 1443. Upon detection of the user request, integration and processing engine (“engine”) 1403 may be triggered for further processing of this user request.

In one embodiment, interfacing logic 1411 of engine 1403 may be used to provide and

15 maintain an interface between computing devices 1440, 1460A-N and their relevant components, such as programming model 1443, distribution framework 1441 at computing device 1440 and programming framework 1461A-N at computing devices 1460A-N. For example, in case of the user request placed by the developer at computing device 1461A, this novel interface, as facilitated by interfacing logic 1411, allows the developer to perform various programming tasks

20 using programming framework 1461A at client computing device 1460A while using and reaping benefits of programming model 1443 at server computing device 1440 without needing to have any knowledge or knowhow of programming model 1443, distribution framework 1441, computing device 1440, etc.

As will be further described with reference to **Figure 15A**, continuing with the example

25 of receiving a user request from the developer having access to computing device 1460A, partition/vector logic 1413 of engine 1403 may triggered to generate pairs for data partitions,

such as generating <Key, Value> pairs for data partition of big data relating to programming and other tasks being performed at various computing devices in a cluster, such as computing devices 1460A-N. Once the pairs are generated for data partition, processing logic 1415 may then be triggered to process the code relating to each of the job identifications (IDs) in kernel, where a
5 job ID may be assigned as a vector-addition kernel as will be further described and illustrated with regard to **Figure 15C**. Further, upon processing of the code, processing logic 1415 may then be used to generate jobs, such as Map and Reduce jobs based on Map() and Reduce() methods of MapReduceTM of HadoopTM, for programming model 1443 of distribution framework 1441. Once the jobs have been generated, execution/calling logic 1417 may then be
10 used to call on distribution framework 1441 (e.g., HadoopTM) to execute the jobs, such as Map/Reduce jobs.

In one embodiment, scalable framework 1310 works with distribution engine 1445 at computing device 1440, to perform one or more tasks relating to distribution framework 1441 and programming model 1443. For example, using dispatch logic 1447 of distribution engine
15 1445 may be used to dispatch the data partitions on a node pool associated with distribution framework 1441, wherein the data partitions created by partition/vector logic 1413. Upon dispatching of the data partitions, scheduling/mapping logic 1449 may then be triggered to schedule the jobs, such as Map/Reduce jobs created by execution/calling logic 1417, on the associated node pool. It is contemplated that distribution framework 1441 may be associated
20 with any number and size of node pools for data processing and job executions.

Similarly, in one embodiment, scalable framework 1310 is further in communication with programming engine 1463A-N to perform one or more tasks to facilitate processing of any data received from distribution engine 1443 as local data at computing devices 1463A-N and their corresponding execution unit (EU) pools. For example, each of partition/assignment logic
25 1465A-N may be used to receive its corresponding data partitions from distribution engine 1445, where this data partitions are then assigned to an EU pool as local data and to be processed as

such. In one embodiment, data processing logic 1467A-N is then triggered to call on programming framework 1461A-B (e.g., OpenCL™ runtime) to process this local data and subsequently, data processing logic 1467A-N facilitates programming framework 1461A-N to split the data into micro blocks and have any work items running on the EU pool.

5 As aforementioned, in one embodiment, scalable framework 1310 facilitates seamless interfacing between distribution and programming engines 1445, 1463A-N to provide for a unified merging and integrating of programming framework 1461A-N with programming model 1443 and its corresponding distribution framework 1441. Further, in one embodiment, scalable framework 1310 provides for a two-level parallelism in distribution and programming
10 frameworks 1441, 1461A-N, where the two levels include the node-level of node pools and the EU-level of EU pools. For example, the node-level part of the parallelism occurs at distribution framework 1441, as facilitate by distribution engine 1445, which splits big data into a bunch of partitions and sends them with their corresponding job identifiers, such as <Key, Value>, to nodes in the cluster. Subsequently, for example, the tasks or jobs, such as Map and Reduce jobs,
15 are schedule on each node to process its assigned partitions independently, such as with each Map and Reduce job, programming framework 1461A-N may spawn a large number of kernels to be running, in parallel, on corresponding EU nodes.

As illustrated, for example, computing device 1300 may be in communication with one or more repositories, data sources, and databases, such as database(s) 1430. In some
20 embodiments, database(s) 1430 may be used to store and maintain any amount and type of data, metadata, etc., relating to computing devices 1300, 1440, 1460, users having access to computing devices 1300, 1440, 1460, and data being processed on and communication between computing devices 1300, 1440, 1460, and/or the like.

Communication/compatibility logic 1405 may be used to facilitate dynamic
25 communication and compatibility between 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.

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. In one example, "logic" may refer to or include a software component that is capable of working with one or more of an operating system (e.g., operating system 1306), a graphics driver (e.g., graphics driver 1316), etc., of a computing device, such as computing device 1300. In another example, "logic" may refer to or include a hardware component that is capable of being physically installed along with or as part of one or more system hardware elements, such as an application processor (e.g., CPU 1312), a graphics processor (e.g., GPU 1314), etc., of a computing device, such as computing device 1300. In yet another embodiment, "logic" may refer to or include a firmware component that is capable of being part of system firmware, such

as firmware of an application processor (e.g., CPU 1312) or a graphics processor (e.g., GPU 1314), etc., of a computing device, such as computing device 1300.

Further, any use of a particular brand, word, term, phrase, name, and/or acronym, such as “GPU”, “GPU domain”, “GPGPU”, “CPU”, “CPU domain”, “graphics driver”, “workload”, “application”, “frame”, “work unit”, “work group”, “draw”, “dispatch”, “API”, “hardware”, “software”, “agent”, “graphics driver”, “kernel mode graphics driver”, “user-mode driver”, “UMD”, “user-mode driver framework”, “UMDF”, “3D graphics API”, “GPU programmability”, “node pool”, “EU pool”, “scalable framework”, “programming framework”, “distribution framework”, “programming model”, “OpenCLTM”, “HadoopTM”, “MapReduceTM”, “job ID”, “vector-addition”, 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 scalable framework 1310 to facilitate various embodiments including adding, removing, and/or enhancing certain features. For brevity, clarity, and ease of understanding of scalable framework 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 adapt to and adopt any future changes.

Figure 14B illustrates an architectural placement 1480 employing a scalable framework 1310 of according to one embodiment. As an initial matter, for brevity, many of the details relating to various processes, components, etc., discussed with reference to any of the preceding such as **Figures 1-14A**, may not be discussed or repeated hereafter. Moreover, it is contemplated and to be noted that embodiments are not limited to any particular setup, placement, structure, etc., such as architectural placement 1480.

As described with reference to **Figure 14A**, architectural placement 1480 is shown to include computing device 1300 serving as a server computer hosting scalable framework 1310

and communicatively coupled with computing device 1440 serving as a distribution server computer hosting distribution framework 1441 (e.g., “HadoopTM”, etc.) and high-level programming model 1443 (e.g., “MapReduceTM”, etc.). In one embodiment, computing devices 1300, 1440 are further in communication with one or more computing devices 1460A-N serving as client computers hosting programming framework 1461A-N (e.g., “OpenCLTM”, etc.) and one or more other components and/or devices 1481A-N, such as CPU 1312, GPU 1314, I/O sources 1304, memory 1308, operating system 1306, graphics driver 1316, and/or the like.

As illustrated with reference to **Figure 14A**, computing devices 1300, 1440, 1460A-N may be in communication with each other over communication medium 1425, such as one or more networks including a cloud network, the Internet, etc. Further, in one embodiment, scalable framework 1310 provides for integration and merging of programming frameworks 1461A-N (e.g., “OpenCLTM”, etc.) at programming client computers 1460A-N with distribution framework 1441 (e.g., “HadoopTM”, etc.) and/or high-level programming model 1443 (e.g., “MapReduceTM”, etc.) at distribution computer server 1440. Further, in one embodiment, having scalable framework 1310 on top of distribution framework 1441 helps generate programming model tasks, such as MapReduceTM jobs, in accordance with the developer’s input.

Figure 15A illustrates a transaction sequence 1500 for scalable data processing according to one embodiment. Transaction 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, as facilitated by scalable framework 1310 and/or distribution engine 1445 and/or programming engine 1463A-N of **Figure 13**. The processes of transaction 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 1-14B** may not be discussed or repeated hereafter.

Transaction sequence 1500 begins with data, such as big data 1501, being received at scalable framework 1310. In one embodiment, as described with reference to **Figure 15A**, scalable framework 1310 may be used to generate pairs, such as <Key, Value> pairs, for performing data partition of big data 1501 into smaller data partitions 1505, such as smaller partitions or pieces. Further, for example, scalable framework 1310 is used to process the code relating to job IDs in kernel and generate jobs (e.g., Map/Reduce jobs) for further processing at distribution framework 1441. For example, each job having an associated job ID may relate to processing of a corresponding data partition of data partitions 1505. Subsequently, scalable framework 1310 calls on distribution framework 1441 to execute the jobs as identified by their job IDs and corresponding to data partitions 1505.

Transaction sequence 1500 continues with distribution engine 1445 at a distribution server computer, such as computing device 1440 of **Figure 14A**. In one embodiment, distribution engine 1445 works with distribution framework 1441 hosting high-level programming model 1443 to dispatch data partitions 1505 on node pool 1503 as well as schedule their corresponding jobs, such as Map/Reduce jobs, in node pool 1503. For example, any singular data partition, such as data partition 1507, of data partitions 1505 may be identified using a job ID of its corresponding job and communicated on to a corresponding programming client computer, such as computing device 1460A, to be processed as local data 1511.

In one embodiment, this data partition 1507 is assigned and received as local data 1511 at programming engine 1463A which then calls on programming framework 1461A to process local data 1511 and facilitates programming framework 1461A to split this local data 1511 into any number and size of micro-blocks corresponding to work items 1515 running on EU pool 1513.

As illustrated, scalable framework 1310 provides for integration of programming framework 1461A with distribution framework 1441 and high-level programming model 1443 along with offering a multi-level parallel environment that that the node-level parallelism occurs

at the level of distribution framework 1441, while the EU-level parallelism occurs at the local level of programming framework 1441.

Figure 15B illustrates a two-dimensional (2D) placement 1520 and a three-dimensional (3D) placement 1530 of work groups according to one embodiment. As an initial matter, it is contemplated and to be noted that embodiments are not limited to any particular setup, placement, structure, etc., such as placements 1520, 1530.

In one embodiment, as previously described with reference to **Figure 15A**, this integration of distribution framework 1441 and programming framework 1461A-N provides for merging of programming model 1443 into programming framework 1461A-N introducing one or more dimensions of work group as to indicate node-level parallelism being performed by programming model 1443, such as MapReduceTM. For example, as illustrated, in conventional techniques, work groups may only be presented and processed in 2D placement 1520, but scalable framework 1310 of **Figure 13** provides for additional, third dimensions, to present and process work groups as shown in 3D placement 1530.

For example, as illustrated with reference to **Figure 15A**, a programming framework, such as OpenCLTM, may spawn work groups on a computing device, where each work group may contain a set of work items. For example, a work group's size and dimension may depend the workload size and the size of a micro-block that each work item processes. Conventionally, for example, a work item may use its ID (e.g., work group ID, local ID, global ID, etc.) to calculate the location of the assigned data partition, such as data partition 1507 being assigned as local data 1511 in **Figure 15A**. However, in one embodiment, with the help of $N^{th}+1$ dimension of 3D placement 1530, a work item may locate its local data corresponding to the data partition in the whole data space as facilitated by scalable framework 1310 of **Figure 13**. Further, for example, the $N^{th}+1$ dimension work group is shown when $N = 2$, where one kernel extension `get_job_id()` is introduced to get the job ID belonging to the work item.

Figure 15C illustrates a conventional vector kernel 1530 and a scalable vector kernel

1540 according to one embodiment. As an initial matter, it is contemplated and to be noted that embodiments are not limited to any particular setup, placement, structure, code, etc., such as conventional vector kernel 1530, scalable vector kernel 1540, etc.

As illustrated, a comparison of conventional vector kernel 1530, such as associated with
5 OpenCL™, with scalable vector kernel 1540 as facilitated by scalable framework 1310 of
Figure 13 indicates that each work item, such as an OpenCL™ work item, calculates nearly 128
integers and uses global ID to locate the relevant micro-block. Now, for example, two vectors
whose size is 100GB may run on a system of 400-node cluster where each job handles about
256MB of data. In one embodiment, scalable vector kernel 1530 is efficient and distinct because,
10 for example, it includes an additional kernel 1541, such as kernel *VectorAddition_SCL()*
representing *unsigned int jobid = get_job_id(0)*, that uses job ID and location ID to local the
relevant data, such as local data from a data partition. For example, the interface exposed by
scalable framework of **Figure 13** may be used as a subset of OpenCL™ interfaces, allowing
developers to control the size and dimensions of work groups along with performing resource
15 management and data partition, etc.

