(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 March 2017 (30.03.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/049583 A1

(51) International Patent Classification:
G06F 13/00 (2006.01)

(21) International Application Number:
PCT/CN2015/090701

(22) International Filing Date:
25 September 2015 (25.09.2015)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventors; and

(71) Applicants *(for US only)*: **KONG, Lingyi** [CN/CN]; Zizhu2 2W062, Zizhu2 IT Flex Lab, Shanghai 200052 (CN). **SHEN, Lei** [CN/CN]; Room 301, No. 5, Lane 288, Wending Road, Xuhui District, Shanghai 200030 (CN). **LI, Yuanyuan** [CN/CN]; Intel No. 880 Zixing Road, Shanghai Zizhu Science Park, Shanghai 200241 (CN). **YANG, Yuting** [CN/US]; 832 Humewick Way, Sunnyvale, CA 94087 (US). **LUEH, Guei-Yuan** [US/US]; 1239 Quail Creek Circle, San Jose, CA 95120 (US).

(74) Agent: **SHANGHAI PATENT & TRADEMARK LAW OFFICE, LLC**; 435 Guiping Road, Shanghai 200233 (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, 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))*

(54) **Title**: GPU-CPU TWO-PATH MEMORY COPY

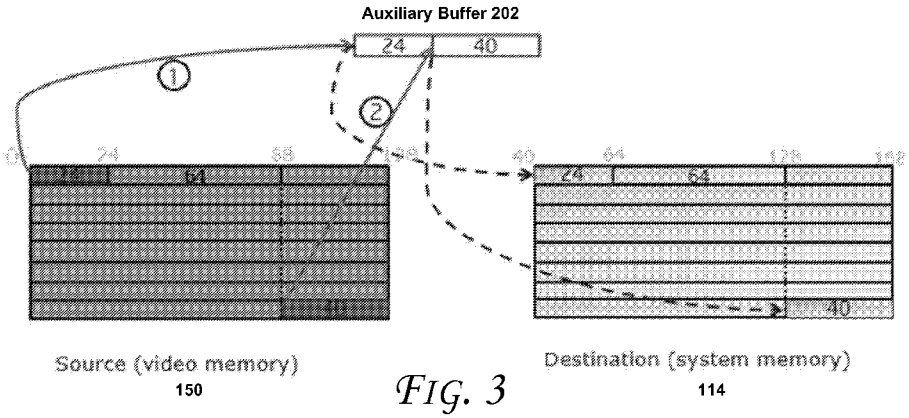


FIG. 3

(57) **Abstract**: Methods and apparatus relating to GPU-CPU (Graphics Processing Unit-Central Processing Unit) two-path memory copy are described. In an embodiment, a Graphics Processing Unit (GPU) copies at least a first portion of a data block from a source to a buffer. The GPU also copies a second portion of the data block from the source to a destination. A Central Processing Unit (CPU) copies the first portion of the data block from the first buffer to one or more corresponding locations in the destination. Other embodiments are also disclosed and claimed.

WO 2017/049583 A1

GPU-CPU TWO-PATH MEMORY COPY

FIELD

[0001] The present disclosure generally relates to the field of electronics. More particularly, an embodiment relates to GPU-CPU (Graphics Processing Unit-Central Processing Unit) two-path memory copy.

BACKGROUND

[0002] Some processors include both a CPU and a GPU. When processing image and/or video data, this data generally needs to be copied back and forth between a host memory and a video memory. These copy operations can be very expensive in terms of compute cycles and memory bandwidth usage.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The detailed description is provided with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different figures indicates similar or identical items.

[0004] Figs. 1, 6-7, 16, and 18 illustrate block diagrams of embodiments of computing systems, which may be utilized to implement various embodiments discussed herein.

[0005] Fig. 2 illustrates data flow associated with a pure CPU copy versus a hybrid copy in accordance with an embodiment.

[0006] Fig. 3 illustrates a block diagram of an implementation for a two-path memory copy operation, according to an embodiment.

[0007] Fig. 4 illustrates a flow diagram of a method to perform a two-path memory copy operation, according to an embodiment.

[0008] Fig. 5 illustrates a sample graph of throughput performance for the two-path memory copy achievable in accordance with an embodiment.

[0009] Figs. 8-12 and 14 illustrate various components of processers in accordance with some embodiments.

[0010] Fig. 13 illustrates graphics core instruction formats, according to some embodiments.

[0011] Figs. 15A and 15B illustrate graphics processor command format and sequence, respectively, according to some embodiments.

[0012] Fig. 17 illustrates a diagram of IP core development according to an embodiment.

DETAILED DESCRIPTION

[0013] In the following description, numerous specific details are set forth in order to provide a thorough understanding of various embodiments. However, various embodiments may be practiced without the specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as not to obscure the particular embodiments. Further, various aspects of embodiments may be performed using various means, such as integrated semiconductor circuits ("hardware"), computer-readable instructions organized into one or more programs ("software"), or some combination of hardware and software. For the purposes of this disclosure reference to "logic" shall mean either hardware, software, firmware, or some combination thereof.

[0014] Generally, image and/or video frame data needs to be copied from system or host memory (which may also be referred to as CPU memory) to video memory (which may be referred to as GPU memory) for GPU access. Once the processing finishes, the data will be copied back to system/host memory to perform the next processing operation(s) or, for example, be displayed on a screen. The copy-in (copy from memory associated with CPU to memory associated with GPU) and copy-out (from memory associated with GPU to memory associated with CPU) are expensive, in terms of compute cycles and/or memory bandwidth usage. And, sometimes the conversion of frames layout is essential with the copy operations, such as tiled to linear, linear to tiled, etc. The linear format is generally well suited for one-dimensional, row-sequential access patterns for system memory, where each row of the operand is stored in sequentially increasing memory locations.

Tiled format divides the enclosing region of image/video frames into an array of smaller rectangular regions, increasing performance for two-dimensional (2D) sub-region accesses for video memory. Thus, the overhead of the memory copy or transfer is one of the most common performance bottlenecks in image and/or video processing applications.

[0015] Moreover, there are generally three possible solutions to exchange data between GPU memory and CPU host memory in processors where both a CPU and a GPU are integrated on the same integrated circuit device, as follows: (1) a zero copy is the fastest solution but with relatively the largest number of restrictions; (2) a pure GPU copy is slower than the zero copy but it is still the fastest copy solution with fewer limitations; and (3) a pure CPU copy (e.g., using SSE/AVX or Streaming SIMD (Single Instruction, Multiple Data) Extensions/Advanced Vector Extensions) is the slowest copy solution with no limitation.

[0016] To this end, some embodiments provide a two-path memory copy technique for both GPU and CPU memory copy operations, leveraging both devices to transfer memory data. In an embodiment, most of data is copied by the GPU and the remaining portion of the data is copied by the CPU. This hybrid approach removes obstacles of a pure GPU copy, while reducing performance relatively very little. Moreover, compared with a pure CPU copy operation (e.g., using SSE/AVX), the disclosed hybrid approach is far faster.

[0017] Further, some embodiments may be applied in computing systems that include one or more processors (e.g., with one or more processor cores), such

as those discussed with reference to Figs. 1-9, including for example mobile computing devices, e.g., a smartphone, tablet, UMPC (Ultra-Mobile Personal Computer), laptop computer, Ultrabook™ computing device, smart watch, smart glasses, etc. More particularly, Fig. 1 illustrates a block diagram of a computing system 100, according to an embodiment. The system 100 may include one or more processors 102-1 through 102-N (generally referred to herein as "processors 102" or "processor 102"). The processors 102 may include general-purpose CPUs and/or GPUs in various embodiments. The processors 102 may communicate via an interconnection or bus 104. Each processor may include various components some of which are only discussed with reference to processor 102-1 for clarity. Accordingly, each of the remaining processors 102-2 through 102-N may include the same or similar components discussed with reference to the processor 102-1.

[0018] In an embodiment, the processor 102-1 may include one or more processor cores 106-1 through 106-M (referred to herein as "cores 106," or "core 106"), a cache 108, and/or a router 110. The processor cores 106 may be implemented on a single Integrated Circuit (IC) chip. Moreover, the chip may include one or more shared and/or private caches (such as cache 108), buses or interconnections (such as a bus or interconnection 112), graphics and/or memory controllers (such as those discussed with reference to Figs. 6-9), or other components.

[0019] In one embodiment, the router 110 may be used to communicate between various components of the processor 102-1 and/or system 100. Moreover, the processor 102-1 may include more than one router 110. Furthermore, the

multitude of routers 110 may be in communication to enable data routing between various components inside or outside of the processor 102-1.

[0020] The cache 108 may store data (e.g., including instructions) that are utilized by one or more components of the processor 102-1, such as the cores 106. For example, the cache 108 may locally cache data stored in a memory 114 for faster access by the components of the processor 102 (e.g., faster access by cores 106). As shown in Fig. 1, the memory 114 may communicate with the processors 102 via the interconnection 104. In an embodiment, the cache 108 (that may be shared) may be a mid-level cache (MLC), a last level cache (LLC), etc. Also, each of the cores 106 may include a level 1 (L1) cache (116-1) (generally referred to herein as "L1 cache 116") or other levels of cache such as a level 2 (L2) cache. Moreover, various components of the processor 102-1 may communicate with the cache 108 directly, through a bus (e.g., the bus 112), and/or a memory controller or hub.

[0021] As shown in Fig. 1, the processor 102 may further include graphics logic 140 (e.g., which may include one or more Graphics Processing Unit (GPU) core(s) such as those discussed with reference to Figs. 6-9) to perform various graphics and/or general purpose computation(s) related operations such as discussed herein. Logic 140 may have access to one or more storage devices discussed herein (such as video (or image, graphics, etc.) memory 150, cache 108, L1 cache 116, memory 114, register(s), or another memory in system 100) to store information relating to operations of the logic 140, such as information communicated with various components of system 100 as discussed herein. Also,

while logic 140 and video memory 150 are shown inside the processor 102 (or coupled to interconnection 104), they may be located elsewhere in the system 100 in various embodiments. For example, logic 140 may replace one of the cores 106, may be coupled directly to interconnection 112, etc. Also, video memory 150 may be directly coupled to interconnection 112, etc.

[0022] Fig. 2 illustrates data flow associated with a pure CPU copy versus a hybrid copy in accordance with an embodiment. Referring to Figs. 1-2, for a pure CPU copy operation (shown in the top portion of Fig. 2), the CPU locks the data (e.g., to avoid issues with multiple accesses to the same data) and then copies the locked data form video memory 150 to the system memory 114. In an embodiment, for a hybrid copy operation (shown in the bottom portion of Fig. 2), the GPU 140 copies at least a portion of data from the video memory 150 to some auxiliary/secondary buffer/storage 202. The CPU then copies the data from the storage 202 to the system memory 114 as will be further discussed with reference to Fig. 3.

[0023] Furthermore, as mentioned above, there are three possible solutions for transferring data from GPU to CPU as follows. A zero copy is the fastest method but with relatively many limitations such as: (a) the memory needs to be allocated in system memory space and mapped to video memory address space; (b) system memory storage layout needs to have a linear layout (which favors CPU access, but does not favor GPU access, and generally uses Y_tiled 2D surface storage format in video memory for faster GPU access); and (c) to map to GPU address space, the system memory needs to be (e.g., 4K bytes) page aligned (which

can be a hard restriction in many existing software products). A pure GPU copy is the fastest copy solution besides the zero copy solution, but it still has some restrictions (such as aligning the data in a certain way) which may prevent many developers from adopting it widely. A CPU copy operation has no restriction, but it is the slowest copy solution, e.g., due to the sub-line interleaving in Y_tiled surface storage format. The efficiency of memory read bandwidth is only 25% because it needs four cache-line read operations from video memory to build a full translated 64 Byte cache line in system memory.

[0024] Fig. 3 illustrates a block diagram of an MDF implementation for the two-path memory copy operation, according to an embodiment. As discussed herein, MDF (Media Development Framework) generally refers to a high-level programming framework to expose a GPU's general computing/processing capability, and it can greatly boost the performance for highly parallel or computing intensive tasks. Usually, an MDF application has two components: a kernel and a host program. The host program sets up and launches the kernel. The kernel in turn executes on GPU hardware.