Figure 16A illustrates a method 1600 for facilitating scalable runtime 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, as facilitated by scalable framework 1310 and/or
20 distribution engine 1445 and/or programming engine 1463A-N of **Figure 13**. 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 1-15C may not be discussed or repeated hereafter.

25 Method 1600 begins at block 1601 with construction of pairs, such as <Key, Value> pairs,
for data partition of big data by scalable framework 1310 of **Figure 13**, where these pairs may be

used as in input for a distribution framework, such as HadoopTM, etc. Further, in one embodiment, the scalable runtime as facilitated by scalable framework 1310 of **Figure 13** splits data into smaller blocks or partitions in accordance with the settings as provided by the relevant software application and then assigns job IDs to each of the data partitions. At block 1603, as
5 facilitated by scalable framework 1310 of **Figure 13**, the extension *get_job_id* is stripped from the kernel and replaced by using a zero because, for example, an OpenCLTM kernel running on each computing device processes data partitions associated with jobs as local data.

At block 1605, continuing with the example of MapReduceTM being the high-level programming model, a code (e.g., *Map()* and *Reduce()* code) is generated in accordance with the
10 programming model (e.g., MapReduceTM) to run the programming framework (e.g., OpenCLTM) (such as OpenCLTM Java® binding may be used) on each node to perform any necessary computing and processing. At block 1607, various corresponding jobs (e.g., *Map()* and *Reduce()* jobs) are scheduled on a cluster of computing machines (such as computing devices 1460A-N of **Figure 14A**) through the distribution framework, such as HadoopTM framework.

15 **Figure 16B** illustrates a method 1650 for facilitating scalable data processing 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, as facilitated by scalable framework 1310 and/or distribution engine 1445 and/or programming engine 1463A-N of
20 **Figure 13**. 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 begins at block 1651 with generation of pairs, such as <Key, Value> pairs,
25 for data partition of big data according to the specifications provided by the relevant software application. At block 1653, a code (e.g., *Map()* and *Reduce()* code) is generated and processed

relating to job IDs in kernel in accordance with the high-level programming model (e.g., MapReduceTM) to run a programming framework (e.g., OpenCLTM). At block 1655, jobs (e.g., *Map()* and *Reduce()* jobs) are generated for and to be distributed to a distribution framework (e.g., HadoopTM), where the distribution framework is called to execute the jobs at block 1657.

5 In one embodiment, processes 1651, 1653, 1655, 1657 may be performed and facilitated by scalable framework 1310 of **Figure 13**.

At block 1659, the various data partitions received from scalable framework 1310 of **Figure 13** are then dispatched to node pool and subsequently, at block 1661, the jobs (e.g., *Map()* and *Reduce()* jobs) are scheduled for a node pool. In one embodiment, processes 1659 and 1661
10 may be performed and facilitated by distribution engine 1445 of **Figure 14A** as facilitated by scalable framework 1310 of **Figure 13** and/or distribution framework 1441 and/or high-level programming model 1443 of **Figure 14A**. For example, execution/calling logic 1417 of scalable framework 1310 may call on and facilitate distribution engine 1445 and/or distribution
15 framework 1441 and/or programming model 1443 of **Figure 14A** to perform one or more of processes 1659 and 1661.

At block 1663, a data partition of the data partitions is received at a computing device, where the data partition is assigned to a node as local data. At block 1665, a relevant programming framework (e.g., OpenCLTM runtime) is called to process the local data. At block 1667, the programming framework splits data into several micro-blocks to run and process them
20 as work items on an EU pool. In one embodiment, processes 1663, 1665, and 1667 may be performed and facilitated by programming engine 1463A-N of **Figure 14A** as facilitated by scalable framework 1310 of **Figure 13** and/or programming framework 1461A-N of **Figure 14A**. For example, execution/calling logic 1417 of scalable framework 1310 may call on and facilitate programming engine 1463A-N and/or programming framework 1461A-N of **Figure 14A** to
25 perform one or more of processes 1663, 1665, and 1667.

References to “one embodiment”, “an embodiment”, “example embodiment”, “various

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

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

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

As used in the claims, unless otherwise specified the use of the ordinal adjectives “first”, “second”, “third”, etc., to describe a common element, merely indicate that different instances of like elements are being referred to, and are not intended to imply that the elements so described must be in a given sequence, either temporally, spatially, in ranking, or in any other manner.

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

20

25

Some embodiments pertain to Example 1 that includes an apparatus to facilitate efficient communication and data processing across clusters of computing machines in a heterogeneous computing environment, comprising: detection/reception logic to detect a request for processing of data using a programming framework and a programming model; interfacing logic to facilitate
5 interfacing between the programming framework and the programming model, wherein interfacing includes merging the programming model into the programming framework, wherein interfacing further includes integrating the programming framework with a distribution framework hosting the programming model; and execution/calling logic to call on the distribution framework to schedule processing of a plurality of jobs based on the request.

10 Example 2 includes the subject matter of Example 1, further comprising partition/vector logic to split the data into a plurality of data partitions, wherein the detection/reception logic is further to receive the data, wherein interfacing is provided over a network including a cloud network.

Example 3 includes the subject matter of Example 1 or 2, further comprising processing
15 logic to process the data to correspond the plurality of data partitions to the plurality of jobs, and assign a plurality of job identifications (IDs) to the plurality of jobs such that each job is identified by a job ID.

Example 4 includes the subject matter of Example 1, wherein the execution/calling logic is further to facilitate the distribution framework to dispatch the plurality of data partitions to a
20 node pool, wherein the execution/calling logic is further to facilitate the distribution framework to schedule the plurality of jobs corresponding to the plurality of data partitions via the node pool.

Example 5 includes the subject matter of Example 1 or 4, wherein the execution/calling logic is further to facilitate the distribution framework to provide each data partition of the plurality of data partitions as local data to the programming framework.

25 Example 6 includes the subject matter of Example 1 or 4, wherein the execution/calling logic is further to facilitate the programming framework to split the local data into a plurality of

micro-blocks representing a plurality of work items to be processed via an execution unit (EU) pool.

Example 7 includes the subject matter of Example 1, wherein the distribution framework and the programming model reside at a first computing device serving as a distribution server computer, wherein the distribution framework includes HadoopTM framework, and wherein the programming model includes a MapReduceTM model, wherein a job of the plurality of jobs includes a Map job and a Reduce job.

Example 8 includes the subject matter of Example 1, wherein the programming framework resides a second computing device serving as a programming client computer, wherein the programming framework includes OpenCLTM framework.

Some embodiments pertain to Example 9 that includes a method for facilitating efficient communication and data processing across clusters of computing machines in a heterogeneous computing environment, comprising: detecting a request for processing of data using a programming framework and a programming model; facilitating interfacing between the programming framework and the programming model, wherein interfacing includes merging the programming model into the programming framework, wherein interfacing further includes integrating the programming framework with a distribution framework hosting the programming model; and calling on the distribution framework to schedule processing of a plurality of jobs based on the request.

Example 10 includes the subject matter of Example 9, further comprising: splitting the data into a plurality of data partitions; and receiving the data, wherein interfacing is provided over a network including a cloud network.

Example 11 includes the subject matter of Example 9 or 10, further comprising: processing the data to correspond the plurality of data partitions to the plurality of jobs; and assigning a plurality of job identifications (IDs) to the plurality of jobs such that each job is identified by a job ID.

Example 12 includes the subject matter of Example 9, further comprising: facilitating the distribution framework to dispatch the plurality of data partitions to a node pool; and facilitating the distribution framework to schedule the plurality of jobs corresponding to the plurality of data partitions via the node pool.

5 Example 13 includes the subject matter of Example 9 or 12, further comprising facilitating the distribution framework to provide each data partition of the plurality of data partitions as local data to the programming framework.

Example 14 includes the subject matter of Example 9 or 12, further comprising facilitating the programming framework to split the local data into a plurality of micro-blocks
10 representing a plurality of work items to be processed via an execution unit (EU) pool.

Example 15 includes the subject matter of Example 9, wherein the distribution framework and the programming model reside at a first computing device serving as a distribution server computer, wherein the distribution framework includes HadoopTM framework, and wherein the programming model includes a MapReduceTM model, wherein a job of the
15 plurality of jobs includes a Map job and a Reduce job.

Example 16 includes the subject matter of Example 9, wherein the programming framework resides a second computing device serving as a programming client computer, wherein the programming framework includes OpenCLTM framework.

Some embodiments pertain to Example 17 includes a system comprising a storage device
20 having instructions, and a processor to execute the instructions to facilitate a mechanism to perform one or more operations comprising: detecting a request for processing of data using a programming framework and a programming model; facilitating interfacing between the programming framework and the programming model, wherein interfacing includes merging the programming model into the programming framework, wherein interfacing further includes
25 integrating the programming framework with a distribution framework hosting the programming

model; and calling on the distribution framework to schedule processing of a plurality of jobs based on the request.

Example 18 includes the subject matter of Example 17, wherein the one or more operations comprise: splitting the data into a plurality of data partitions; and receiving the data, wherein interfacing is provided over a network including a cloud network.

Example 19 includes the subject matter of Example 17 or 18, wherein the one or more operations comprise: processing the data to correspond the plurality of data partitions to the plurality of jobs; and assigning a plurality of job identifications (IDs) to the plurality of jobs such that each job is identified by a job ID.

Example 20 includes the subject matter of Example 17, wherein the one or more operations comprise facilitating: the distribution framework to dispatch the plurality of data partitions to a node pool; and facilitating the distribution framework to schedule the plurality of jobs corresponding to the plurality of data partitions via the node pool.

Example 21 includes the subject matter of Example 17 or 20, wherein the one or more operations comprise facilitating the distribution framework to provide each data partition of the plurality of data partitions as local data to the programming framework.

Example 22 includes the subject matter of Example 17 or 20, wherein the one or more operations comprise facilitating the programming framework to split the local data into a plurality of micro-blocks representing a plurality of work items to be processed via an execution unit (EU) pool.

Example 23 includes the subject matter of Example 17, wherein the distribution framework and the programming model reside at a first computing device serving as a distribution server computer, wherein the distribution framework includes HadoopTM framework, and wherein the programming model includes a MapReduceTM model, wherein a job of the plurality of jobs includes a Map job and a Reduce job.

Example 24 includes the subject matter of Example 17, wherein the programming framework resides a second computing device serving as a programming client computer, wherein the programming framework includes OpenCL™ framework.

Some embodiments pertain to Example 25 includes an apparatus comprising: means for
5 detecting a request for processing of data using a programming framework and a programming model; means for facilitating interfacing between the programming framework and the programming model, wherein interfacing includes merging the programming model into the programming framework, wherein interfacing further includes integrating the programming framework with a distribution framework hosting the programming model; and means for calling
10 on the distribution framework to schedule processing of a plurality of jobs based on the request.

Example 26 includes the subject matter of Example 25, further comprising: means for splitting the data into a plurality of data partitions; and means for receiving the data, wherein interfacing is provided over a network including a cloud network.

Example 27 includes the subject matter of Example 25 or 26, further comprising: means
15 for processing the data to correspond the plurality of data partitions to the plurality of jobs; and means for assigning a plurality of job identifications (IDs) to the plurality of jobs such that each job is identified by a job ID.

Example 28 includes the subject matter of Example 25, further comprising: means for facilitating the distribution framework to dispatch the plurality of data partitions to a node pool;
20 and means for facilitating the distribution framework to schedule the plurality of jobs corresponding to the plurality of data partitions via the node pool.

Example 29 includes the subject matter of Example 25 or 28, further comprising means for facilitating the distribution framework to provide each data partition of the plurality of data partitions as local data to the programming framework.

Example 30 includes the subject matter of Example 25 or 28, further comprising means for facilitating the programming framework to split the local data into a plurality of micro-blocks representing a plurality of work items to be processed via an execution unit (EU) pool.

5 Example 31 includes the subject matter of Example 25, wherein the distribution framework and the programming model reside at a first computing device serving as a distribution server computer, wherein the distribution framework includes HadoopTM framework, and wherein the programming model includes a MapReduceTM model, wherein a job of the plurality of jobs includes a Map job and a Reduce job.

10 Example 32 includes the subject matter of Example 25, wherein the programming framework resides a second computing device serving as a programming client computer, wherein the programming framework includes OpenCLTM framework.

Example 33 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.

15 Example 34 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 35 includes a system comprising a mechanism to implement or perform a method as claimed in any of claims or examples 9-16.

20 Example 36 includes an apparatus comprising means for performing a method as claimed in any of claims or examples 9-16.

Example 37 includes a computing device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

25 Example 38 includes a communications device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

Example 39 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 40 includes at least one non-transitory or tangible machine-readable medium
5 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 41 includes a system comprising a mechanism to implement or perform a method or realize an apparatus as claimed in any preceding claims.

Example 42 includes an apparatus comprising means to perform a method as claimed in
10 any preceding claims.