[0025] In the use case for transfer of data from video memory 150 to system memory 114 with two-path copy, the destination is not 16 bytes-aligned system memory. Due to this unaligned destination memory address, GPU 140 cannot directly copy the first and the last several bytes to the destination in system memory 114. The reason is that the Oword (Octoword) block write command is used to write the data to destination, but it requires at least 16 bytes-aligned system memory and surface width. To address this limitation, an embodiment introduces a

(e.g., pre-allocated) auxiliary buffer 202 as a temporary buffer, which may be a user-provided memory buffer with page-aligned system memory. As discussed herein, auxiliary buffer 202 can be a zero-copied buffer, also called bufferUP.

[0026] Referring to Figs. 1-3, an embodiment utilizes a solution with the following two steps. First (labeled as circled 1 in Fig. 3), GPU part: MDF copy kernel: (a) GPU 140 copies the full cache lines (cache line width is 64 bytes in one embodiment) in the image from video memory 150 to (e.g., 16 bytes) aligned system memory to the destination (system memory 114). These full cache lines with 64 bytes-width represent the pixels in the image. For this part of GPU copy, an embodiment uses a media block read and Oword block write commands, and have a transpose to map tiled layout in video memory to linear layout in system memory; and (b) the GPU 140 copies those partial cache lines at the beginning and at end of row into the auxiliary memory 202 with media block read and Oword block write commands. This part of GPU copy performs the same functionality as described in (a) above since the same read/write commands would be sent to perform the data transfer. Second (labeled as circled 2 in Fig. 3), the CPU part includes: (a) after GPU finishes execution described above, the CPU accesses the data in auxiliary buffer 202 since it is zero-copied; and (b) the CPU copies this data to the corresponding places (i.e., the beginning and the end) in the destination system memory 114 as shown in Fig. 3.

[0027] Moreover, the two-path solution shown in Fig. 3 to use an auxiliary buffer 202 (e.g., user-provided memory buffer) to copy the unaligned data provides higher performance compared with the way that CPU copies them directly from

video memory to system memory shown in the top portion of Fig. 2. The reason is that the two-path copy operation uses the same fast GPU copy mechanism as pure GPU copy. Since the buffer 202 leverages zero copy, the data is already in system memory after being copied in by GPU. Then, the cost of copying these few data pieces (e.g., at the beginning and end of the data block) to the system memory is relatively very low. However, as discussed with reference to Fig. 2, a pure CPU copy requires the lock operation on surface to avoid resource confliction, which costs a lot in overhead. Refer to data transfer pipeline in figure 1.

[0028] Referring to Fig. 3, assuming that the start address of destination is 40 byte aligned (which is not 16 bytes aligned), the two-path copy solution is used to deal with the mis-alignment problem. In an embodiment, the kernel deals with 8*128 bytes in each thread as will be further discussed with reference to Fig. 4.

[0029] More particularly, Fig. 4 illustrates a flow diagram of a method 400 to perform a two-path memory copy operation, according to an embodiment. In one embodiment, method 400 shows operations performed by a software thread that copies a block of data from a source (e.g., video memory 150) to a destination (e.g., system memory 114) by utilizing both a CPU and a GPU. In an embodiment, various components discussed with reference to the other figures may be utilized to perform one or more of the operations of method 400.

[0030] Referring to Figs. 1-4, at operation 402, the beginning portion of a data block (e.g., provided in a row of memory or cache line, such as bits 0 to 24 video memory 150 shown in Fig. 3) is copied (e.g., by graphics logic/GPU 140) from a source (e.g., video memory 150) to an auxiliary buffer (e.g., buffer 202).

The beginning portion of the data block for operation 402 may be calculated (e.g., with a 64 bytes-aligned offset) based on the start address in system memory (for example, 64-40 or 24, as shown in the example of Fig. 3).

[0031] At operation 404, the remaining portion of the data block (i.e., after the beginning portion of operation 402, which could be the remaining full cache line (64 bytes)) is copied (e.g., by the graphics logic/GPU 140) from the source to an (e.g., aligned system memory) destination (e.g., system memory 114). The start address of aligned system memory for operation 404 may be calculated (e.g., with a 64 bytes-aligned roundup) based on the start address roundup (for example, 40, alignment or 64, as shown in the example of Fig. 3). Operation 404 repeats until the end of the data block is reached, as determined by operation 406 (e.g., hit the last 40 bytes at the end of the row, as shown in the example of Fig. 3).

[0032] At operation 408, the last portion of the data block (e.g., the last 40 bytes of the row as shown in the example of Fig. 3) is copied (e.g., by the graphics logic/GPU 140) from the source to the auxiliary buffer . At operation 410, the data stored in the auxiliary buffer is copied (e.g., by the CPU or one of the cores 106 of Fig. 1) to the correct place (e.g., beginning and end as shown in Fig. 3) in the destination. Method 400 may then be repeated for another copy thread/operation.

[0033] Fig. 5 illustrates a sample graph of throughput performance for the two-path memory copy achievable in accordance with an embodiment. As shown, the two-path copy provides a performance that is far better than a pure CPU copy (e.g., using SSE4) and only a bit less than a pure GPU copy. Hence, some embodiments that utilize the two-path memory copy achieve expected functionality

and good performance with both single plane surface and multi-plane surface. For the implementation of planar surface, the above case may be extended to cover both UV planes and Y plane. And, as shown in Fig. 5, the two-path solution is about 1.68x faster than a pure CPU copy (using SSE4) based on the illustrate sample test results, and without alignment restrictions. Moreover, the destination system memory address of two-path copy is 1 byte aligned in the test above, while pure GPU copy requires 16 bytes-aligned for destination system memory.

[0034] As discussed herein, YUV (also called YCbCr) is one of two primary color spaces used to represent digital component video (the other is RGB). The difference between YCbCr and RGB is that YCbCr represents color as brightness and two color difference signals, while RGB represents color as red, green and blue. In YCbCr, the Y is the brightness (luma), Cb is blue minus luma (B-Y) and Cr is red minus luma (R-Y). One of the YUV formats puts Y into one plane and puts UV into another plane.

[0035] As discussed above, some embodiments utilize both a CPU and a GPU to implement memory copy operation. Most of (e.g., aligned) data is copied by using a GPU copy kernel, and the rest of the (e.g., unaligned) data would be copied through a CPU copy operation. More particularly, one embodiment is implemented based on an MDF GPU programming framework. However, embodiments are not limited to an MDF framework and any runtime SDK (Software Development Kit) exposing the APIs (Application Programming Interfaces) which uses both a CPU and a GPU to perform memory data transfer may be used to implement various embodiments.

[0036] Accordingly, the GPU/CPU two-path copy embodiments discussed herein leverage both a CPU (for no restrictions) and a GPU (for high performance). Such solutions maintain the high performance of a pure GPU copy (with very little performance loss) and remove obstacles associated with a pure GPU copy operation. As a result, they also remove obstacles for users to use a GPU copy operation to accelerate memory transfers in their software products.

[0037] In some embodiments, one or more of the components discussed herein can be embodied as a System On Chip (SOC) device. Fig. 6 illustrates a block diagram of an SOC package in accordance with an embodiment. As illustrated in Fig. 6, SOC 602 includes one or more Central Processing Unit (CPU) cores 620 (which may be the same as or similar to the cores 106 of Fig. 1), one or more Graphics Processor Unit (GPU) cores 630 (which may be the same as or similar to the graphics logic 140 of Fig. 1), an Input/Output (I/O) interface 640, and a memory controller 642. Various components of the SOC package 602 may be coupled to an interconnect or bus such as discussed herein with reference to the other figures. Also, the SOC package 602 may include more or less components, such as those discussed herein with reference to the other figures. Further, each component of the SOC package 620 may include one or more other components, e.g., as discussed with reference to the other figures herein. In one embodiment, SOC package 602 (and its components) is provided on one or more Integrated Circuit (IC) die, e.g., which are packaged into a single semiconductor device.

[0038] As illustrated in Fig. 6, SOC package 602 is coupled to a memory 660 (which may be similar to or the same as memory discussed herein with

reference to the other figures such as system memory 114 of Fig. 1) via the memory controller 642. In an embodiment, the memory 660 (or a portion of it) can be integrated on the SOC package 602.

[0039] The I/O interface 640 may be coupled to one or more I/O devices 670, e.g., via an interconnect and/or bus such as discussed herein with reference to other figures. I/O device(s) 670 may include one or more of a keyboard, a mouse, a touchpad, a display, an image/video capture device (such as a camera or camcorder/video recorder), a touch screen, a speaker, or the like. Furthermore, SOC package 602 may include/integrate the logic 140 and/or video memory 150 (or a portion of the video memory 150) in an embodiment. Alternatively, the logic 140 and/or video memory 150 (or a portion of the video memory 150) may be provided outside of the SOC package 602 (i.e., as a discrete logic).

[0040] Fig. 7 is a block diagram of a processing system 700, according to an embodiment. In various embodiments the system 700 includes one or more processors 702 and one or more graphics processors 708 (such as the graphics logic 140 of Fig. 1), and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 702 (such as processor 102 of Fig. 1) or processor cores 707 (such as cores 106 of Fig. 1). In on embodiment, the system 700 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

[0041] An embodiment of system 700 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 700 is a mobile phone, smart phone, tablet computing device or mobile Internet device. Data processing system 700 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 700 is a television or set top box device having one or more processors 702 and a graphical interface generated by one or more graphics processors 708.

[0042] In some embodiments, the one or more processors 702 each include one or more processor cores 707 to process instructions which, when executed, perform operations for system and user software. In some embodiments, each of the one or more processor cores 707 is configured to process a specific instruction set 709. In some embodiments, instruction set 709 may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). Multiple processor cores 707 may each process a different instruction set 709, which may include instructions to facilitate the emulation of other instruction sets. Processor core 707 may also include other processing devices, such a Digital Signal Processor (DSP).

[0043] In some embodiments, the processor 702 includes cache memory 704. Depending on the architecture, the processor 702 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 702. In some embodiments, the processor 702 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 707 using known cache coherency techniques. A register file 706 is additionally included in processor 702 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 702.

[0044] In some embodiments, processor 702 is coupled to a processor bus 710 to transmit communication signals such as address, data, or control signals between processor 702 and other components in system 700. In one embodiment the system 700 uses an exemplary 'hub' system architecture, including a memory controller hub 716 and an Input Output (I/O) controller hub 730. A memory controller hub 716 facilitates communication between a memory device and other components of system 700, while an I/O Controller Hub (ICH) 730 provides connections to I/O devices via a local I/O bus. In one embodiment, the logic of the memory controller hub 716 is integrated within the processor.

[0045] Memory device 720 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 720 can operate as system memory for the system 700, to store data 722 and instructions 721 for use when the one or more processors 702 executes an application or process. Memory controller hub 716 also couples with an optional external graphics processor 712, which may communicate with the one or more

graphics processors 708 in processors 702 to perform graphics and media operations.

[0046] In some embodiments, ICH 730 enables peripherals to connect to memory device 720 and processor 702 via a high-speed I/O bus. The I/O peripherals include, but are not limited to, an audio controller 746, a firmware interface 728, a wireless transceiver 726 (e.g., Wi-Fi, Bluetooth), a data storage device 724 (e.g., hard disk drive, flash memory, etc.), and a legacy I/O controller 740 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to the system. One or more Universal Serial Bus (USB) controllers 742 connect input devices, such as keyboard and mouse 744 combinations. A network controller 734 may also couple to ICH 730. In some embodiments, a high-performance network controller (not shown) couples to processor bus 710. It will be appreciated that the system 700 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 730 may be integrated within the one or more processor 702, or the memory controller hub 716 and I/O controller hub 730 may be integrated into a discreet external graphics processor, such as the external graphics processor 712.

[0047] Fig. 8 is a block diagram of an embodiment of a processor 800 having one or more processor cores 802A-802N, an integrated memory controller 814, and an integrated graphics processor 808. The processor 800 may be similar to or the same as the processor 102 discussed with reference to Fig. 1. Those 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. Processor 800 can include additional cores up to and including additional core 802N represented by the dashed lined boxes. Each of processor cores 802A-802N includes one or more internal cache units 804A-804N. In some embodiments each processor core also has access to one or more shared cached units 806.

[0048] The internal cache units 804A-804N and shared cache units 806 represent a cache memory hierarchy within the processor 800. 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 806 and 804A-804N.