Example 43 includes a computing device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

Example 44 includes a communications device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

15 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
20 herein. Moreover, the actions of any flow diagram need not be implemented in the order shown; nor do all of the acts necessarily need to be performed. Also, those acts that are not dependent on other acts may be performed in parallel with the other acts. The scope of embodiments is by no means limited by these specific examples. Numerous variations, whether explicitly given in the specification or not, such as differences in structure, dimension, and use of material, are
25 possible. The scope of embodiments is at least as broad as given by the following claims.

CLAIMS

What is claimed is:

1. An apparatus to facilitate efficient communication and data processing across
5 clusters of computing machines in a heterogeneous computing environment, comprising:
detection/reception logic to detect a request for processing of data using a programming
framework and a programming model;
interfacing logic to facilitate interfacing between the programming framework and the
programming model, wherein interfacing includes merging the programming model into the
10 programming framework, wherein interfacing further includes integrating the programming
framework with a distribution framework hosting the programming model; and
execution/calling logic to call on the distribution framework to schedule processing of a
plurality of jobs based on the request.
2. The apparatus of claim 1, further comprising partition/vector logic to split the data
15 into a plurality of data partitions, wherein the detection/reception logic is further to receive the
data, wherein interfacing is provided over a network including a cloud network.
3. The apparatus of claim 1 or 2, further comprising processing logic to process the
data to correspond the plurality of data partitions to the plurality of jobs, and assign a plurality of
job identifications (IDs) to the plurality of jobs such that each job is identified by a job ID.
- 20 4. The apparatus of claim 1, wherein the execution/calling logic is further to
facilitate the distribution framework to dispatch the plurality of data partitions to a node pool,
wherein the execution/calling logic is further to facilitate the distribution framework to schedule
the plurality of jobs corresponding to the plurality of data partitions via the node pool.
5. The apparatus of claim 1 or 4, wherein the execution/calling logic is further to
25 facilitate the distribution framework to provide each data partition of the plurality of data
partitions as local data to the programming framework.

6. The apparatus of claim 1 or 4, wherein the execution/calling logic is further to facilitate the programming framework to split the local data into a plurality of micro-blocks representing a plurality of work items to be processed via an execution unit (EU) pool.

7. The apparatus of claim 1, wherein the distribution framework and the programming model reside at a first computing device serving as a distribution server computer, wherein the distribution framework includes HadoopTM framework, and wherein the programming model includes a MapReduceTM model, wherein a job of the plurality of jobs includes a Map job and a Reduce job.

8. The apparatus of claim 1, wherein the programming framework resides a second computing device serving as a programming client computer, wherein the programming framework includes OpenCLTM framework.

9. A method for facilitating efficient communication and data processing across clusters of computing machines in a heterogeneous computing environment, comprising:

detecting a request for processing of data using a programming framework and a programming model;

facilitating interfacing between the programming framework and the programming model, wherein interfacing includes merging the programming model into the programming framework, wherein interfacing further includes integrating the programming framework with a distribution framework hosting the programming model; and

calling on the distribution framework to schedule processing of a plurality of jobs based on the request.

10. The method of claim 9, further comprising:

splitting the data into a plurality of data partitions; and

receiving the data, wherein interfacing is provided over a network including a cloud

network.

11. The method of claim 9, further comprising:

processing the data to correspond the plurality of data partitions to the plurality of jobs;
and

assigning a plurality of job identifications (IDs) to the plurality of jobs such that each job is identified by a job ID.

5 12. The method of claim 9, further comprising:
 facilitating the distribution framework to dispatch the plurality of data partitions to a node pool; and

 facilitating the distribution framework to schedule the plurality of jobs corresponding to the plurality of data partitions via the node pool.

10 13. The method of claim 12, further comprising facilitating the distribution framework to provide each data partition of the plurality of data partitions as local data to the programming framework.

 14. The method of claim 12, further comprising facilitating the programming framework to split the local data into a plurality of micro-blocks representing a plurality of work
15 items to be processed via an execution unit (EU) pool.

 15. The method of claim 9, wherein the distribution framework and the programming model reside at a first computing device serving as a distribution server computer, wherein the distribution framework includes HadoopTM framework, and wherein the programming model includes a MapReduceTM model, wherein a job of the plurality of jobs includes a Map job and a
20 Reduce job.

 16. The method of claim 9, wherein the programming framework resides a second computing device serving as a programming client computer, wherein the programming framework includes OpenCLTM framework.

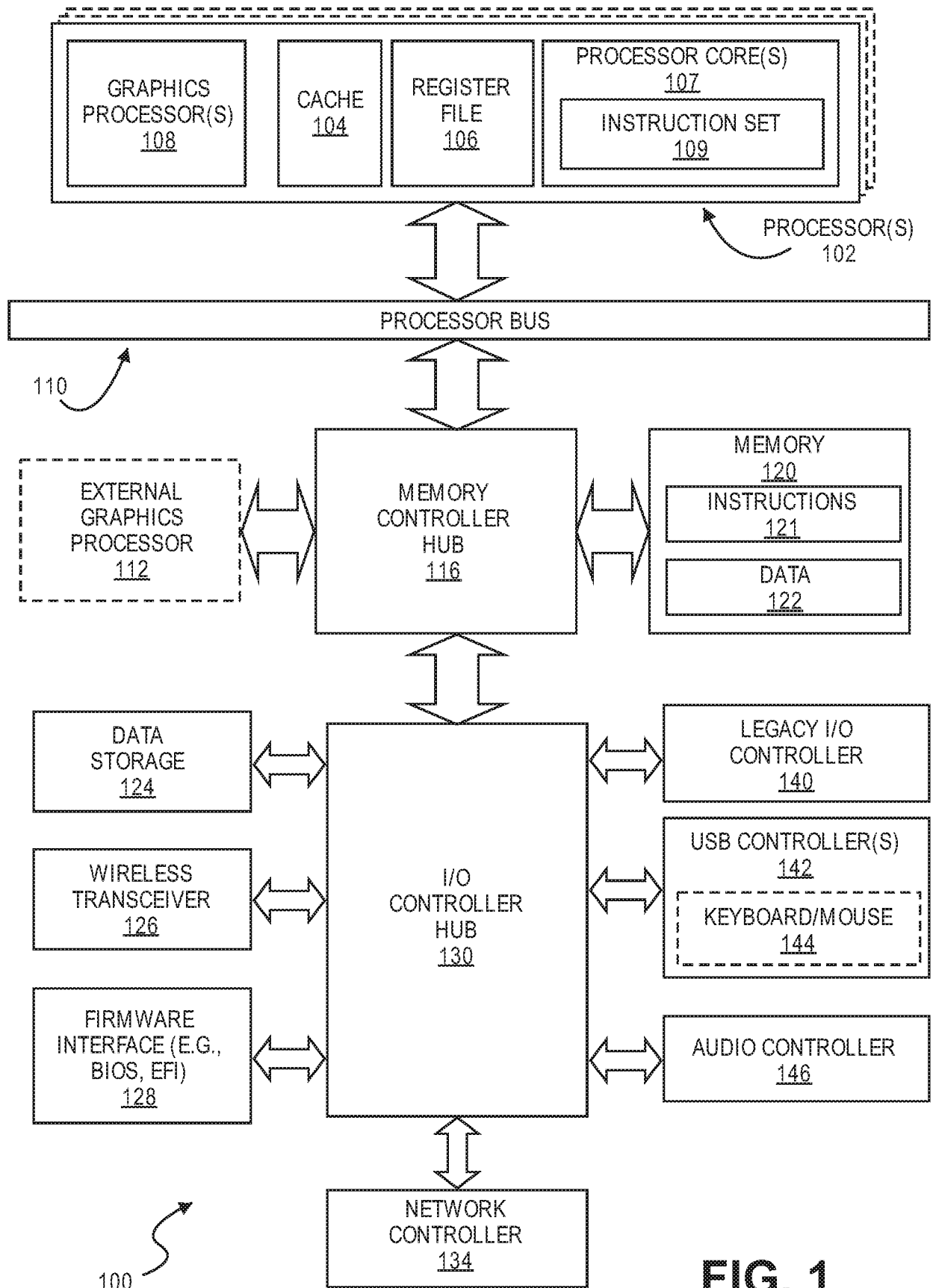
 17. At least one machine-readable medium comprising a plurality of instructions,
25 when executed on a computing device, to implement or perform a method as claimed in any of claims 9-16.

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

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

5 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

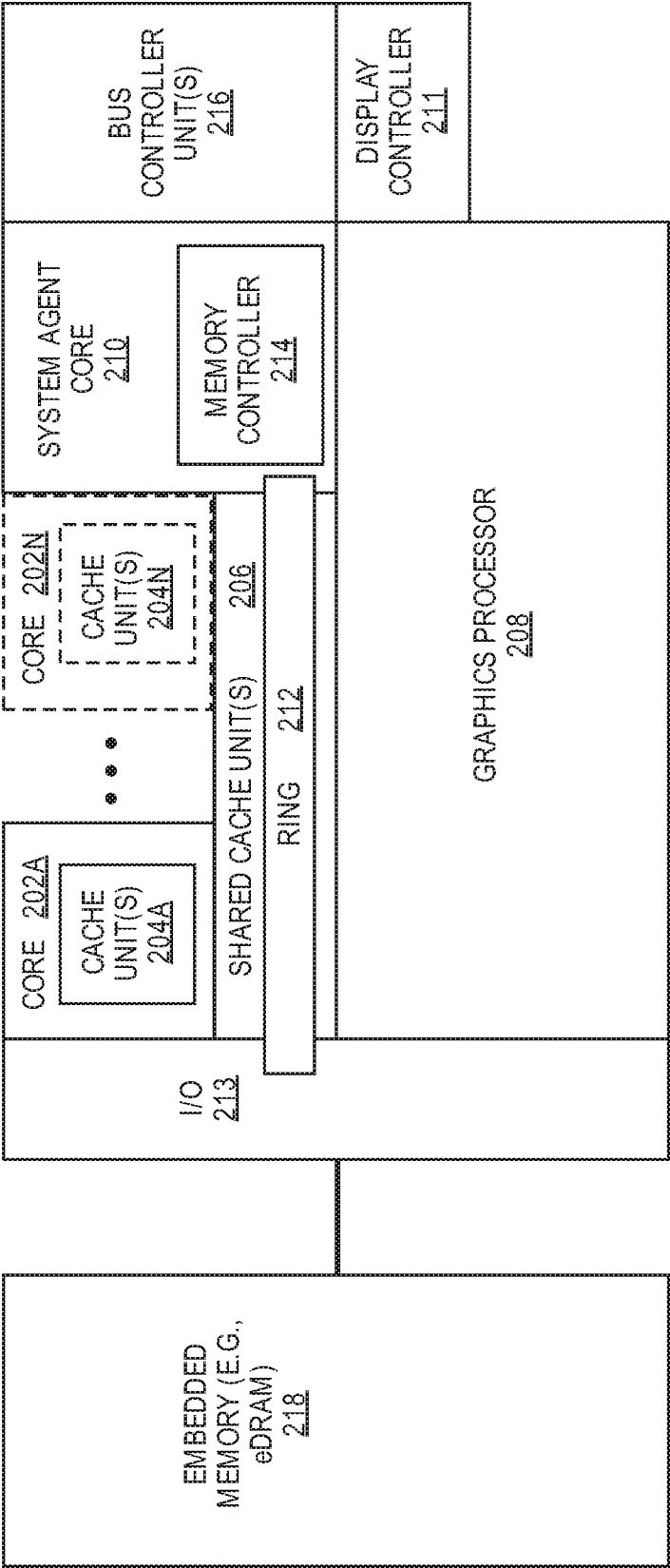
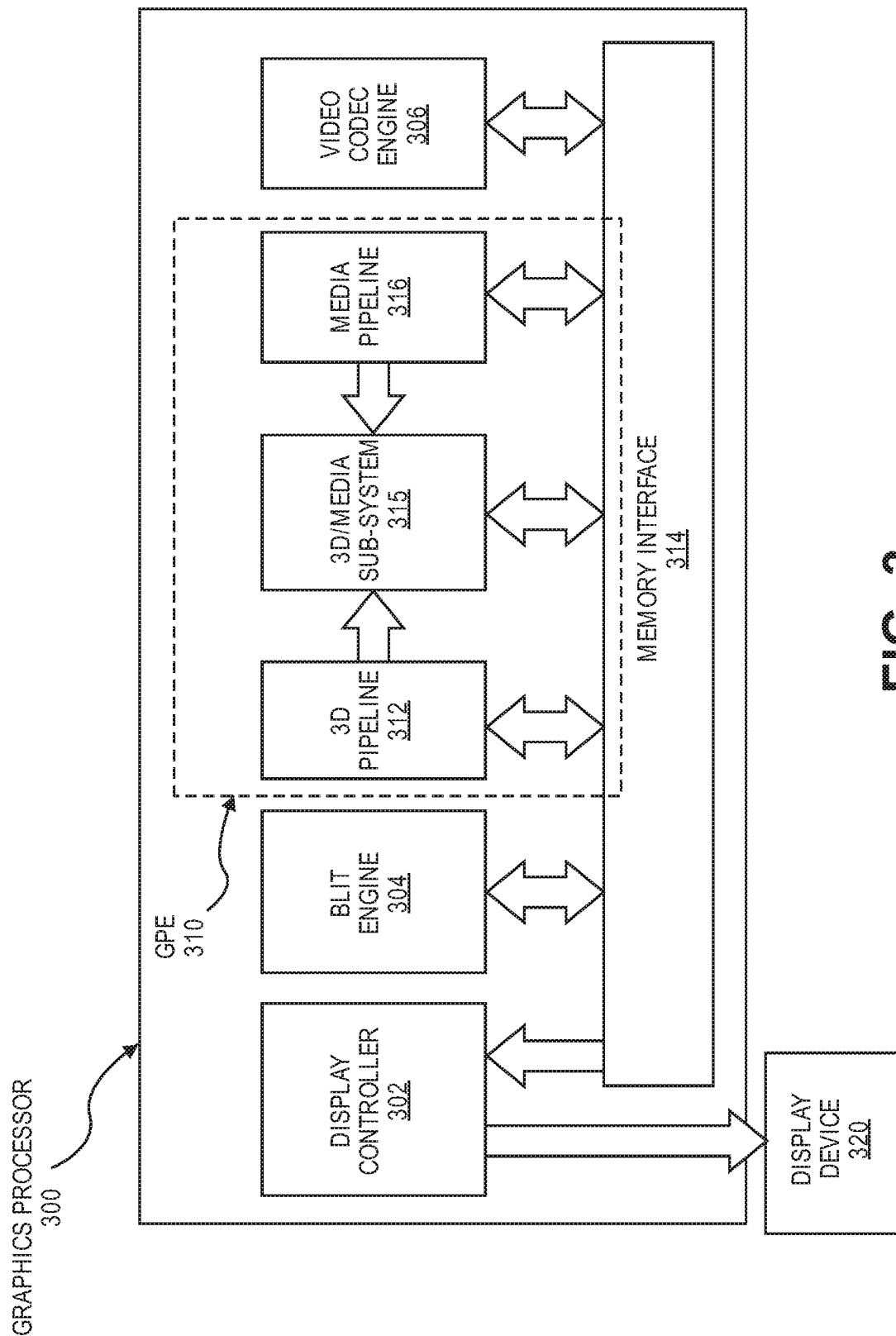


FIG. 2



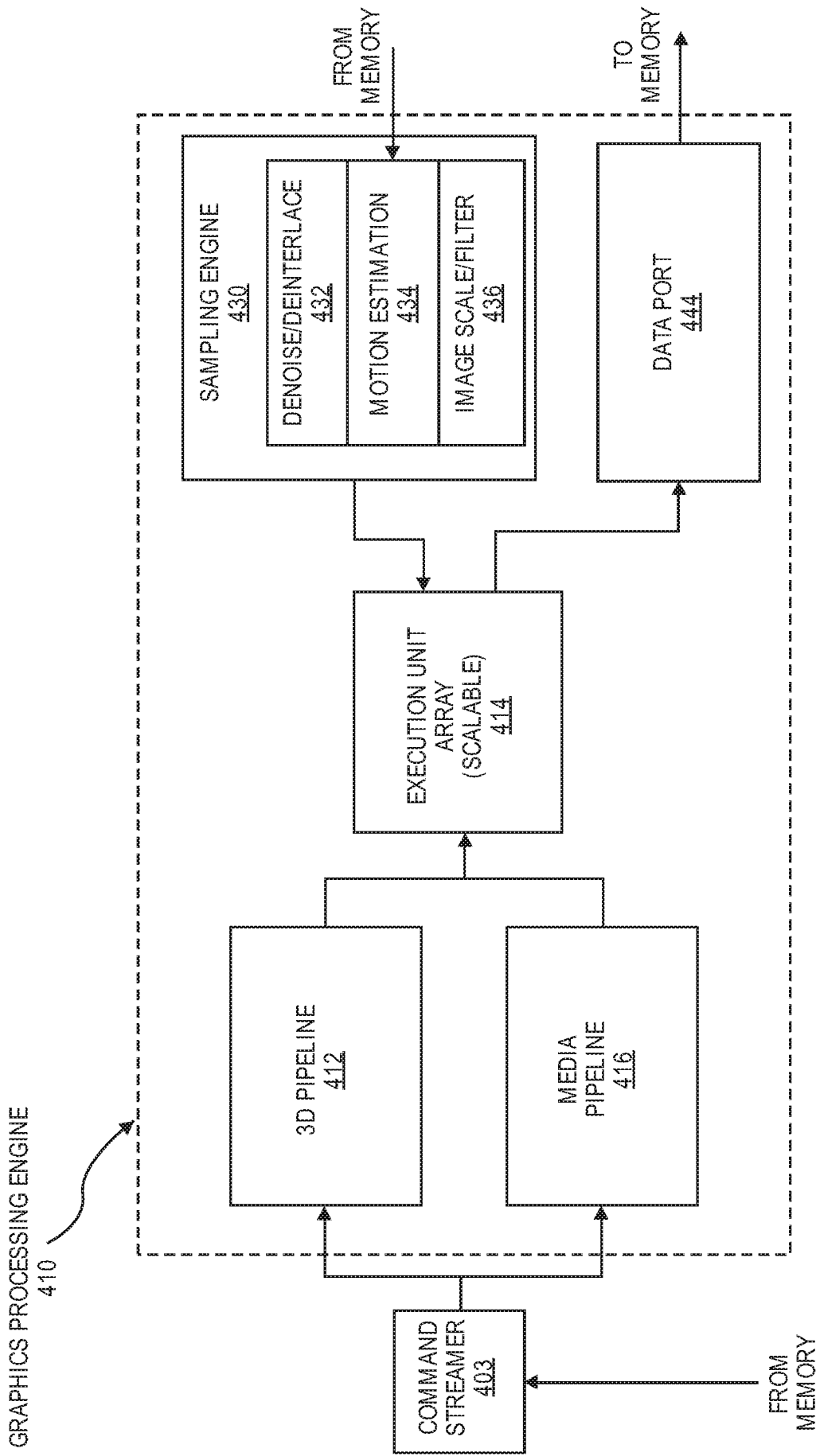
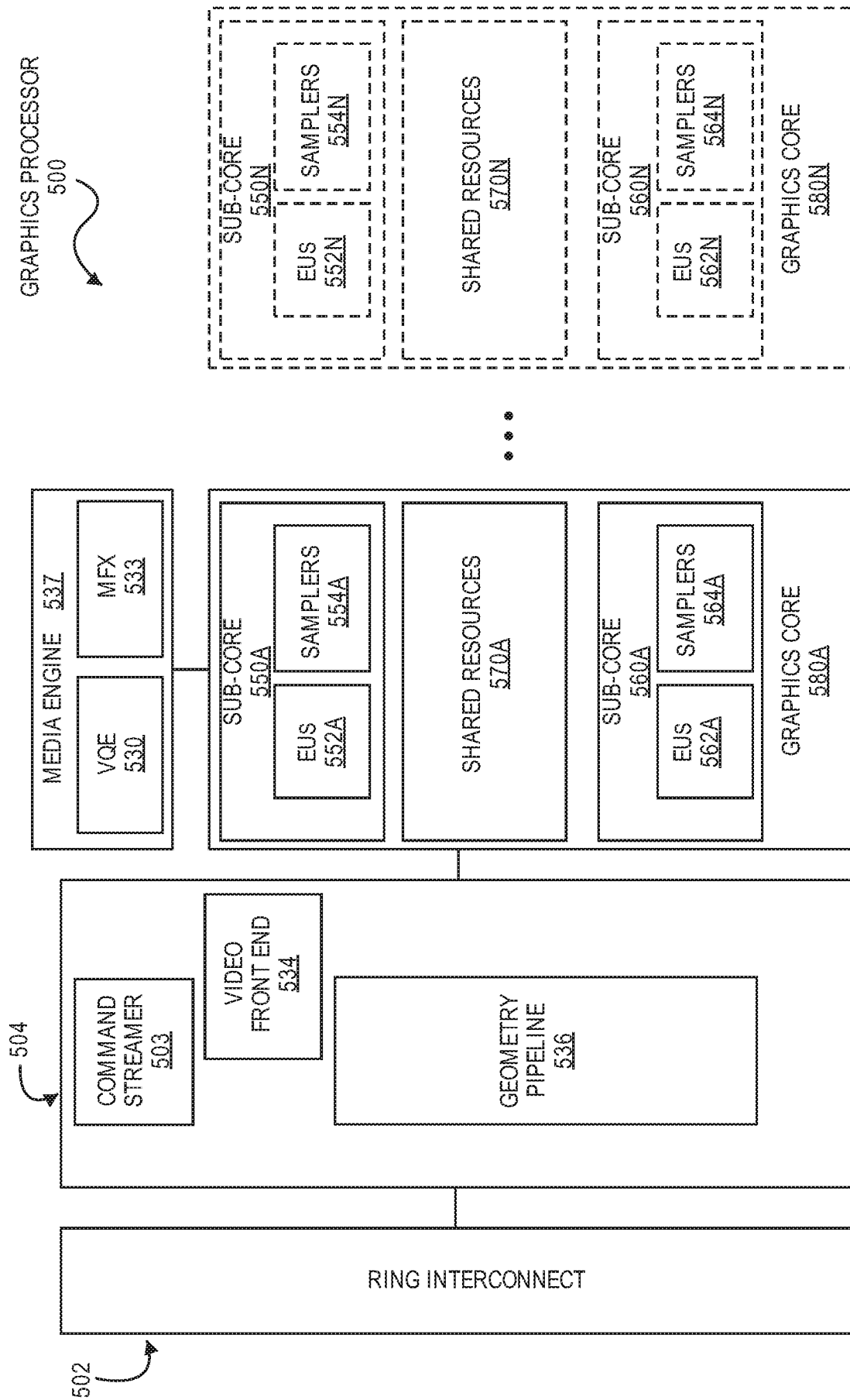


FIG. 4



LE

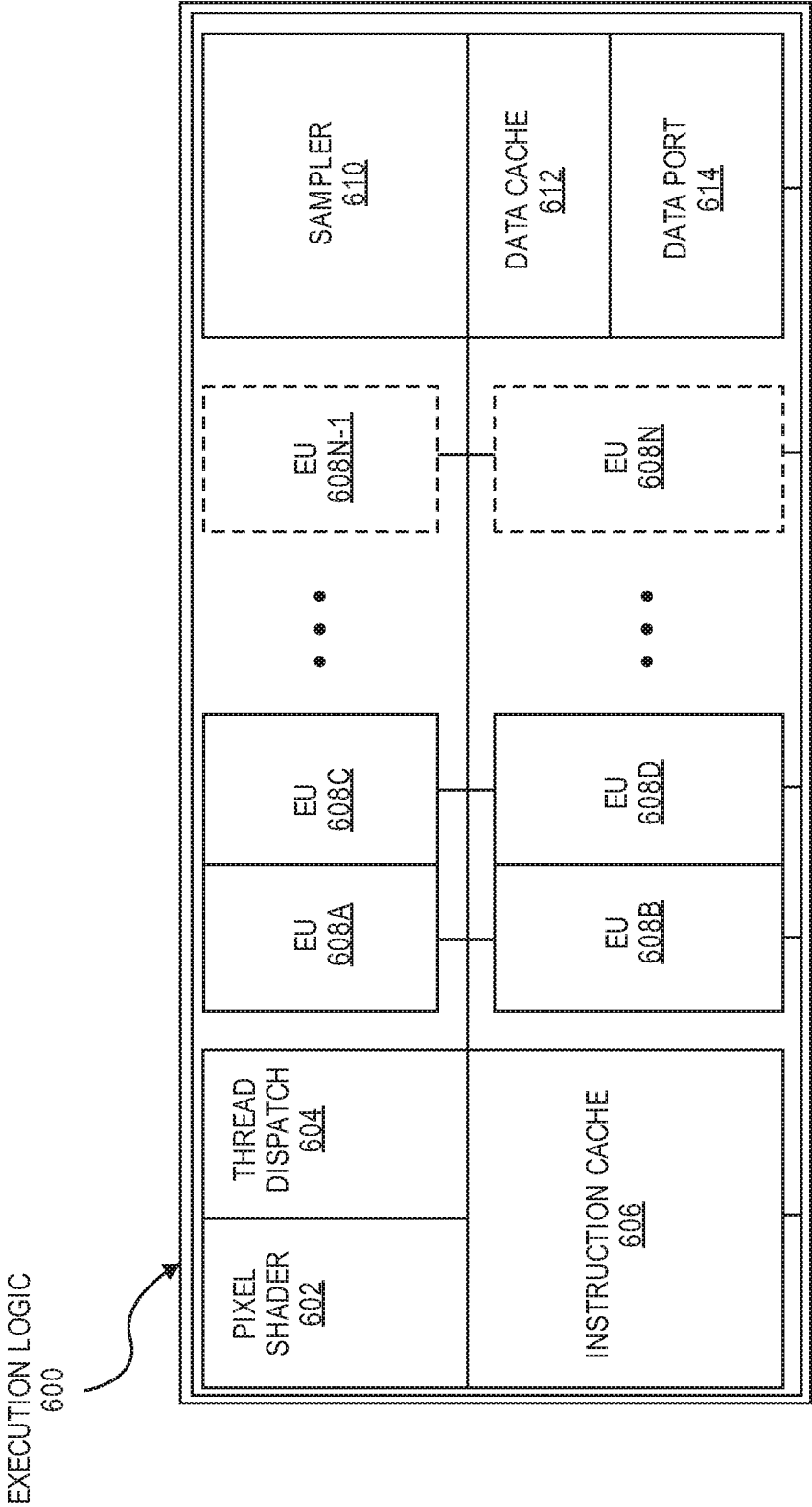
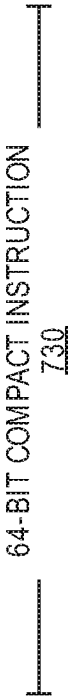
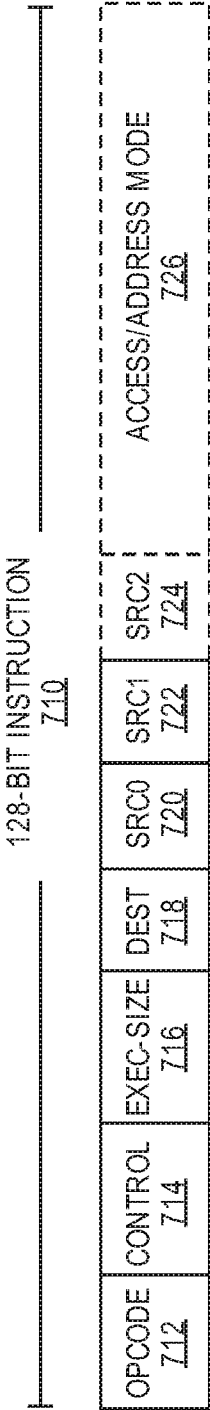