[0049] In some embodiments, processor 800 may also include a set of one or more bus controller units 816 and a system agent core 810. The one or more bus controller units 816 manage a set of peripheral buses, such as one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express). System agent core 810 provides management functionality for the various processor components. In some embodiments, system agent core 810 includes one or more integrated memory controllers 814 to manage access to various external memory devices (not shown).

[0050] In some embodiments, one or more of the processor cores 802A-802N include support for simultaneous multi-threading. In such embodiment, the

system agent core 810 includes components for coordinating and operating cores 802A-802N during multi-threaded processing. System agent core 810 may additionally include a power control unit (PCU), which includes logic and components to regulate the power state of processor cores 802A-802N and graphics processor 808.

[0051] In some embodiments, processor 800 additionally includes graphics processor 808 to execute graphics processing operations. In some embodiments, the graphics processor 808 couples with the set of shared cache units 806, and the system agent core 810, including the one or more integrated memory controllers 814. In some embodiments, a display controller 811 is coupled with the graphics processor 808 to drive graphics processor output to one or more coupled displays. In some embodiments, display controller 811 may be a separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 808 or system agent core 810.

[0052] In some embodiments, a ring based interconnect unit 812 is used to couple the internal components of the processor 800. 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 808 couples with the ring interconnect 812 via an I/O link 813.

[0053] The exemplary I/O link 813 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 818, such as an eDRAM (or embedded DRAM) module. In some embodiments, each of the processor cores 802-802N and graphics processor 808 use embedded memory modules 818 as a shared Last Level Cache.

[0054] In some embodiments, processor cores 802A-802N are homogenous cores executing the same instruction set architecture. In another embodiment, processor cores 802A-802N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 802A-802N execute a first instruction set, while at least one of the other cores executes a subset of the first instruction set or a different instruction set. In one embodiment processor cores 802A-802N 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 800 can be implemented on one or more chips or as an SoC integrated circuit having the illustrated components, in addition to other components.

[0055] Fig. 9 is a block diagram of a graphics processor 900, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. The graphics processor 900 may be similar to or the same as the graphics logic 140 discussed with reference to Fig. 1. 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 900 includes a memory interface 914 to access memory. Memory interface 914 can be an interface

to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0056] In some embodiments, graphics processor 900 also includes a display controller 902 to drive display output data to a display device 920. Display controller 902 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 900 includes a video codec engine 906 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.

[0057] In some embodiments, graphics processor 900 includes a block image transfer (BLIT) engine 904 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in one embodiment, 8D graphics operations are performed using one or more components of graphics processing engine (GPE) 910. In some embodiments, graphics processing engine 910 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0058] In some embodiments, GPE 910 includes a 3D pipeline 912 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 912 includes programmable and fixed function elements that perform various tasks within the element and/or spawn execution threads to a 3D/Media sub-system 915. While 3D pipeline 912 can be used to perform media operations, an embodiment of GPE 910 also includes a media pipeline 916 that is specifically used to perform media operations, such as video post-processing and image enhancement.

[0059] In some embodiments, media pipeline 916 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 906. In some embodiments, media pipeline 916 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 915. The spawned threads perform computations for the media operations on one or more graphics execution units included in 3D/Media sub-system 915.

[0060] In some embodiments, 3D/Media subsystem 915 includes logic for executing threads spawned by 3D pipeline 912 and media pipeline 916. In one embodiment, the pipelines send thread execution requests to 3D/Media subsystem 915, 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 915 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.

[0061] Fig. 10 is a block diagram of a graphics processing engine 1010 of a graphics processor in accordance with some embodiments. In one embodiment, the GPE 1010 is a version of the GPE 910 shown in Fig. 9. Elements of Fig. 10 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.

[0062] In some embodiments, GPE 1010 couples with a command streamer 1003, which provides a command stream to the GPE 3D and media pipelines 1012, 1016. In some embodiments, command streamer 1003 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 1003 receives commands from the memory and sends the commands to 3D pipeline 1012 and/or media pipeline 1016. The commands are directives fetched from a ring buffer, which stores commands for the 3D and media pipelines 1012, 1016. In one embodiment, the ring buffer can additionally include batch command buffers storing batches of multiple commands. The 3D and media pipelines 1012, 1016 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 1014. In some embodiments, execution unit array 1014 is scalable, such that the array includes a variable number of execution units based on the target power and performance level of GPE 1010.

[0063] In some embodiments, a sampling engine 1030 couples with memory (e.g., cache memory or system memory) and execution unit array 1014. In some embodiments, sampling engine 1030 provides a memory access mechanism for execution unit array 1014 that allows execution array 1014 to read graphics and media data from memory. In some embodiments, sampling engine 1030 includes logic to perform specialized image sampling operations for media.

[0064] In some embodiments, the specialized media sampling logic in sampling engine 1030 includes a de-noise/de-interlace module 1032, a motion estimation module 1034, and an image scaling and filtering module 1036. In some embodiments, de-noise/de-interlace module 1032 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 fame 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 1032 includes dedicated motion detection logic (e.g., within the motion estimation engine 1034).

[0065] In some embodiments, motion estimation engine 1034 provides hardware acceleration for video operations by performing video acceleration functions such as motion vector estimation and prediction on video data. The motion estimation engine determines motion vectors that describe the transformation of image data between successive video frames. In some

embodiments, a graphics processor media codec uses video motion estimation engine 1034 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 1034 is generally available to graphics processor components to assist with video decode and processing functions that are sensitive or adaptive to the direction or magnitude of the motion within video data.

[0066] In some embodiments, image scaling and filtering module 1036 performs image-processing operations to enhance the visual quality of generated images and video. In some embodiments, scaling and filtering module 1036 processes image and video data during the sampling operation before providing the data to execution unit array 1014.

[0067] In some embodiments, the GPE 1010 includes a data port 1044, which provides an additional mechanism for graphics subsystems to access memory. In some embodiments, data port 1044 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 1044 includes cache memory space to cache accesses to memory. The cache memory can be a single data cache or separated into multiple caches for the multiple subsystems that access memory via the data port (e.g., a render buffer cache, a constant buffer cache, etc.). In some embodiments, threads executing on an execution unit in execution unit array 1014 communicate with the data port by

exchanging messages via a data distribution interconnect that couples each of the sub-systems of GPE 1010.

[0068] Fig. 11 is a block diagram of another embodiment of a graphics processor 1100. Elements of Fig. 11 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.

[0069] In some embodiments, graphics processor 1100 includes a ring interconnect 1102, a pipeline front-end 1104, a media engine 1137, and graphics cores 1180A-1180N. In some embodiments, ring interconnect 1102 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.

[0070] In some embodiments, graphics processor 1100 receives batches of commands via ring interconnect 1102. The incoming commands are interpreted by a command streamer 1103 in the pipeline front-end 1104. In some embodiments, graphics processor 1100 includes scalable execution logic to perform 3D geometry processing and media processing via the graphics core(s) 1180A-1180N. For 3D geometry processing commands, command streamer 1103 supplies commands to geometry pipeline 1136. For at least some media processing commands, command streamer 1103 supplies the commands to a video front end 1134, which couples with a media engine 1137. In some embodiments, media engine 1137 includes a Video Quality Engine (VQE) 1130 for video and image post-processing and a

multi-format encode/decode (MFX) 1133 engine to provide hardware-accelerated media data encode and decode. In some embodiments, geometry pipeline 1136 and media engine 1137 each generate execution threads for the thread execution resources provided by at least one graphics core 1180A.

[0071] In some embodiments, graphics processor 1100 includes scalable thread execution resources featuring modular cores 1180A-1180N (sometimes referred to as core slices), each having multiple sub-cores 1150A-1150N, 1160A-1160N (sometimes referred to as core sub-slices). In some embodiments, graphics processor 1100 can have any number of graphics cores 1180A through 1180N. In some embodiments, graphics processor 1100 includes a graphics core 1180A having at least a first sub-core 1150A and a second core sub-core 1160A. In other embodiments, the graphics processor is a low power processor with a single sub-core (e.g., 1150A). In some embodiments, graphics processor 1100 includes multiple graphics cores 1180A-1180N, each including a set of first sub-cores 1150A-1150N and a set of second sub-cores 1160A-1160N. Each sub-core in the set of first sub-cores 1150A-1150N includes at least a first set of execution units 1152A-1152N and media/texture samplers 1154A-1154N. Each sub-core in the set of second sub-cores 1160A-1160N includes at least a second set of execution units 1162A-1162N and samplers 1164A-1164N. In some embodiments, each sub-core 1150A-1150N, 1160A-1160N shares a set of shared resources 1170A-1170N. 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.

[0072] Fig. 12 illustrates thread execution logic 1200 including an array of processing elements employed in some embodiments of a GPE. Elements of Fig. 12 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.

[0073] In some embodiments, thread execution logic 1200 includes a pixel shader 1202, a thread dispatcher 1204, instruction cache 1206, a scalable execution unit array including a plurality of execution units 1208A-1208N, a sampler 1210, a data cache 1212, and a data port 1214. 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 1200 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 1206, data port 1214, sampler 1210, and execution unit array 1208A-1208N. In some embodiments, each execution unit (e.g. 1208A) 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 1208A-1208N includes any number individual execution units.

[0074] In some embodiments, execution unit array 1208A-1208N is primarily used to execute "shader" programs. In some embodiments, the execution units in array 1208A-1208N 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).

[0075] Each execution unit in execution unit array 1208A-1208N 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 1208A-1208N support integer and floating-point data types.

[0076] 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.

[0077] One or more internal instruction caches (e.g., 1206) are included in the thread execution logic 1200 to cache thread instructions for the execution units. In some embodiments, one or more data caches (e.g., 1212) are included to cache thread data during thread execution. In some embodiments, sampler 1210 is included to provide texture sampling for 3D operations and media sampling for media operations. In some embodiments, sampler 1210 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.

[0078] During execution, the graphics and media pipelines send thread initiation requests to thread execution logic 1200 via thread spawning and dispatch logic. In some embodiments, thread execution logic 1200 includes a local thread dispatcher 1204 that arbitrates thread initiation requests from the graphics and media pipelines and instantiates the requested threads on one or more execution units 1208A-1208N. For example, the geometry pipeline (e.g., 1136 of Fig. 11) dispatches vertex processing, tessellation, or geometry processing threads to thread execution logic 1200 (Fig. 12). In some embodiments, thread dispatcher 1204 can also process runtime thread spawning requests from the executing shader programs.

[0079] Once a group of geometric objects has been processed and rasterized into pixel data, pixel shader 1202 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 1202 calculates the values of the various vertex attributes that are to be interpolated across the rasterized object. In some embodiments, pixel shader 1202 then executes an application

programming interface (API)-supplied pixel shader program. To execute the pixel shader program, pixel shader 1202 dispatches threads to an execution unit (e.g., 1208A) via thread dispatcher 1204. In some embodiments, pixel shader 1202 uses texture sampling logic in sampler 1210 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.

[0080] In some embodiments, the data port 1214 provides a memory access mechanism for the thread execution logic 1200 output processed data to memory for processing on a graphics processor output pipeline. In some embodiments, the data port 1214 includes or couples to one or more cache memories (e.g., data cache 1212) to cache data for memory access via the data port.

[0081] Fig. 13 is a block diagram illustrating a graphics processor instruction formats 1300 according to some embodiments. In one or more embodiment, the graphics processor execution units support an instruction set having instructions in multiple formats. The solid lined boxes illustrate the components that are generally included in an execution unit instruction, while the dashed lines include components that are optional or that are only included in a sub-set of the instructions. In some embodiments, instruction format 1300 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.

[0082] In some embodiments, the graphics processor execution units natively support instructions in a 128-bit format 1310. A 64-bit compacted instruction format 1330 is available for some instructions based on the selected instruction, instruction options, and number of operands. The native 128-bit format 1310 provides access to all instruction options, while some options and operations are restricted in the 64-bit format 1330. The native instructions available in the 64-bit format 1330 vary by embodiment. In some embodiments, the instruction is compacted in part using a set of index values in an index field 1313. The execution unit hardware references a set of compaction tables based on the index values and uses the compaction table outputs to reconstruct a native instruction in the 128-bit format 1310.

[0083] For each format, instruction opcode 1312 defines the operation that the execution unit is to perform. The execution units execute each instruction in parallel across the multiple data elements of each operand. For example, in response to an add instruction the execution unit performs a simultaneous add operation across each color channel representing a texture element or picture element. By default, the execution unit performs each instruction across all data channels of the operands. In some embodiments, instruction control field 1314 enables control over certain execution options, such as channels selection (e.g., predication) and data channel order (e.g., swizzle). For 128-bit instructions 1310 an exec-size field 1316 limits the number of data channels that will be executed in parallel. In some embodiments, exec-size field 1316 is not available for use in the 64-bit compact instruction format 1330.

[0084] Some execution unit instructions have up to three operands including two source operands, src0 1322, src1 1322, and one destination 1318. 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 1324), where the instruction opcode 1312 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.

[0085] In some embodiments, the 128-bit instruction format 1310 includes an access/address mode information 1326 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 1310.

[0086] In some embodiments, the 128-bit instruction format 1310 includes an access/address mode field 1326, 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 1310 may use byte-aligned addressing for source and destination operands and when in a second mode, the instruction 1310 may use 16-byte-aligned addressing for all source and destination operands.

[0087] In one embodiment, the address mode portion of the access/address mode field 1326 determines whether the instruction is to use direct or indirect addressing. When direct register addressing mode is used bits in the instruction 1310 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.

[0088] In some embodiments instructions are grouped based on opcode 1312 bit-fields to simplify Opcode decode 1340. For an 8-bit opcode, bits 10, 11, and 12 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 1342 includes data movement and logic instructions (e.g., move (mov), compare (cmp)). In some embodiments, move and logic group 1342 shares the five most significant bits (MSB), where move (mov) instructions are in the form of 0000xxxxb and logic instructions are in the form of 0001xxxxb. A flow control instruction group 1344 (e.g., call, jump (jmp)) includes instructions in the form of 0010xxxxb (e.g., 0x20). A miscellaneous instruction group 1346 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 1348 includes component-wise arithmetic instructions (e.g., add, multiply (mul)) in the form of 0100xxxxb (e.g., 0x40). The parallel math group 1348 performs the arithmetic operations in parallel across data channels. The vector math group 1350 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.

[0089] Fig. 14 is a block diagram of another embodiment of a graphics processor 1400. Elements of Fig. 14 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.

[0090] In some embodiments, graphics processor 1400 includes a graphics pipeline 1420, a media pipeline 1430, a display engine 1440, thread execution logic 1450, and a render output pipeline 1470. In some embodiments, graphics processor 1400 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 1400 via a ring interconnect 1402. In some embodiments, ring interconnect 1402 couples graphics processor 1400 to other processing components, such as other graphics processors or general-purpose processors. Commands from ring interconnect 1402 are interpreted by a command streamer 1403, which supplies instructions to individual components of graphics pipeline 1420 or media pipeline 1430.

[0091] In some embodiments, command streamer 1403 directs the operation of a vertex fetcher 1405 that reads vertex data from memory and executes vertex-processing commands provided by command streamer 1403. In some embodiments, vertex fetcher 1405 provides vertex data to a vertex shader 1407, which performs coordinate space transformation and lighting operations to each vertex. In some

embodiments, vertex fetcher 1405 and vertex shader 1407 execute vertex-processing instructions by dispatching execution threads to execution units 1452A, 1452B via a thread dispatcher 1431.

[0092] In some embodiments, execution units 1452A, 1452B are an array of vector processors having an instruction set for performing graphics and media operations. In some embodiments, execution units 1452A, 1452B have an attached L1 cache 1451 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.

[0093] In some embodiments, graphics pipeline 1420 includes tessellation components to perform hardware-accelerated tessellation of 3D objects. In some embodiments, a programmable hull shader 1411 configures the tessellation operations. A programmable domain shader 1417 provides back-end evaluation of tessellation output. A tessellator 1413 operates at the direction of hull shader 1411 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 1420. In some embodiments, if tessellation is not used, tessellation components 1411, 1413, 1417 can be bypassed.

[0094] In some embodiments, complete geometric objects can be processed by a geometry shader 1419 via one or more threads dispatched to execution units 1452A, 1452B, or can proceed directly to the clipper 1429. 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 1419 receives input from the vertex shader 1407. In some embodiments, geometry shader 1419 is programmable by a geometry shader program to perform geometry tessellation if the tessellation units are disabled.

[0095] Before rasterization, a clipper 1429 processes vertex data. The clipper 1429 may be a fixed function clipper or a programmable clipper having clipping and geometry shader functions. In some embodiments, a rasterizer/depth 1473 in the render output pipeline 1470 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 1450. In some embodiments, an application can bypass the rasterizer 1473 and access un-rasterized vertex data via a stream out unit 1423.

[0096] The graphics processor 1400 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 1452A, 1452B and associated cache(s) 1451, texture and media sampler 1454, and texture/sampler cache 1458 interconnect via a data port 1456 to perform memory access and communicate with render output pipeline components of the processor. In some embodiments, sampler 1454, caches 1451, 1458 and execution units 1452A, 1452B each have separate memory access paths.

[0097] In some embodiments, render output pipeline 1470 contains a rasterizer and depth test component 1473 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 1478 and depth cache 1479 are also available in some embodiments. A pixel operations component 1477 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 1441, or substituted at display time by the display controller 1443 using overlay display planes. In some embodiments, a shared L3 cache 1475 is available to all graphics components, allowing the sharing of data without the use of main system memory.

[0098] In some embodiments, graphics processor media pipeline 1430 includes a media engine 1437 and a video front end 1434. In some embodiments, video front end 1434 receives pipeline commands from the command streamer 1403. In some embodiments, media pipeline 1430 includes a separate command streamer. In some embodiments, video front-end 1434 processes media commands before sending the command to the media engine 1437. In some embodiments, media engine 1437 includes thread spawning functionality to spawn threads for dispatch to thread execution logic 1450 via thread dispatcher 1431.

[0099] In some embodiments, graphics processor 1400 includes a display engine 1440. In some embodiments, display engine 1440 is external to processor 1400 and couples with the graphics processor via the ring interconnect 1402, or some other interconnect bus or fabric. In some embodiments, display engine 1440 includes a 2D engine 1441 and a display controller 1443. In some embodiments, display engine 1440 contains special purpose logic capable of operating

independently of the 3D pipeline. In some embodiments, display controller 1443 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.

[00100] In some embodiments, graphics pipeline 1420 and media pipeline 1430 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.

[00101] Fig. 15A is a block diagram illustrating a graphics processor command format 1500 according to some embodiments. Fig. 15B is a block diagram illustrating a graphics processor command sequence 1510 according to an embodiment. The solid lined boxes in Fig. 15A 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 1500 of Fig. 15A includes data fields to identify a target client 1502 of the command, a command operation code (opcode) 1504, and the relevant data 1506 for the command. A sub-opcode 1505 and a command size 1508 are also included in some commands.

[00102] In some embodiments, client 1502 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 1504 and, if present, sub-opcode 1505 to determine the operation to perform. The client unit performs the command using information in data field 1506. For some commands an explicit command size 1508 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.

[00103] The flow diagram in Fig. 15B shows an exemplary graphics processor command sequence 1510. 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.

[00104] In some embodiments, the graphics processor command sequence 1510 may begin with a pipeline flush command 1512 to cause any active graphics pipeline to complete the currently pending commands for the pipeline. In some embodiments, the 3D pipeline 1522 and the media pipeline 1524 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 1512 can be used for pipeline synchronization or before placing the graphics processor into a low power state.

[00105] In some embodiments, a pipeline select command 1513 is used when a command sequence requires the graphics processor to explicitly switch between pipelines. In some embodiments, a pipeline select command 1513 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 1512 is required immediately before a pipeline switch via the pipeline select command 1513.

[00106] In some embodiments, a pipeline control command 1514 configures a graphics pipeline for operation and is used to program the 3D pipeline 1522 and the media pipeline 1524. In some embodiments, pipeline control command 1514 configures the pipeline state for the active pipeline. In one embodiment, the pipeline control command 1514 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.

[00107] In some embodiments, return buffer state commands 1516 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 1516 includes selecting the size and number of return buffers to use for a set of pipeline operations.

[00108] The remaining commands in the command sequence differ based on the active pipeline for operations. Based on a pipeline determination 1520, the command sequence is tailored to the 3D pipeline 1522 beginning with the 3D pipeline state 1530, or the media pipeline 1524 beginning at the media pipeline state 1540.

[00109] The commands for the 3D pipeline state 1530 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 1530 commands are also able to selectively disable or bypass certain pipeline elements if those elements will not be used.

[00110] In some embodiments, 3D primitive 1532 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 1532 command are forwarded to the vertex fetch function in the graphics pipeline. The vertex fetch function uses the 3D primitive 1532 command data to generate vertex data structures. The vertex data structures are stored in one or more return buffers. In some embodiments, 3D primitive 1532 command is used to perform vertex operations on 3D primitives via vertex shaders. To process vertex shaders, 3D pipeline 1522 dispatches shader execution threads to graphics processor execution units.

[00111] In some embodiments, 3D pipeline 1522 is triggered via an execute 1534 command or event. In some embodiments, a register write triggers command execution. In some embodiments execution is triggered via a 'go' or 'kick' command in the command sequence. In one embodiment command execution is triggered using a pipeline synchronization command to flush the command sequence through the graphics pipeline. The 3D pipeline will perform geometry

processing for the 3D primitives. Once operations are complete, the resulting geometric objects are rasterized and the pixel engine colors the resulting pixels. Additional commands to control pixel shading and pixel back end operations may also be included for those operations.

[00112] In some embodiments, the graphics processor command sequence 1510 follows the media pipeline 1524 path when performing media operations. In general, the specific use and manner of programming for the media pipeline 1524 depends on the media or compute operations to be performed. Specific media decode operations may be offloaded to the media pipeline during media decode. In some embodiments, the media pipeline can also be bypassed and media decode can be performed in whole or in part using resources provided by one or more general purpose processing cores. In one embodiment, the media pipeline also includes elements for general-purpose graphics processor unit (GPGPU) operations, where the graphics processor is used to perform SIMD vector operations using computational shader programs that are not explicitly related to the rendering of graphics primitives.

[00113] In some embodiments, media pipeline 1524 is configured in a similar manner as the 3D pipeline 1522. A set of media pipeline state commands 1540 are dispatched or placed into in a command queue before the media object commands 1542. In some embodiments, media pipeline state commands 1540 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 1540 also support the use one or more pointers to "indirect" state elements that contain a batch of state settings.

[00114] In some embodiments, media object commands 1542 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 1542. Once the pipeline state is configured and media object commands 1542 are queued, the media pipeline 1524 is triggered via an execute command 1544 or an equivalent execute event (e.g., register write). Output from media pipeline 1524 may then be post processed by operations provided by the 3D pipeline 1522 or the media pipeline 1524. In some embodiments, GPGPU operations are configured and executed in a similar manner as media operations.

[00115] Fig. 16 illustrates exemplary graphics software architecture for a data processing system 1600 according to some embodiments. In some embodiments, software architecture includes a 3D graphics application 1610, an operating system 1620, and at least one processor 1630. In some embodiments, processor 1630 includes a graphics processor 1632 and one or more general-purpose processor core(s) 1634. The graphics application 1610 and operating system 1620 each execute in the system memory 1650 of the data processing system.

[00116] In some embodiments, 3D graphics application 1610 contains one or more shader programs including shader instructions 1612. 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 1614 in a machine language suitable for execution by the general-purpose processor core 1634. The application also includes graphics objects 1616 defined by vertex data.

[00117] In some embodiments, operating system 1620 is a Microsoft® Windows® operating system from the Microsoft Corporation, a proprietary UNIX-like operating system, or an open source UNIX-like operating system using a variant of the Linux kernel. When the Direct3D API is in use, the operating system 1620 uses a front-end shader compiler 1624 to compile any shader instructions 1612 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 1610.

[00118] In some embodiments, user mode graphics driver 1626 contains a back-end shader compiler 1627 to convert the shader instructions 1612 into a hardware specific representation. When the OpenGL API is in use, shader instructions 1612 in the GLSL high-level language are passed to a user mode graphics driver 1626 for compilation. In some embodiments, user mode graphics driver 1626 uses operating system kernel mode functions 1628 to communicate with a kernel mode graphics driver 1629. In some embodiments, kernel mode graphics driver 1629 communicates with graphics processor 1632 to dispatch commands and instructions.