FIG. 6

GRAPHICS CORE INSTRUCTION FORMATS

700



OPCODE DECODE

740

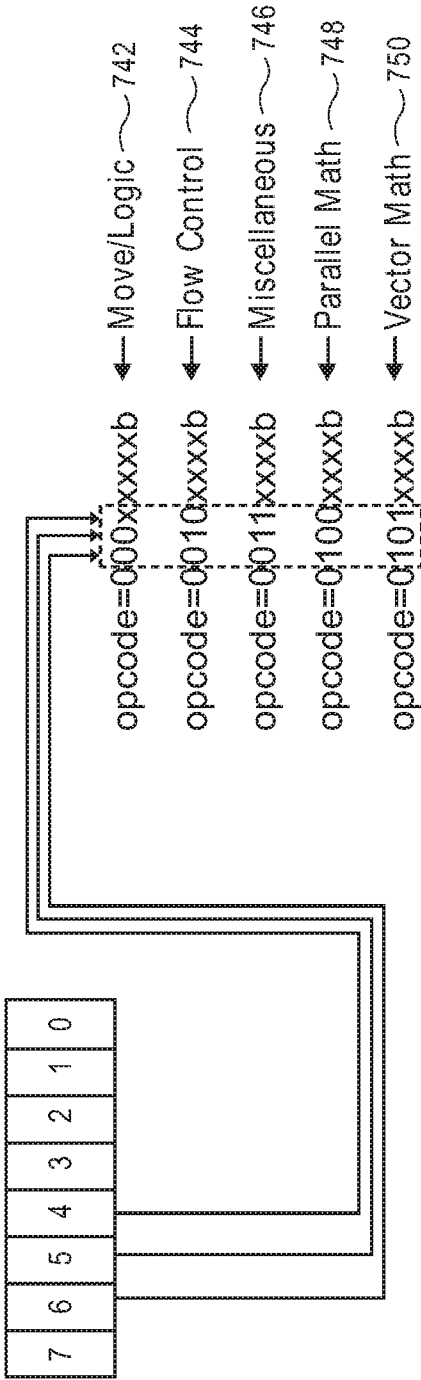


FIG. 7

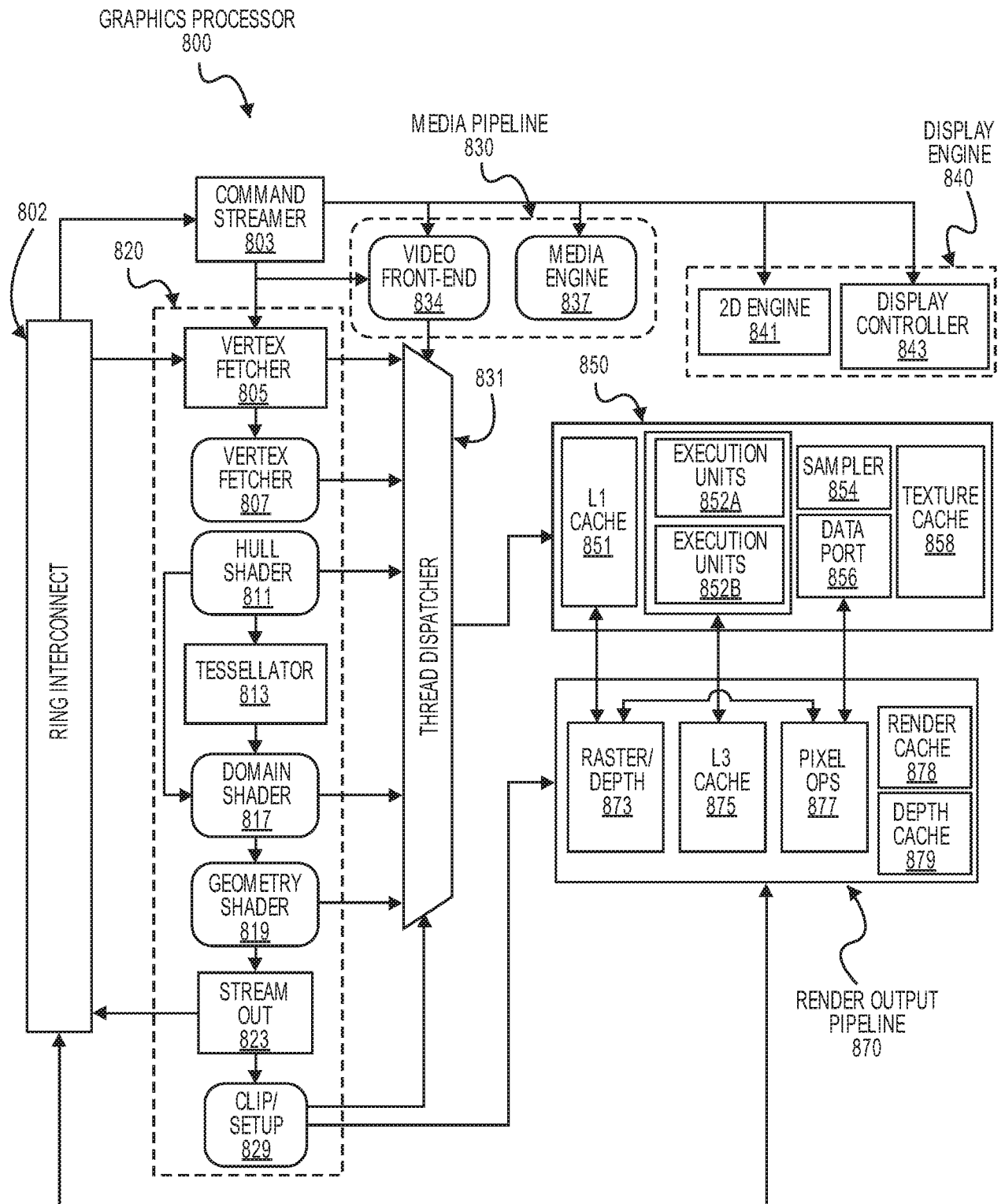


FIG. 8

FIG. 9A **GRAPHICS PROCESSOR COMMAND FORMAT**
900

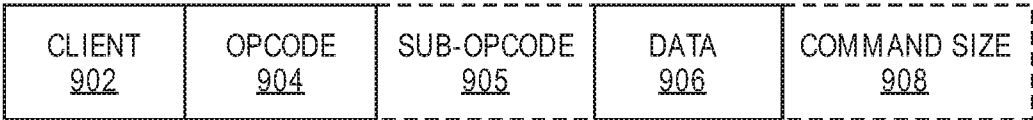
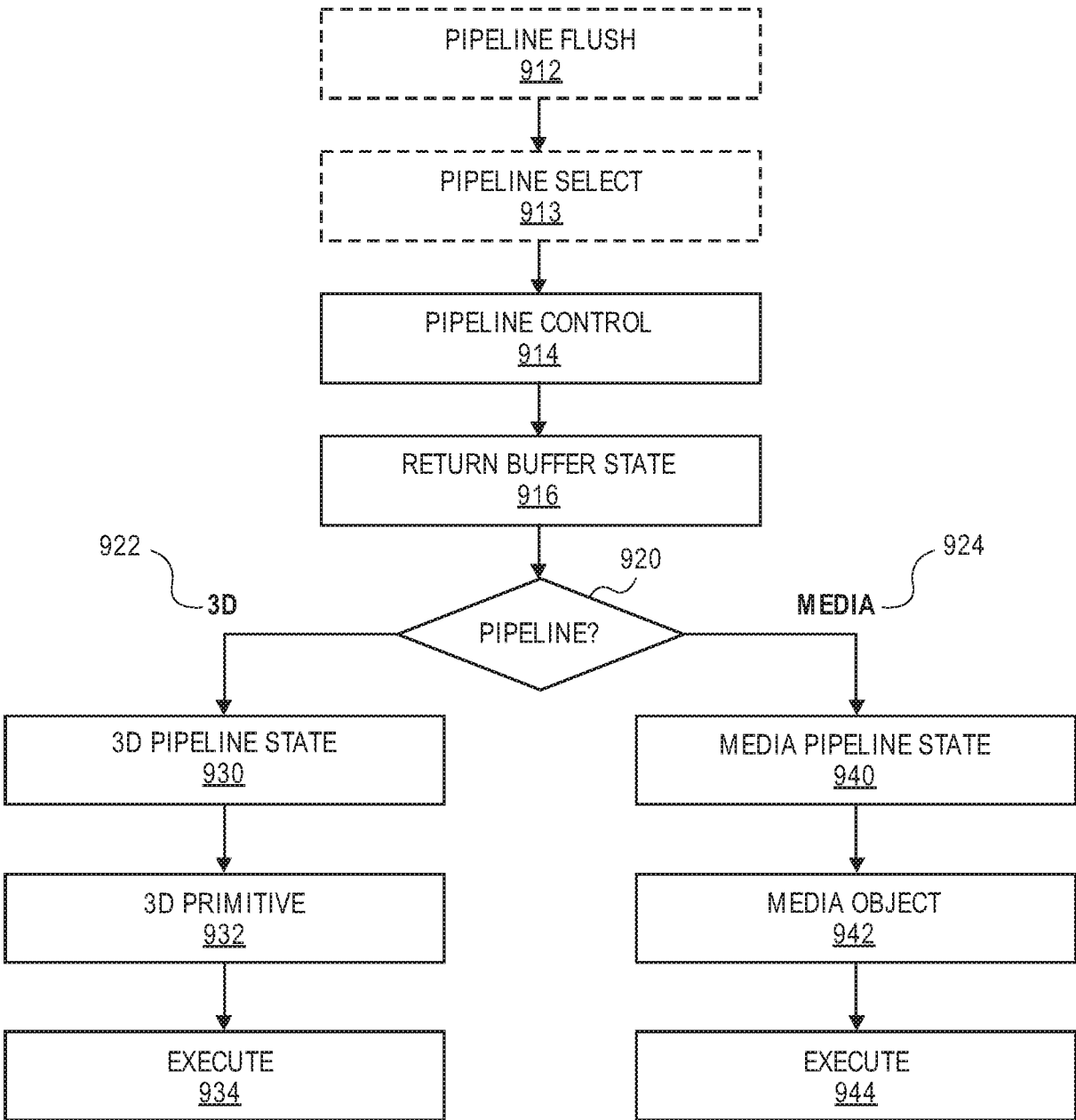
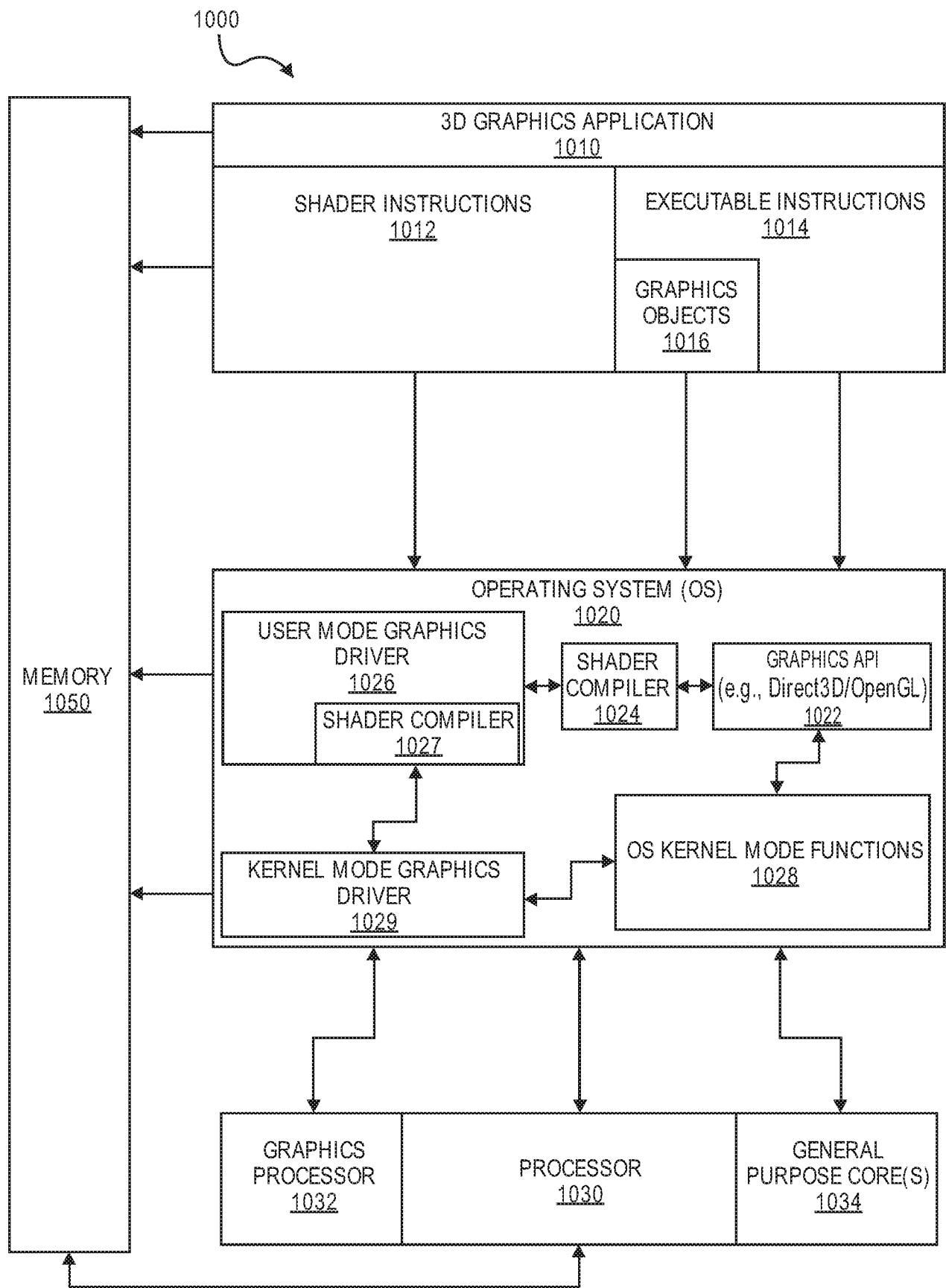