[00119] One or more aspects of at least one embodiment may be implemented by representative code stored on a machine-readable medium which represents and/or defines logic within an integrated circuit such as a processor. For example, the machine-readable medium may include instructions which represent various logic within the processor. When read by a machine, the instructions may cause the machine to fabricate the logic to perform the techniques described herein. Such representations, known as "IP cores," are reusable units of logic for an integrated circuit that may be stored on a tangible, machine-readable medium as a hardware model that describes the structure of the integrated circuit. The hardware model may be supplied to various customers or manufacturing facilities, which load the hardware model on fabrication machines that manufacture the integrated circuit. The integrated circuit may be fabricated such that the circuit performs operations described in association with any of the embodiments described herein.

[00120] Fig. 17 is a block diagram illustrating an IP core development system 1700 that may be used to manufacture an integrated circuit to perform operations according to an embodiment. The IP core development system 1700 may be used to generate modular, re-usable designs that can be incorporated into a larger design or used to construct an entire integrated circuit (e.g., an SOC integrated circuit). A design facility 1730 can generate a software simulation 1710 of an IP core design in a high level programming language (e.g., C/C++). The software simulation 1710 can be used to design, test, and verify the behavior of the IP core. A register transfer level (RTL) design can then be created or synthesized from the simulation model 1700. The RTL design 1715 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 1715, 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.

[00121] The RTL design 1715 or equivalent may be further synthesized by the design facility into a hardware model 1720, 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 1765 using non-volatile memory 1740 (e.g., hard disk, flash memory, or any non-volatile storage medium). Alternatively, the IP core design may be transmitted (e.g., via the Internet) over a wired connection 1750 or wireless connection 1760. The fabrication facility 1765 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.

[00122] Fig. 18 is a block diagram illustrating an exemplary system on a chip integrated circuit 1800 that may be fabricated using one or more IP cores, according to an embodiment. The exemplary integrated circuit includes one or more application processors 1805 (e.g., CPUs), at least one graphics processor 1810, and may additionally include an image processor 1815 and/or a video processor 1820, any of which may be a modular IP core from the same or multiple different design facilities. The integrated circuit includes peripheral or bus logic

including a USB controller 1825, UART controller 1830, an SPI/SDIO controller 1835, and an I2S/I2C controller 1840. Additionally, the integrated circuit can include a display device 1845 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1850 and a mobile industry processor interface (MIPI) display interface 1855. Storage may be provided by a flash memory subsystem 1860 including flash memory and a flash memory controller. Memory interface may be provided via a memory controller 1865 for access to SDRAM or SRAM memory devices. Some integrated circuits additionally include an embedded security engine 1870.

[00123] Additionally, other logic and circuits may be included in the processor of integrated circuit 1800, including additional graphics processors/cores, peripheral interface controllers, or general purpose processor cores.

[00124] The following examples pertain to further embodiments. Example 1 includes an apparatus comprising: a Graphics Processing Unit (GPU) to copy at least a first portion of a data block from a source to a buffer and to copy a second portion of the data block from the source to a destination; and a Central Processing Unit (CPU) to copy the first portion of the data block from the buffer to one or more corresponding locations in the destination. Example 2 includes the apparatus of example 1, wherein the first portion of the data block is to comprise first data at the start of the data block or second data at the end of the data block. Example 3 includes the apparatus of example 2, wherein the first data and the second data are to each have a size less than a memory unit. Example 4 includes the apparatus of example 3, wherein the memory unit is to comprise a cache line or 64 bytes.

Example 5 includes the apparatus of example 1, wherein the second portion of the data block is to comprise one or more complete memory units. Example 6 includes the apparatus of example 5, wherein each of the one or more complete memory units is to comprise 64 bytes. Example 7 includes the apparatus of example 1, wherein the source is to comprise a video memory coupled to the GPU. Example 8 includes the apparatus of example 1, wherein the destination is to comprise a system memory coupled to the CPU. Example 9 includes the apparatus of example 1, wherein the GPU is to comprise one or more graphics processing cores. Example 10 includes the apparatus of example 1, wherein the CPU is to comprise one or more processor cores. Example 11 includes the apparatus of example 1, wherein one or more of the GPU, the CPU, or memory are on a single integrated circuit die.

[00125] Example 12 includes a system comprising: a processor, coupled to system memory, the system memory to store a data block to be copied from a video memory, the processor to comprise: a Graphics Processing Unit (GPU) to copy: a first portion of the data block from the video memory to a first buffer; a second portion of the data block from the video memory to the system memory; and a third portion of the data block from the video memory to a second buffer; and a Central Processing Unit (CPU) to copy: the first portion of the data block from the first buffer to a corresponding location in the system memory; and the second portion of the data block from the second buffer to a corresponding location in the system memory. Example 13 includes the system of example 12, wherein the data block is to contiguously consist of the first portion, the second portion, and the third portion in order. Example 14 includes the system of example 12, wherein the first portion and the third portion of the data block are to each comprise less bytes

than a memory unit. Example 15 includes the system of example 14, wherein the memory unit is to comprise a cache line or 64 bytes. Example 16 includes the system of example 12, wherein the second portion of the data block is to comprise one or more complete memory units. Example 17 includes the system of example 16, wherein each of the one or more complete memory units is to comprise 64 bytes. Example 18 includes the system of example 12, wherein the first buffer and the second buffer are to be combined into a single buffer. Example 19 includes the system of example 12, wherein one or more of the GPU having one or more graphics processing cores, the CPU having one or more processor cores, at least a portion of the system memory, or at least a portion of the video memory are on a single integrated circuit die.

[00126] Example 20 includes a computer-readable medium comprising one or more instructions that when executed on a processor configure the processor to perform one or more operations to: cause a Graphics Processing Unit (GPU) to copy at least a first portion of a data block from a source to a buffer and to copy a second portion of the data block from the source to a destination; and cause a Central Processing Unit (CPU) to copy the first portion of the data block from the first buffer to one or more corresponding locations in the destination. Example 21 includes the computer-readable medium of example 20, wherein the first portion of the data block is to comprise first data at the start of the data block or second data at the end of the data block. Example 22 includes the computer-readable medium of example 21, wherein the first data and the second data are to each have a size less than a memory unit. Example 23 includes the computer-readable medium of example 22, wherein the memory unit is to comprise a cache line or 64 bytes.

Example 24 includes the computer-readable medium of example 20, wherein the second portion of the data block is to comprise one or more complete memory units. Example 25 includes the computer-readable medium of example 20, wherein each of the one or more complete memory units is to comprise 64 bytes.

[00127] Example 26 includes a method comprising: causing a Graphics Processing Unit (GPU) to copy at least a first portion of a data block from a source to a buffer and to copy a second portion of the data block from the source to a destination; and causing a Central Processing Unit (CPU) to copy the first portion of the data block from the first buffer to one or more corresponding locations in the destination. Example 27 includes the method example 26, wherein the first portion of the data block comprises first data at the start of the data block or second data at the end of the data block, or wherein the second portion of the data block is to comprise one or more complete memory units, or wherein each of the one or more complete memory units is to comprise 64 bytes. Example 28 includes the method of example 27, wherein the first data and the second data each have a size less than a memory unit. Example 29 includes the method of example 28, wherein the memory unit comprises a cache line or 64 bytes.

[00128] Example 30 includes an apparatus comprising means to perform a method as set forth in any preceding example. Example 31 comprises machine-readable storage including machine-readable instructions, when executed, to implement a method or realize an apparatus as set forth in any preceding example.

[00129] In various embodiments, the operations discussed herein, e.g., with reference to Figs. 1-18, may be implemented as hardware (e.g., logic circuitry),

software, firmware, or combinations thereof, which may be provided as a computer program product, e.g., including a tangible (e.g., non-transitory) machine-readable or computer-readable medium having stored thereon instructions (or software procedures) used to program a computer to perform a process discussed herein. The machine-readable medium may include a storage device such as those discussed with respect to Figs. 1-18.

[00130] Additionally, such computer-readable media 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 data signals provided in a carrier wave or other propagation medium via a communication link (e.g., a bus, a modem, or a network connection).

[00131] Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, and/or characteristic described in connection with the embodiment may be included in at least an implementation. The appearances of the phrase "in one embodiment" in various places in the specification may or may not be all referring to the same embodiment.

[00132] Also, in the description and claims, the terms "coupled" and "connected," along with their derivatives, may be used. In some embodiments, "connected" may be used to indicate that two or more elements are in direct physical or electrical contact with each other. "Coupled" may mean that two or more elements are in direct physical or electrical contact. However, "coupled" may also mean that two or more elements may not be in direct contact with each other, but may still cooperate or interact with each other.

[00133] Thus, although embodiments have been described in language specific to structural features and/or methodological acts, it is to be understood that claimed subject matter may not be limited to the specific features or acts described. Rather, the specific features and acts are disclosed as sample forms of implementing the claimed subject matter.

CLAIMS

1. An apparatus comprising:

a Graphics Processing Unit (GPU) to copy at least a first portion of a data block from a source to a buffer and to copy a second portion of the data block from the source to a destination; and

a Central Processing Unit (CPU) to copy the first portion of the data block from the buffer to one or more corresponding locations in the destination.

2. The apparatus of claim 1, wherein the first portion of the data block is to comprise first data at the start of the data block or second data at the end of the data block.

3. The apparatus of claim 2, wherein the first data and the second data are to each have a size less than a memory unit.

4. The apparatus of claim 3, wherein the memory unit is to comprise a cache line or 64 bytes.

5. The apparatus of claim 1, wherein the second portion of the data block is to comprise one or more complete memory units.

6. The apparatus of claim 5, wherein each of the one or more complete memory units is to comprise 64 bytes.

7. The apparatus of claim 1, wherein the source is to comprise a video memory coupled to the GPU.

8. The apparatus of claim 1, wherein the destination is to comprise a system memory coupled to the CPU.

9. The apparatus of claim 1, wherein the GPU is to comprise one or more graphics processing cores.

10. The apparatus of claim 1, wherein the CPU is to comprise one or more processor cores.

11. The apparatus of claim 1, wherein one or more of the GPU, the CPU, or memory are on a single integrated circuit die.

12. A system comprising:

a processor, coupled to system memory, the system memory to store a data block to be copied from a video memory, the processor to comprise:

a Graphics Processing Unit (GPU) to copy:

a first portion of the data block from the video memory to a first buffer;

a second portion of the data block from the video memory to the system memory; and

a third portion of the data block from the video memory to a second buffer; and

a Central Processing Unit (CPU) to copy:

the first portion of the data block from the first buffer to a corresponding location in the system memory; and

the second portion of the data block from the second buffer to a corresponding location in the system memory.

13. The system of claim 12, wherein the data block is to contiguously consist of the first portion, the second portion, and the third portion in order.

14. The system of claim 12, wherein the first portion and the third portion of the data block are to each comprise less bytes than a memory unit.

15. The system of claim 14, wherein the memory unit is to comprise a cache line or 64 bytes.

16. The system of claim 12, wherein the second portion of the data block is to comprise one or more complete memory units.

17. The system of claim 16, wherein each of the one or more complete memory units is to comprise 64 bytes.

18. The system of claim 12, wherein the first buffer and the second buffer are to be combined into a single buffer.

19. The system of claim 12, wherein one or more of the GPU having one or more graphics processing cores, the CPU having one or more processor cores, at least a portion of the system memory, or at least a portion of the video memory are on a single integrated circuit die.

20. A method comprising:

causing a Graphics Processing Unit (GPU) to copy at least a first portion

of a data block from a source to a buffer and to copy a second portion of the data block from the source to a destination; and

causing a Central Processing Unit (CPU) to copy the first portion of the data block from the first buffer to one or more corresponding locations in the destination.

21. The method claim 20, wherein the first portion of the data block comprises first data at the start of the data block or second data at the end of the data block, or wherein the second portion of the data block is to comprise one or more complete memory units, or wherein each of the one or more complete memory units is to comprise 64 bytes.