FIG. 9B **GRAPHICS PROCESSOR COMMAND SEQUENCE**
910



**FIG. 10**

IP CORE DEVELOPMENT - 1100

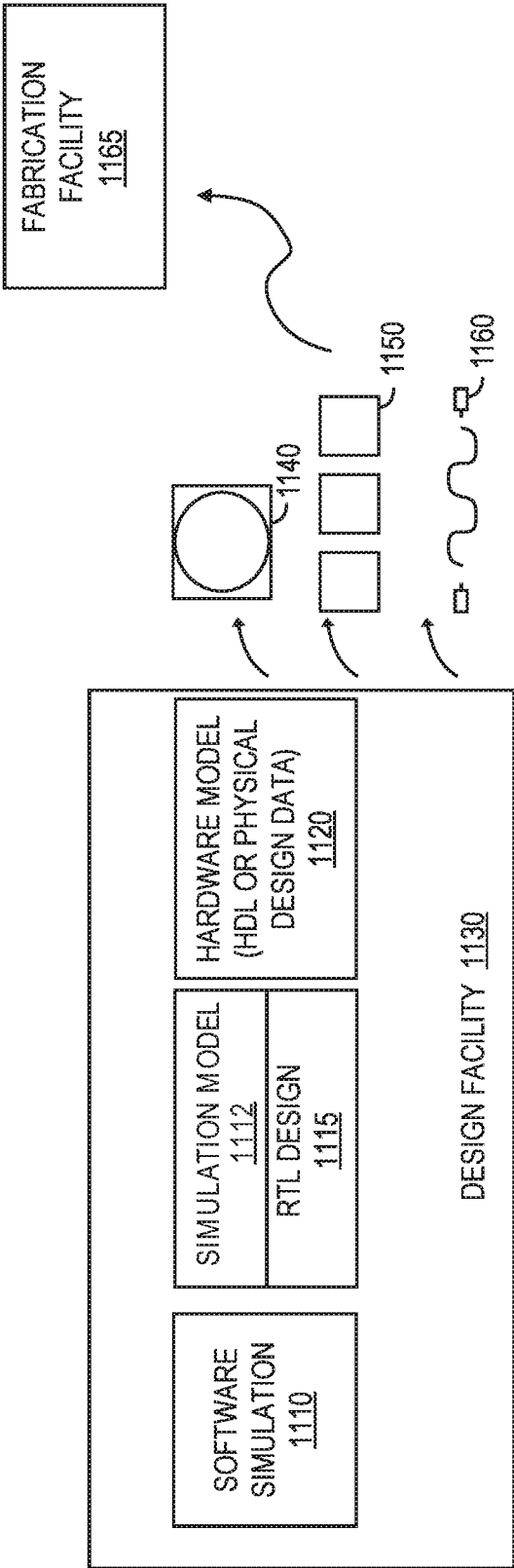
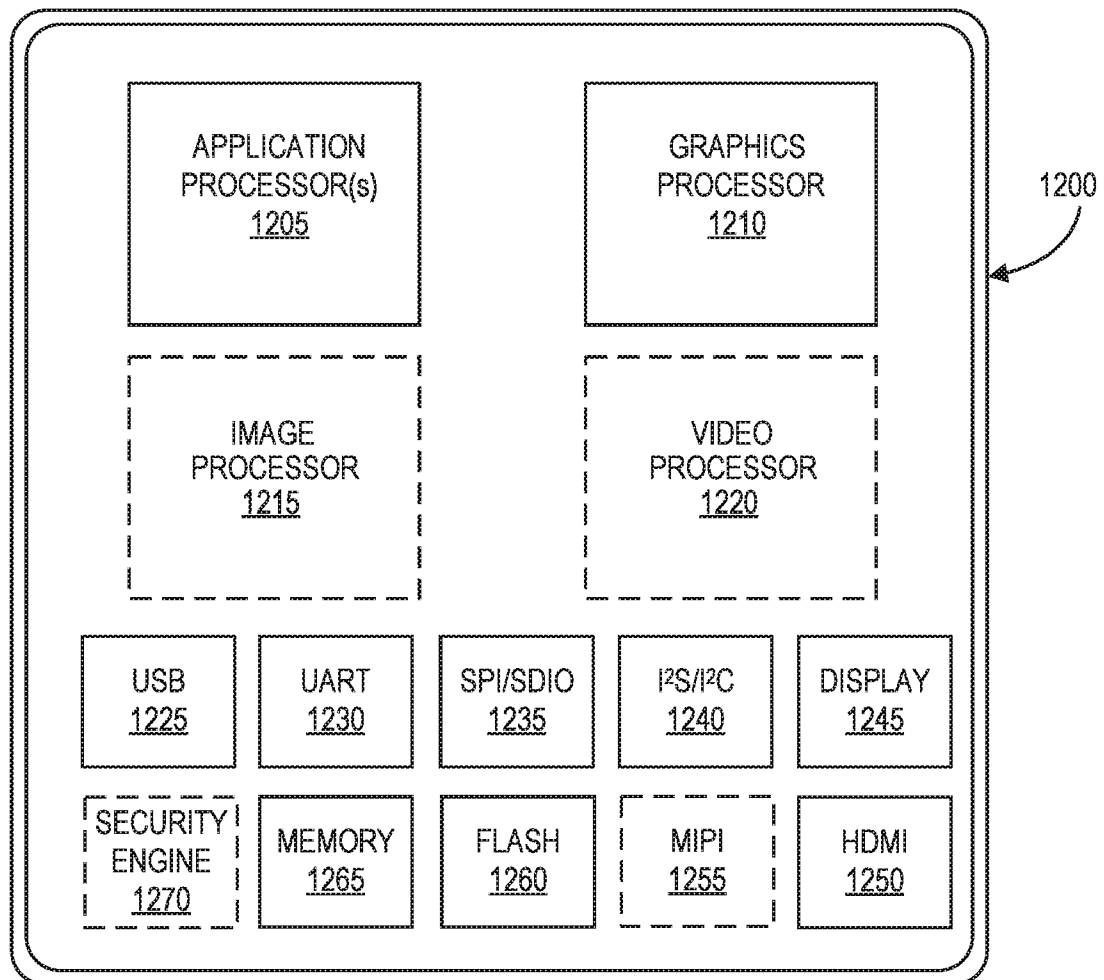
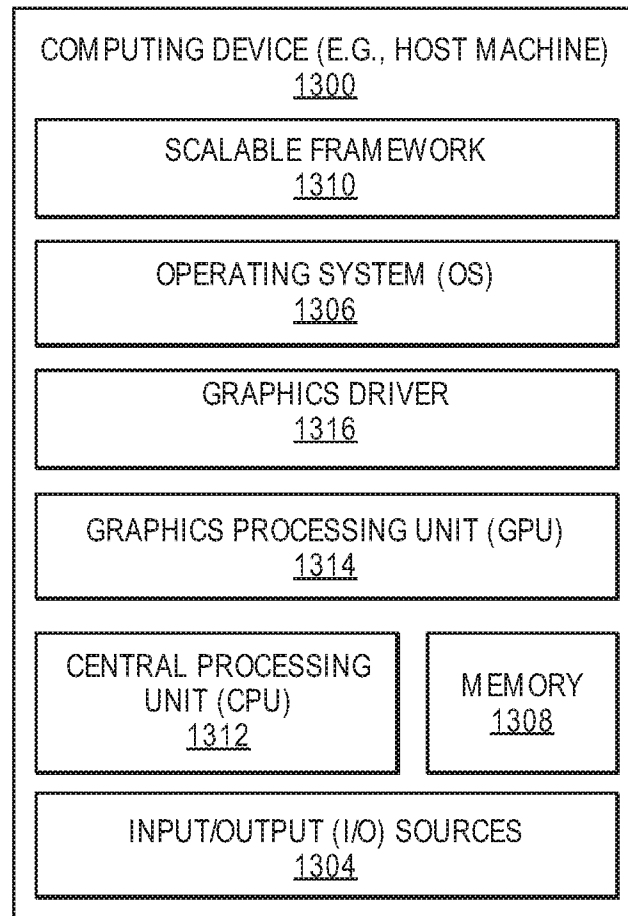
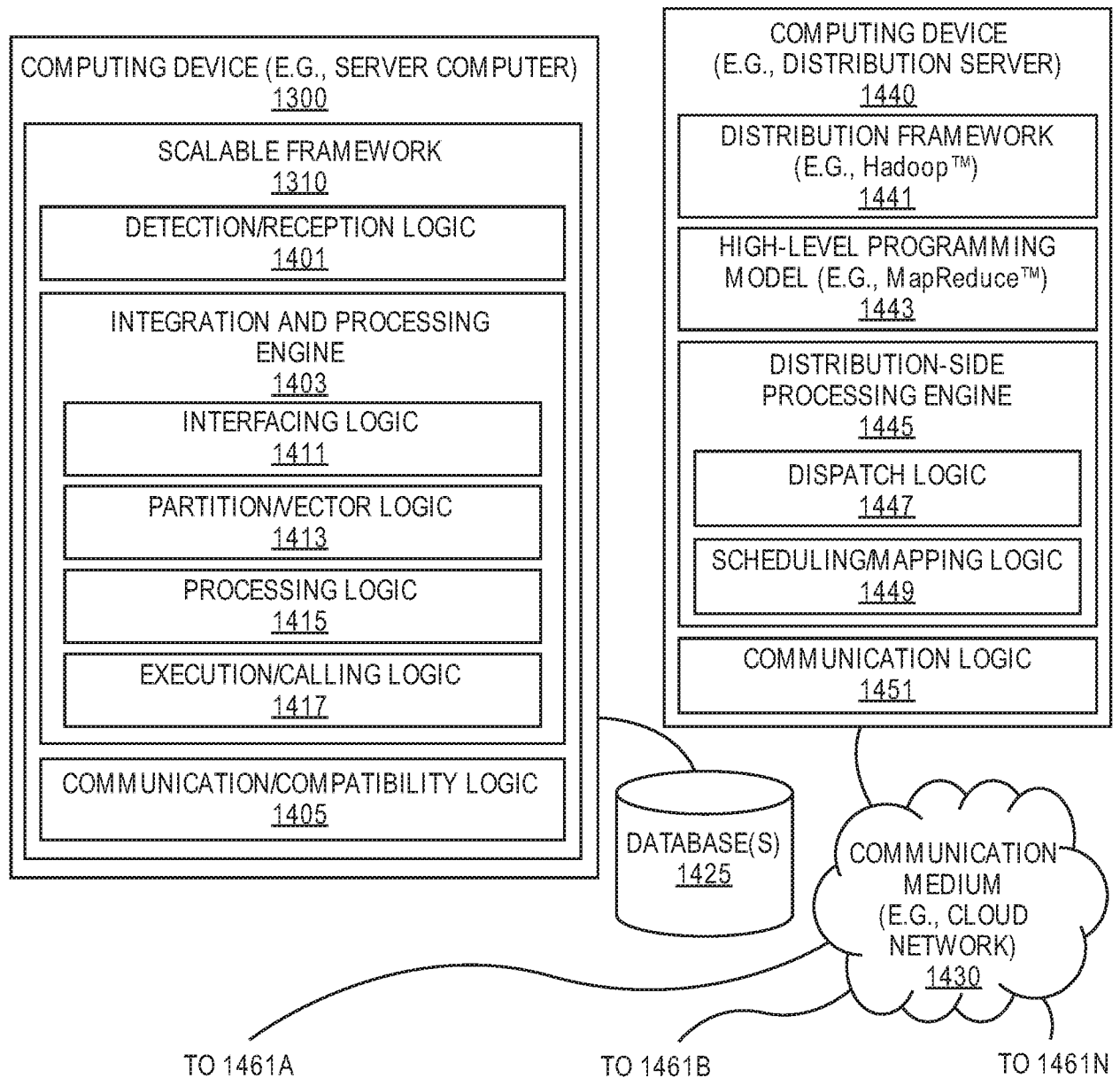