22. The method of claim 21, wherein the first data and the second data each have a size less than a memory unit.

23. The method of claim 22, wherein the memory unit comprises a cache line or 64 bytes.

24. A computer-readable medium comprising one or more instructions that when executed on a processor configure the processor to perform one or more operations of any one of claims 20 to 23.

25. An apparatus comprising means to perform a method as set forth in any one of claims 20 to 23.

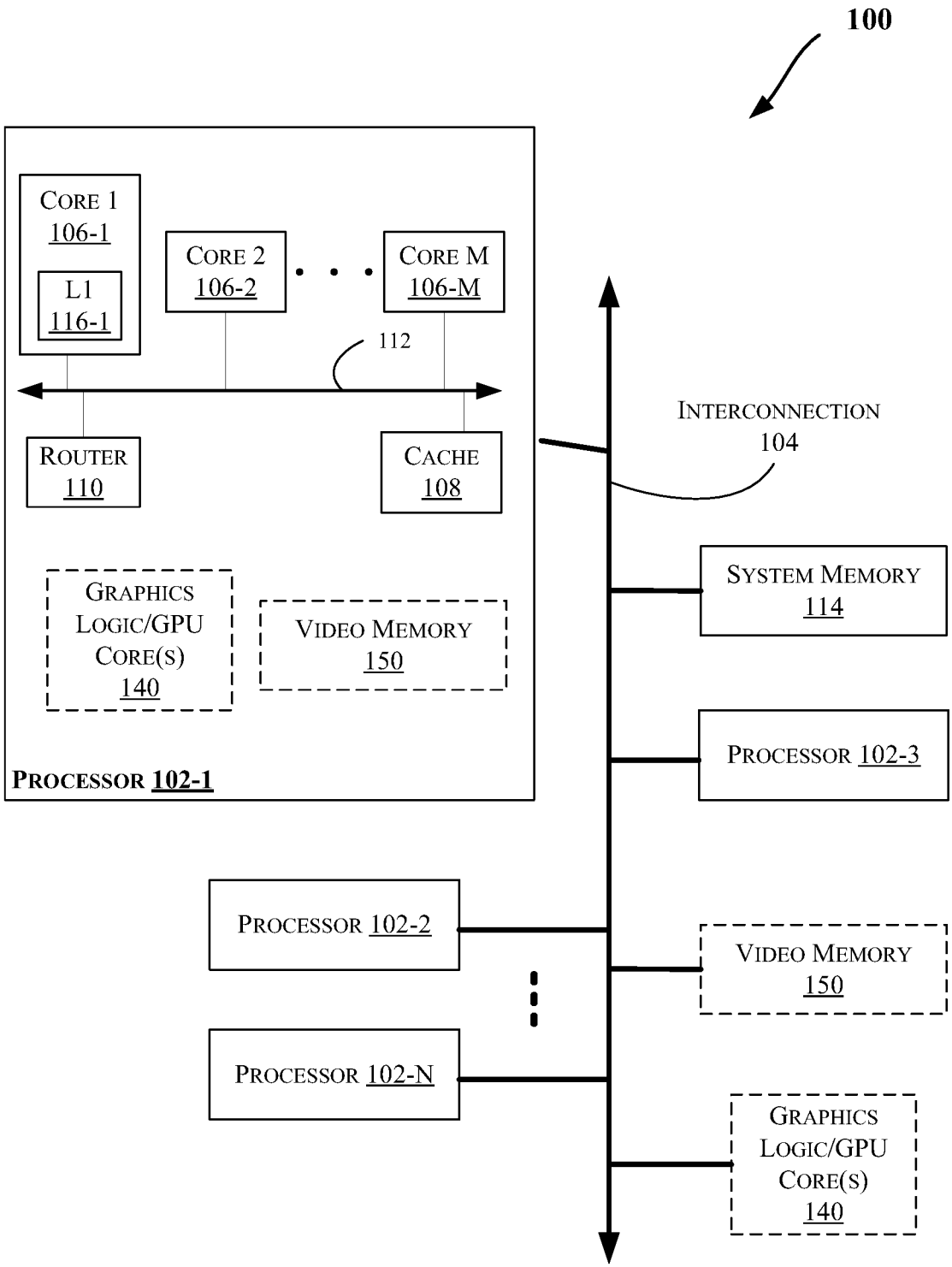

100
CORE 1
106-1
L1
116-1
CORE 2
106-2
CORE M
106-M
112
ROUTER
110
CACHE
108
GRAPHICS LOGIC/GPU CORE(S)
140
VIDEO MEMORY
150
PROCESSOR 102-1
INTERCONNECTION
104
SYSTEM MEMORY
114
PROCESSOR 102-3
PROCESSOR 102-2
VIDEO MEMORY
150
PROCESSOR 102-N
GRAPHICS LOGIC/GPU CORE(S)
140

FIG. 1

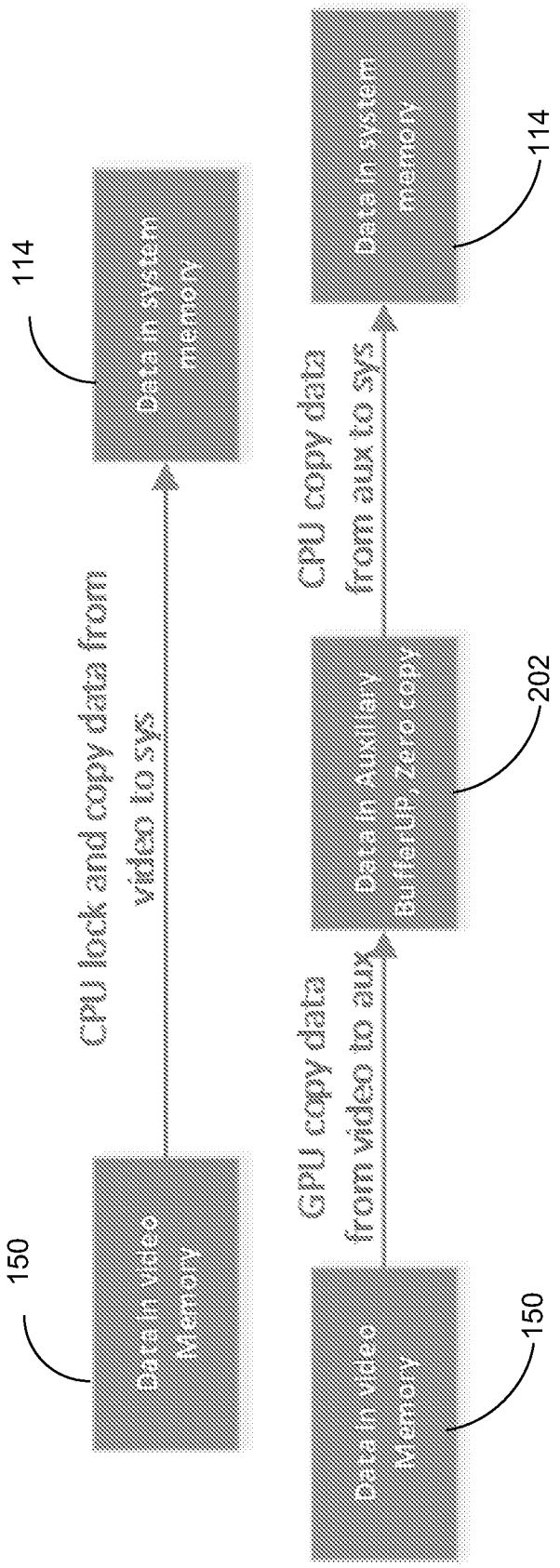

150
Data in video Memory
CPU lock and copy data from video to sys
114
Data in system memory
GPU copy data from video to aux
202
Data in Auxiliary Buffer(Up Zero copy
CPU copy data from aux to sys
114
Data in system memory
150
Data in Video Memory

FIG. 2

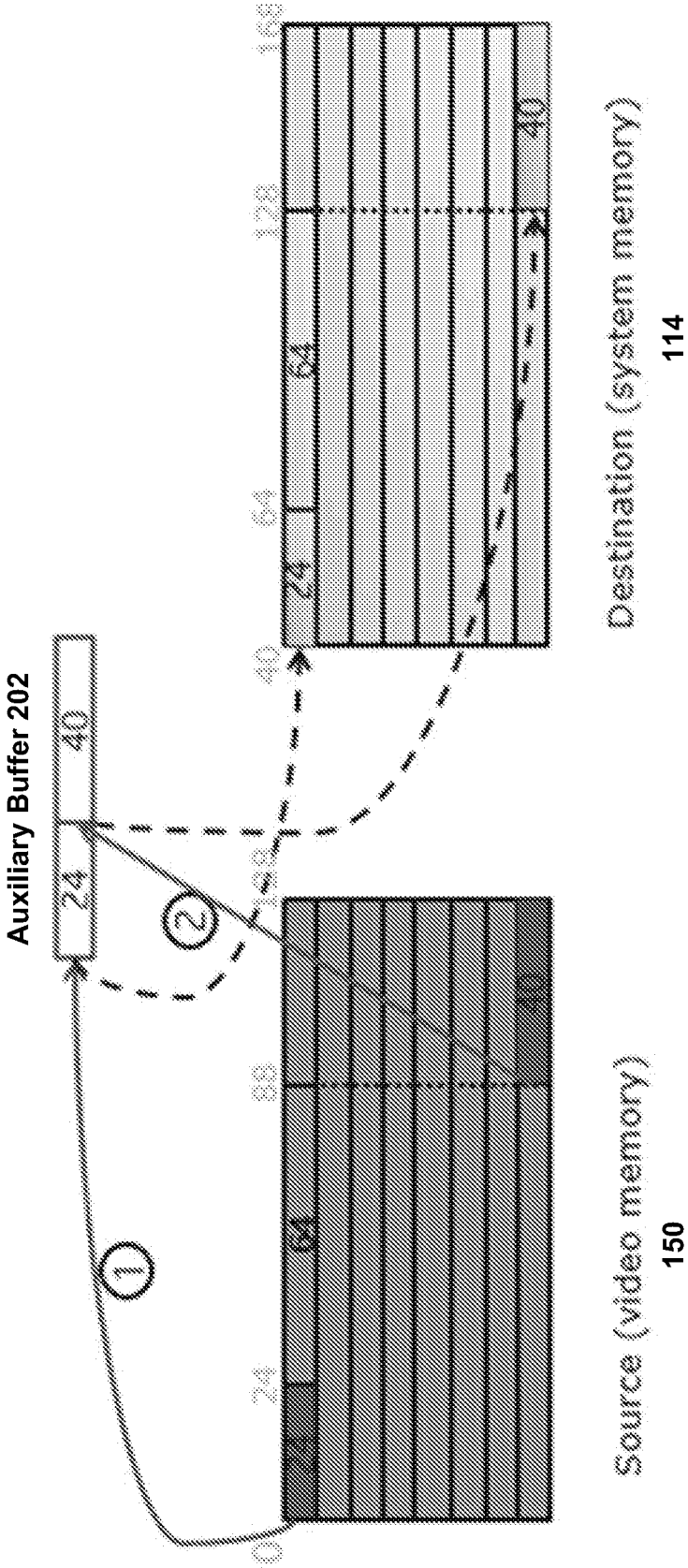

Auxiliary Buffer 202
40
24
Source (video memory)
150
Destination (system memory)
114
0
24
88
128
64
24
40
168
1
2

FIG. 3

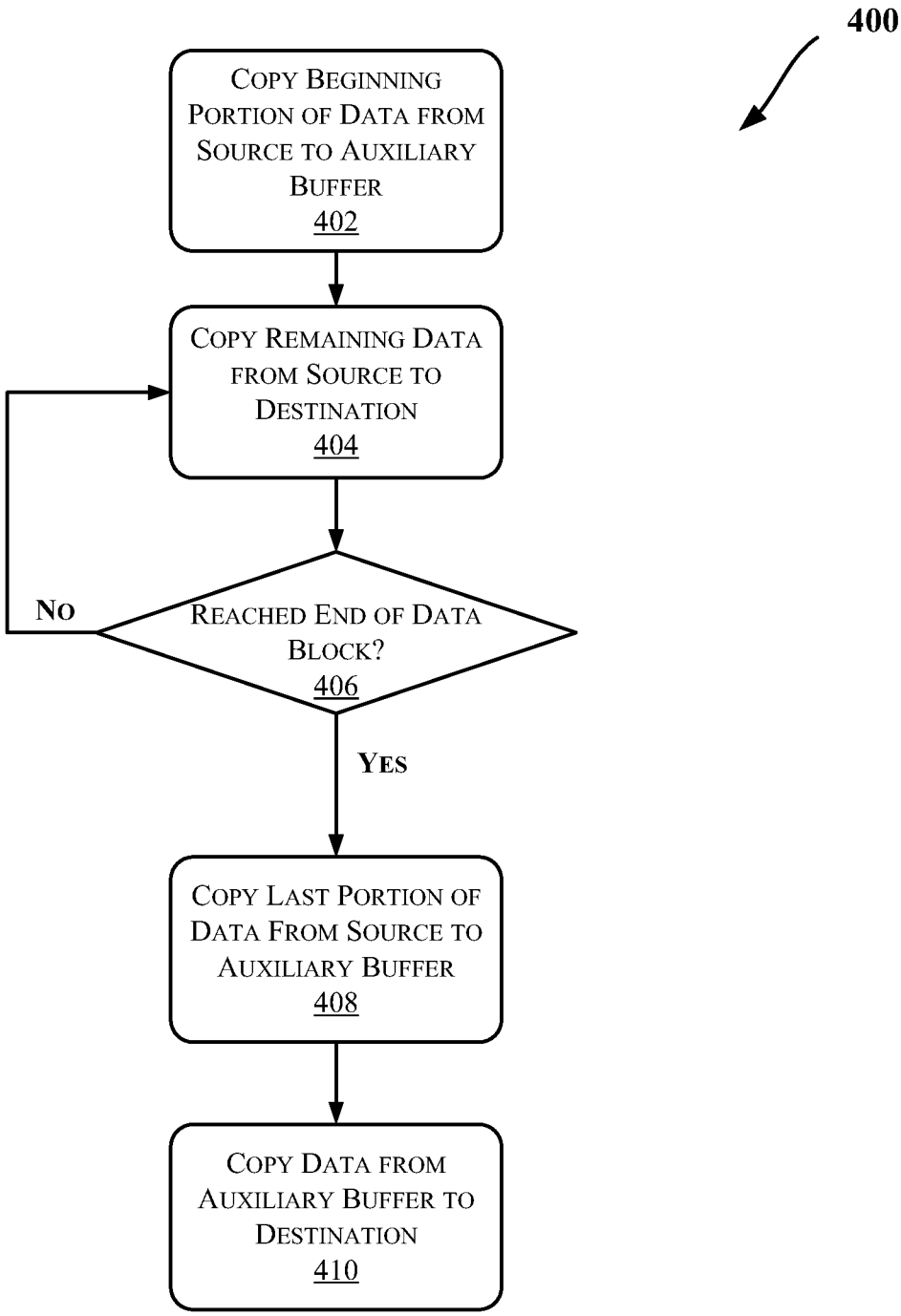

400
COPY BEGINNING PORTION OF DATA FROM SOURCE TO AUXILIARY BUFFER
402
COPY REMAINING DATA FROM SOURCE TO DESTINATION
404
REACHED END OF DATA BLOCK?
406
NO
YES
COPY LAST PORTION OF DATA FROM SOURCE TO AUXILIARY BUFFER
408
COPY DATA FROM AUXILIARY BUFFER TO DESTINATION
410

FIG. 4

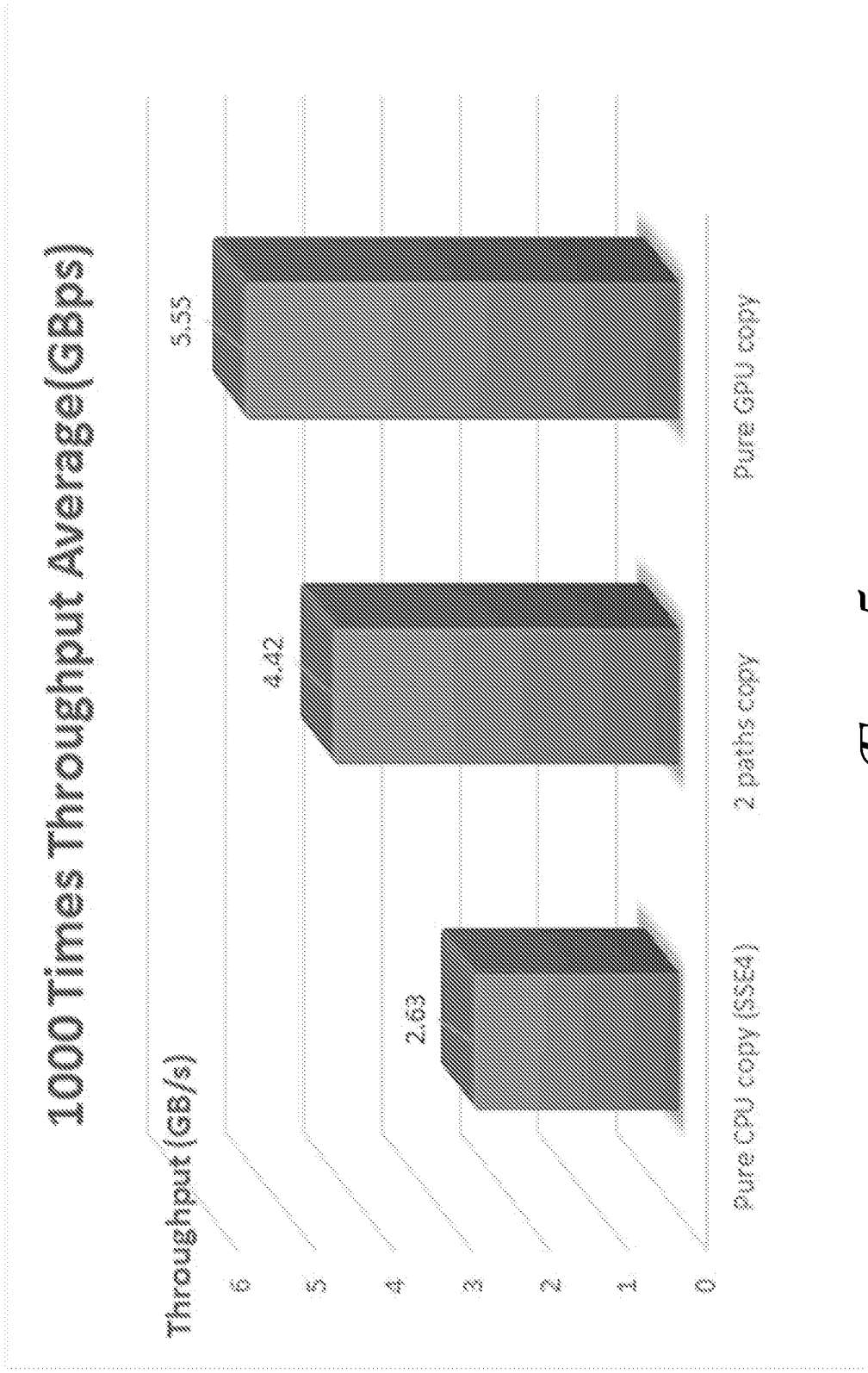

1000 Times Throughput Average(GBps)
Throughput (GB/s)
6
5
4
3
2
1
0
2.63
4.42
5.35
Pure CPU copy (SSE4)
2 paths copy
Pure GPU copy

FIG. 5

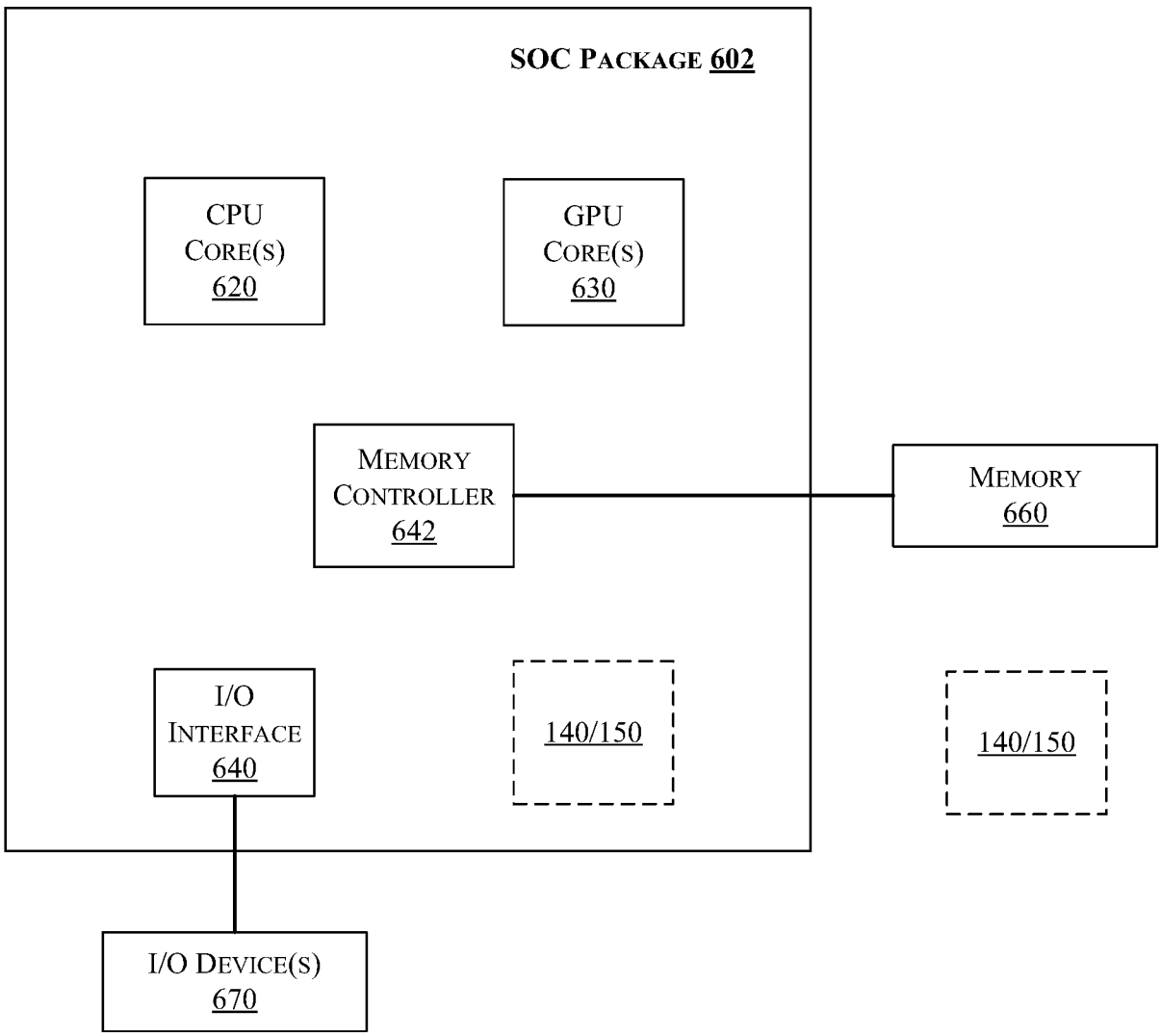

SOC PACKAGE 602
CPU CORE(S) 620
GPU CORE(S) 630
MEMORY CONTROLLER 642
MEMORY 660
I/O INTERFACE 640
140/150
140/150
I/O DEVICE(S) 670