FIG. 11

**FIG. 12**

**FIG. 13**

**FIG. 14A**

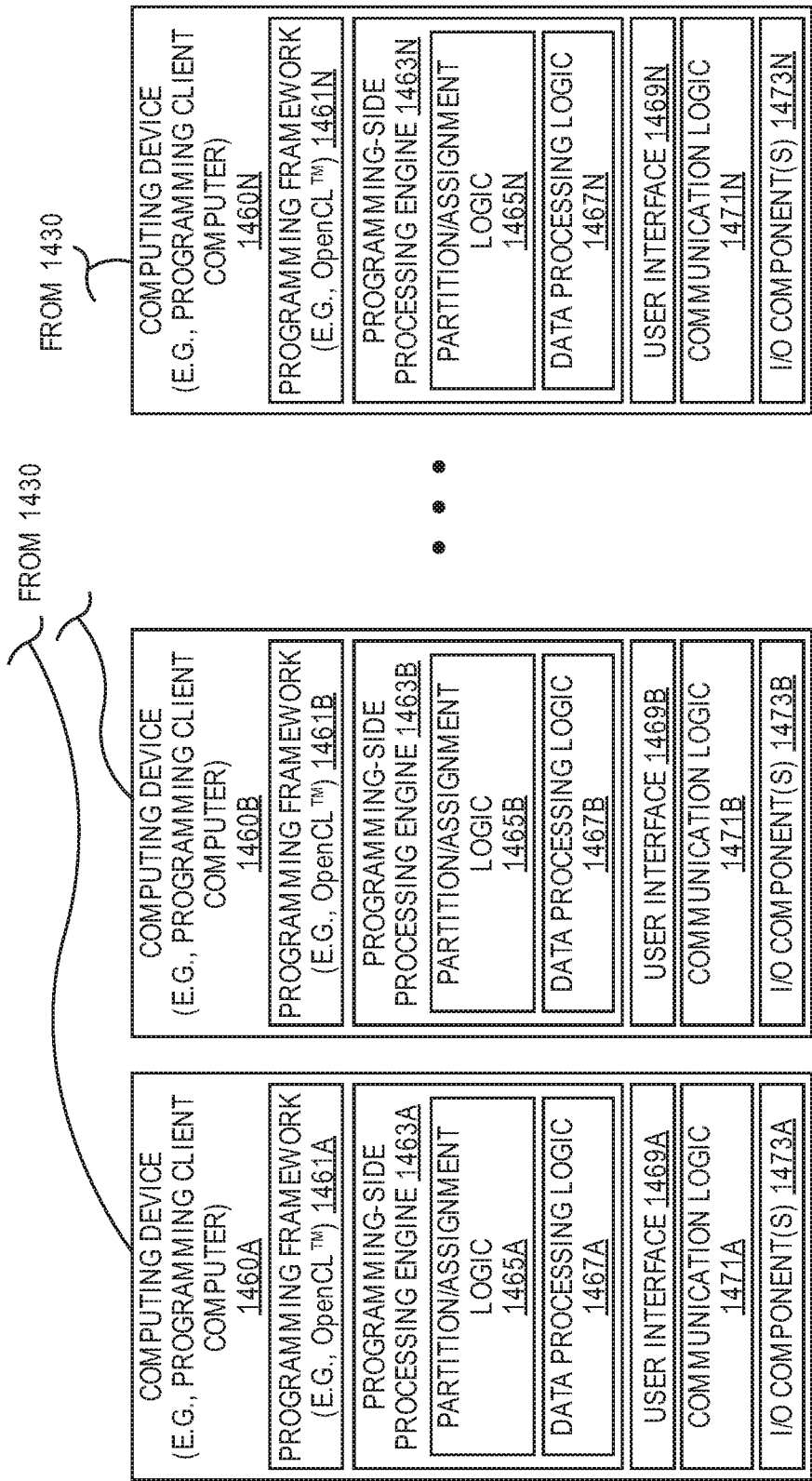


FIG. 14A (CONT.)

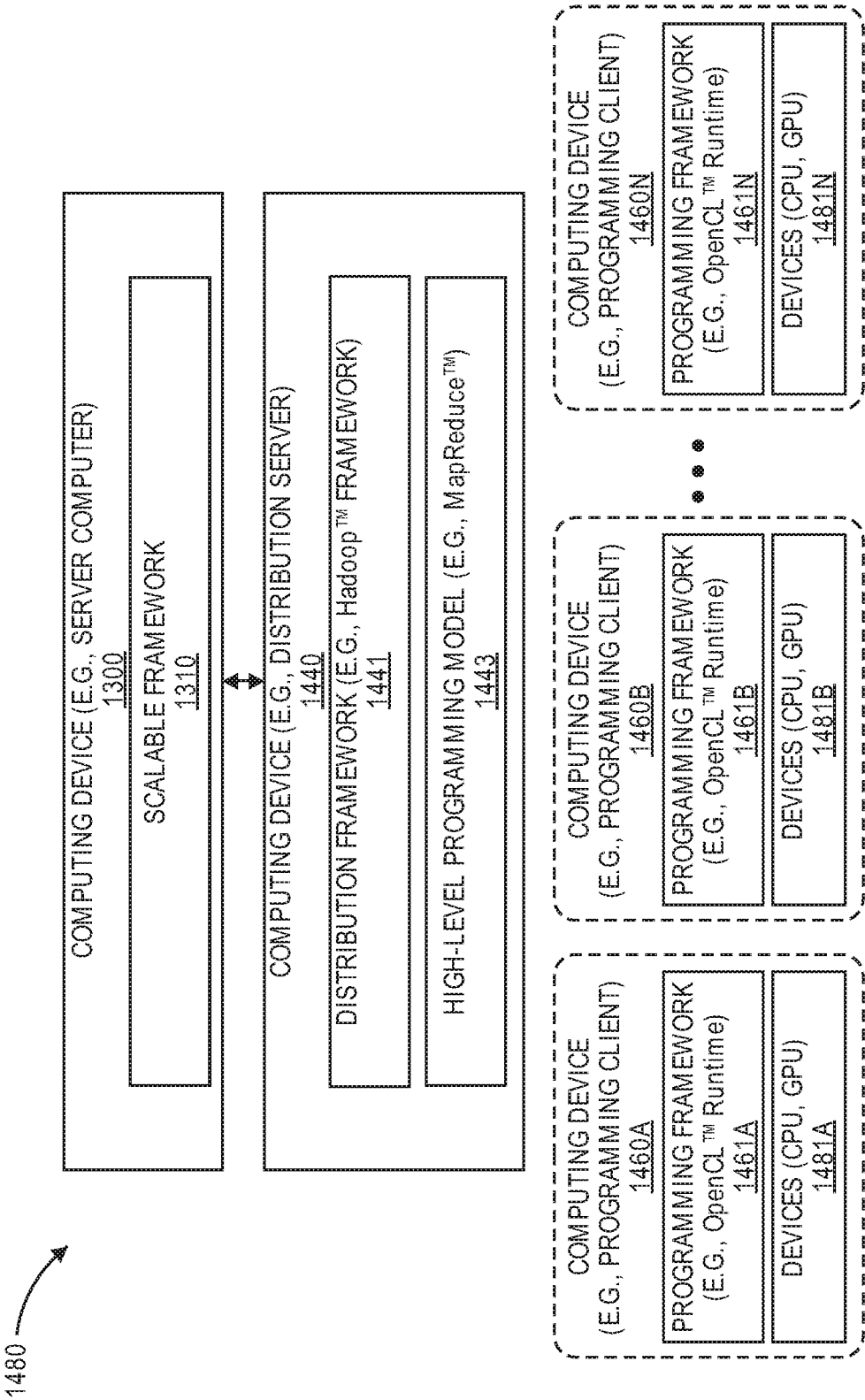
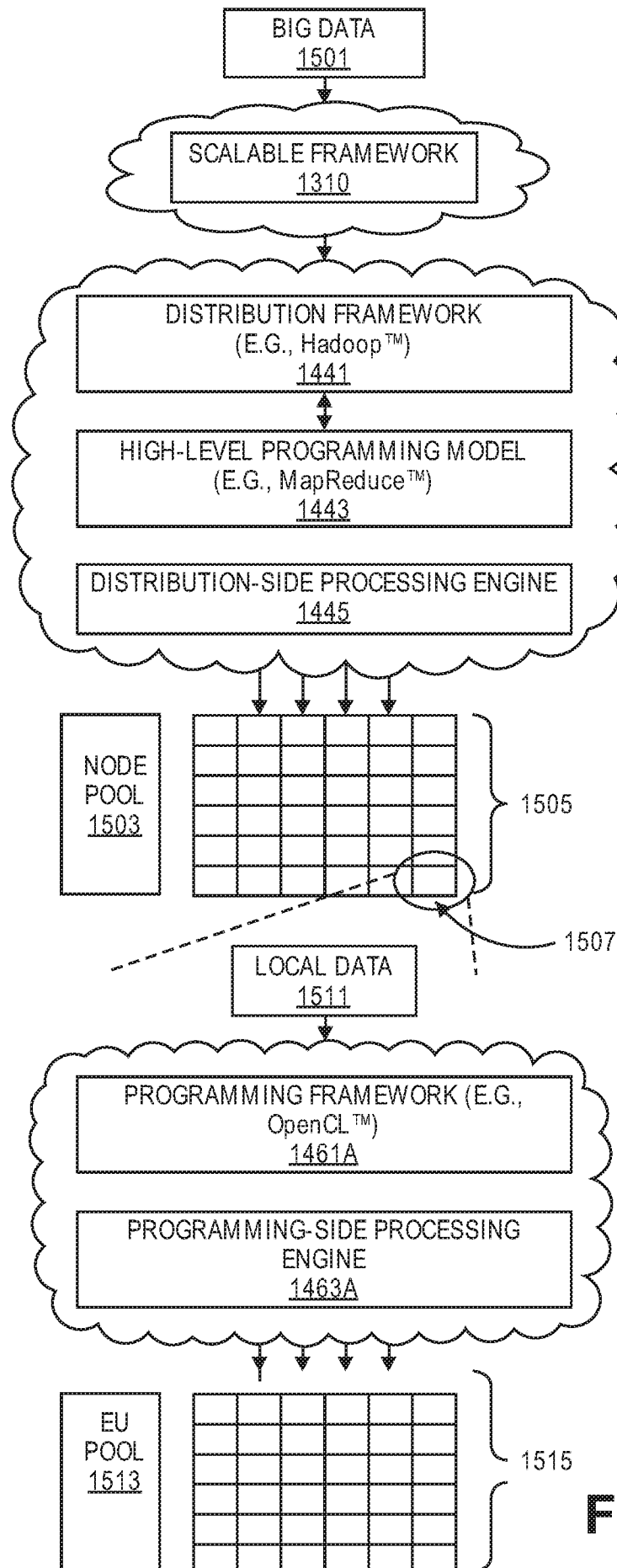


FIG. 14B

**FIG. 15A**

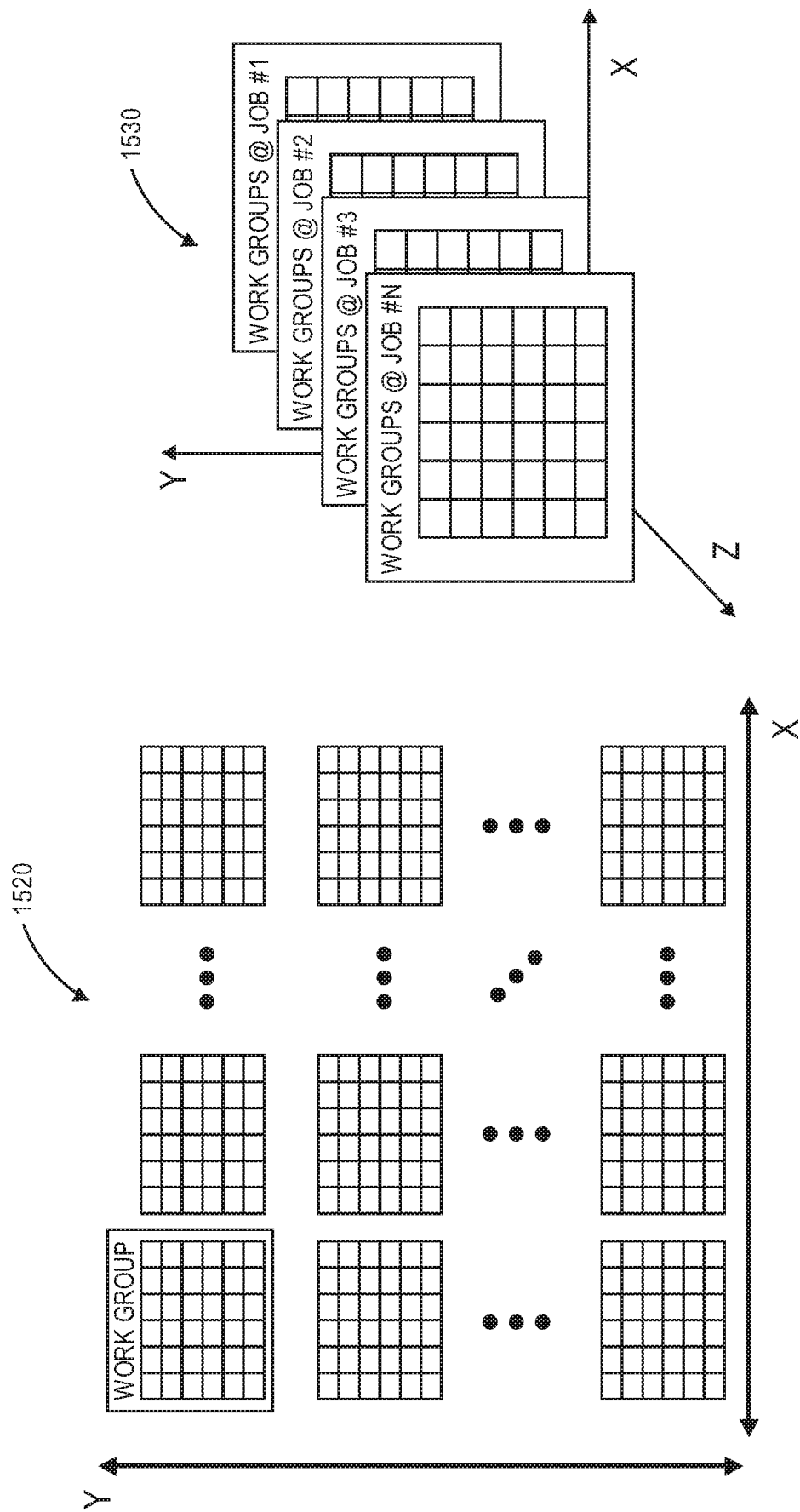


FIG. 15B

1530

```

__kernel void VectorAddition (
    __global uint4* A,
    __global uint4* B,
    __global uint4* C)
{
    // Calculate block offset in global data
    unsigned int gid = get_global_id(0);
    unsigned int gindex = gid * 128;

    //Add and output each element
    for(unsigned int i =0 ; i< 128 ; i++)
    {
        C[gindex + i] = A[gindex + i] + B[gindex + i];
    }
}

```

1540

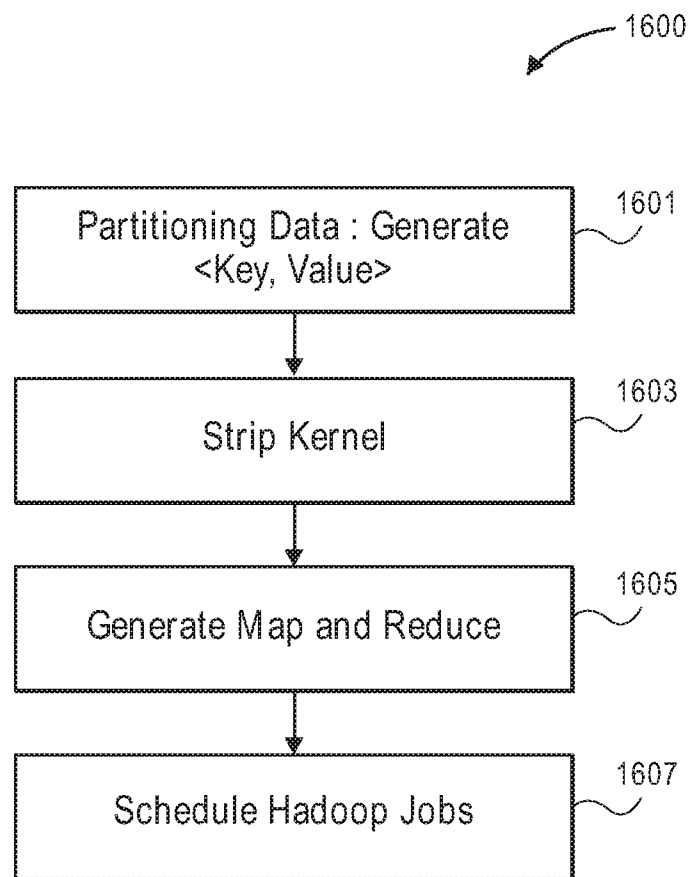
```

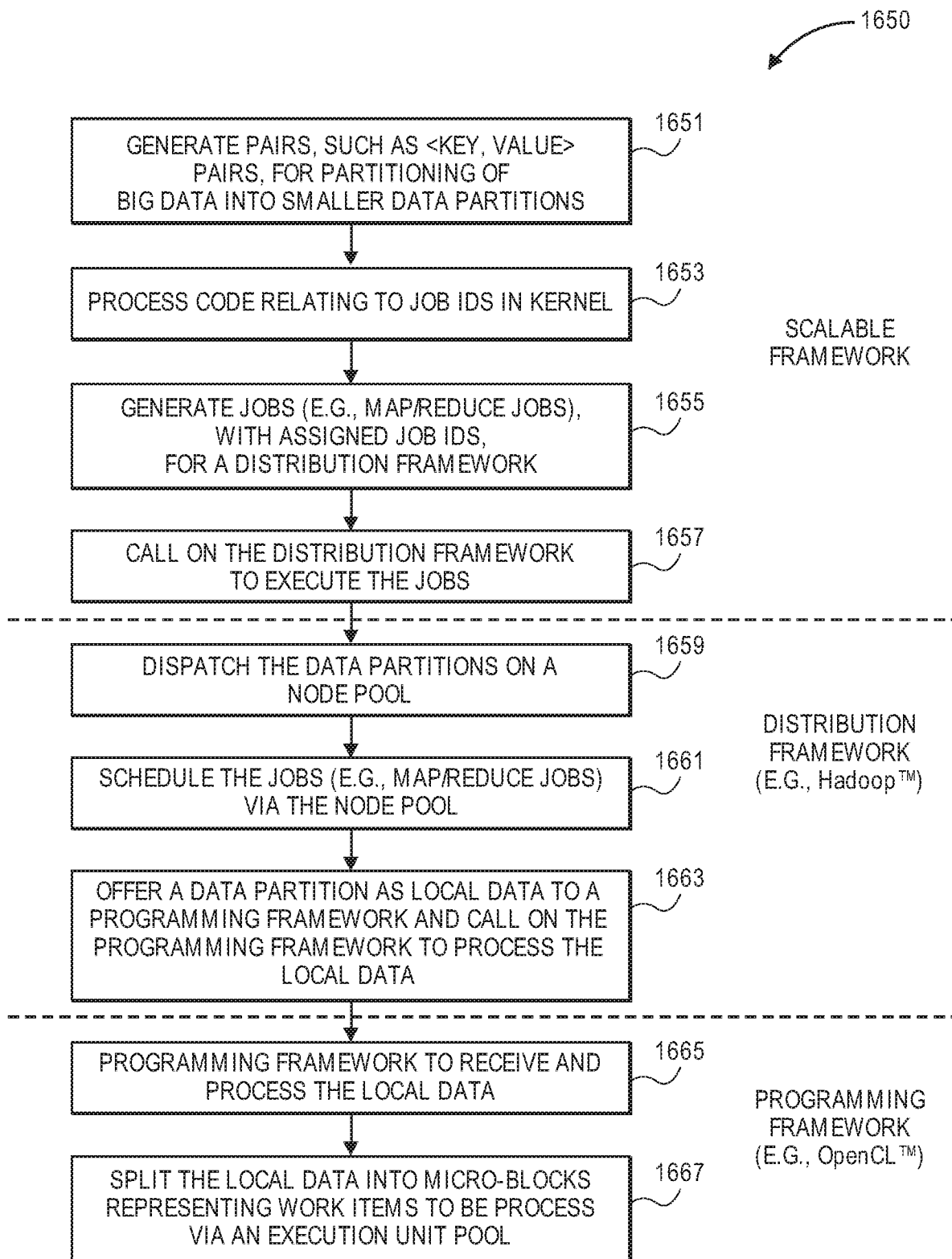
__kernel void VectorAdditionSCL(
    __global uint4* A,
    __global uint4* B,
    __global uint4* A,
{
    // Get id in work group and jobid in job group
    1541 unsigned int gid = get_global_id(0);
    unsigned int jobid = get_job_id(0);
    unsigned int gindex = gid * 128 + jobid * 256 *102

    //Add and output each element
    for(unsigned int i =0 ; i< 128 ; i++)
    {
        C[gindex + i] = A[gindex + i] + B[gindex + i];
    }
}

```

FIG. 15C

**FIG. 16A**

**FIG. 16B**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/CN2015/098634

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/46(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

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

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

CNKI, CNPAT, EPODOC, WPI: heterogeneous, comput+, model, program+, framework, interfac???, distribut+, process???, merg???, integrat???, job?, host???, hadoop, mapreduce, opencl, spark

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2010011374 A1 (AJOU UNIVERSITY INDUSTRY COOPERATION FOUNDATION) 14 January 2010 (2010-01-14) description paragraphs [0022]-[0052]	1-21
A	US 2015295731 A1 (CISCO TECHNOLOGY, INC.) 15 October 2015 (2015-10-15) the whole document	1-21
A	US 7039912 B1 (APPLE COMPUTER, INC.) 02 May 2006 (2006-05-02) the whole document	1-21
A	US 6240512 B1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 29 May 2001 (2001-05-29) the whole document	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

15 August 2016

Date of mailing of the international search report

01 September 2016

Name and mailing address of the ISA/CN

STATE INTELLECTUAL PROPERTY OFFICE OF THE
P.R.CHINA
6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing
100088, China

Facsimile No. (86-10)62019451

Authorized officer

TIAN, Jing

Telephone No. (86-10)62413701

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.

PCT/CN2015/098634

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	2010011374	A1	14 January 2010	KR	20080064587	A	09 July 2008
				WO	2008082021	A1	10 July 2008
US	2015295731	A1	15 October 2015	None			
US	7039912	B1	02 May 2006	US	2006206870	A1	14 September 2006
US	6240512	B1	29 May 2001	None			