FIG. 6

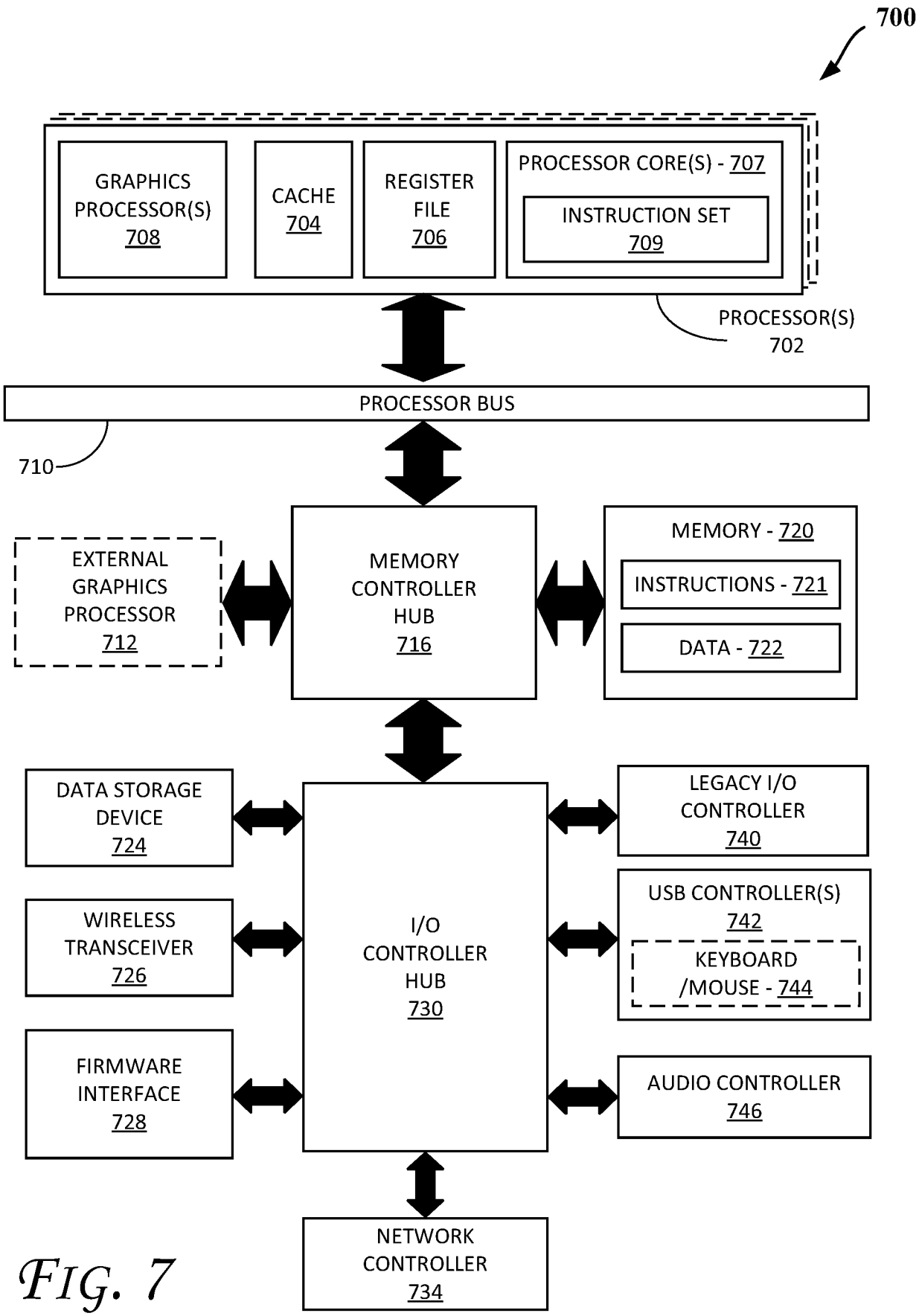

700
GRAPHICS PROCESSOR(S) 708
CACHE 704
REGISTER FILE 706
PROCESSOR CORE(S) - 707
INSTRUCTION SET 709
PROCESSOR(S) 702
PROCESSOR BUS
710
EXTERNAL GRAPHICS PROCESSOR 712
MEMORY CONTROLLER HUB 716
MEMORY - 720
INSTRUCTIONS - 721
DATA - 722
DATA STORAGE DEVICE 724
WIRELESS TRANSCEIVER 726
FIRMWARE INTERFACE 728
I/O CONTROLLER HUB 730
LEGACY I/O CONTROLLER 740
USB CONTROLLER(S) 742
KEYBOARD /MOUSE - 744
AUDIO CONTROLLER 746
NETWORK CONTROLLER 734

FIG. 7

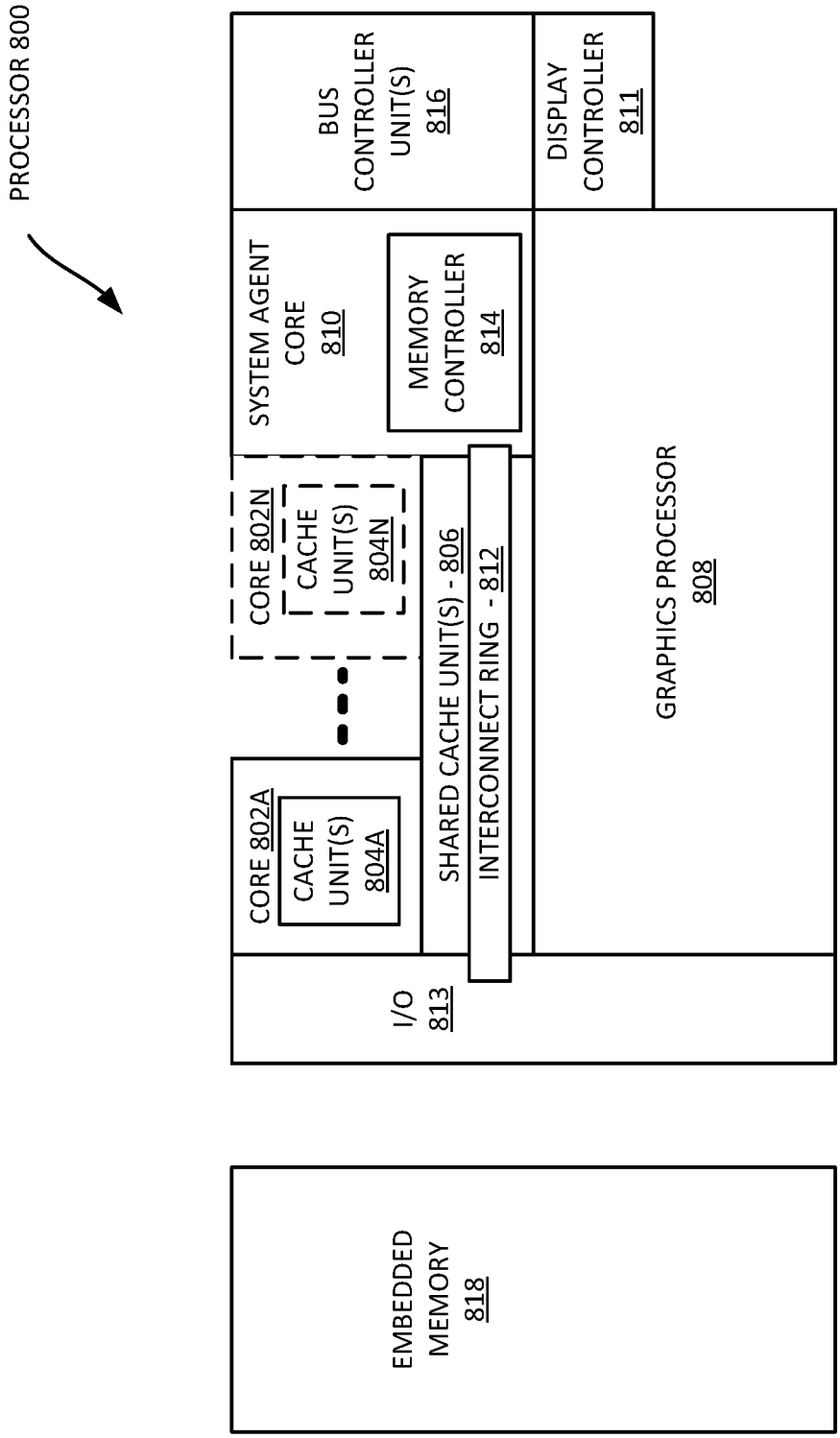

PROCESSOR 800
BUS CONTROLLER UNIT(S) 816
DISPLAY CONTROLLER 811
SYSTEM AGENT CORE 810
MEMORY CONTROLLER 814
CORE 802N
CACHE UNIT(S) 804N
CORE 802A
CACHE UNIT(S) 804A
SHARED CACHE UNIT(S) - 806
INTERCONNECT RING - 812
GRAPHICS PROCESSOR 808
I/O 813
EMBEDDED MEMORY 818

FIG. 8

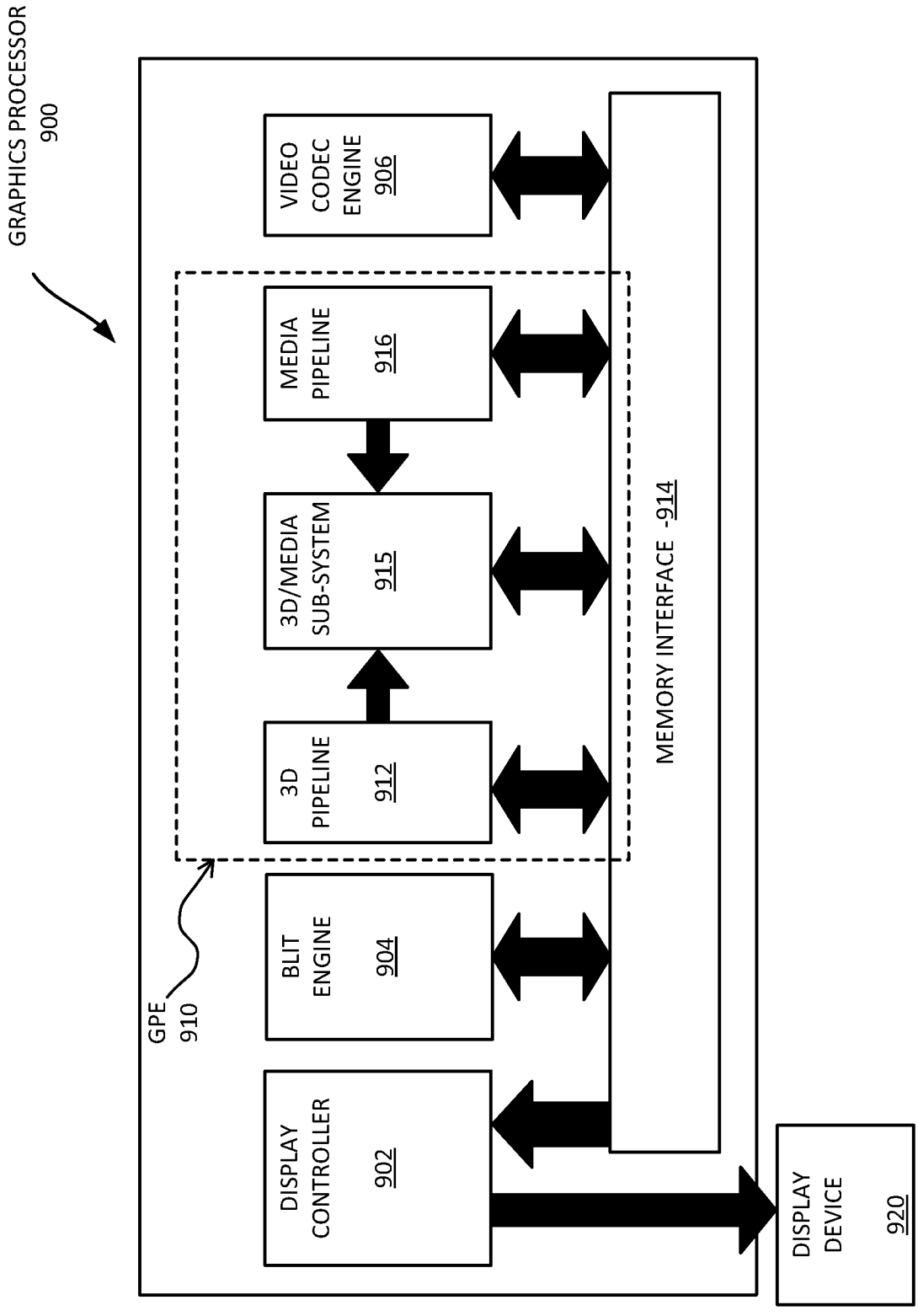

GRAPHICS PROCESSOR
900
VIDEO CODEC ENGINE
906
MEDIA PIPELINE
916
3D/MEDIA SUB-SYSTEM
915
3D PIPELINE
912
BLIT ENGINE
904
DISPLAY CONTROLLER
902
GPE
910
MEMORY INTERFACE -914
DISPLAY DEVICE
920

FIG. 9

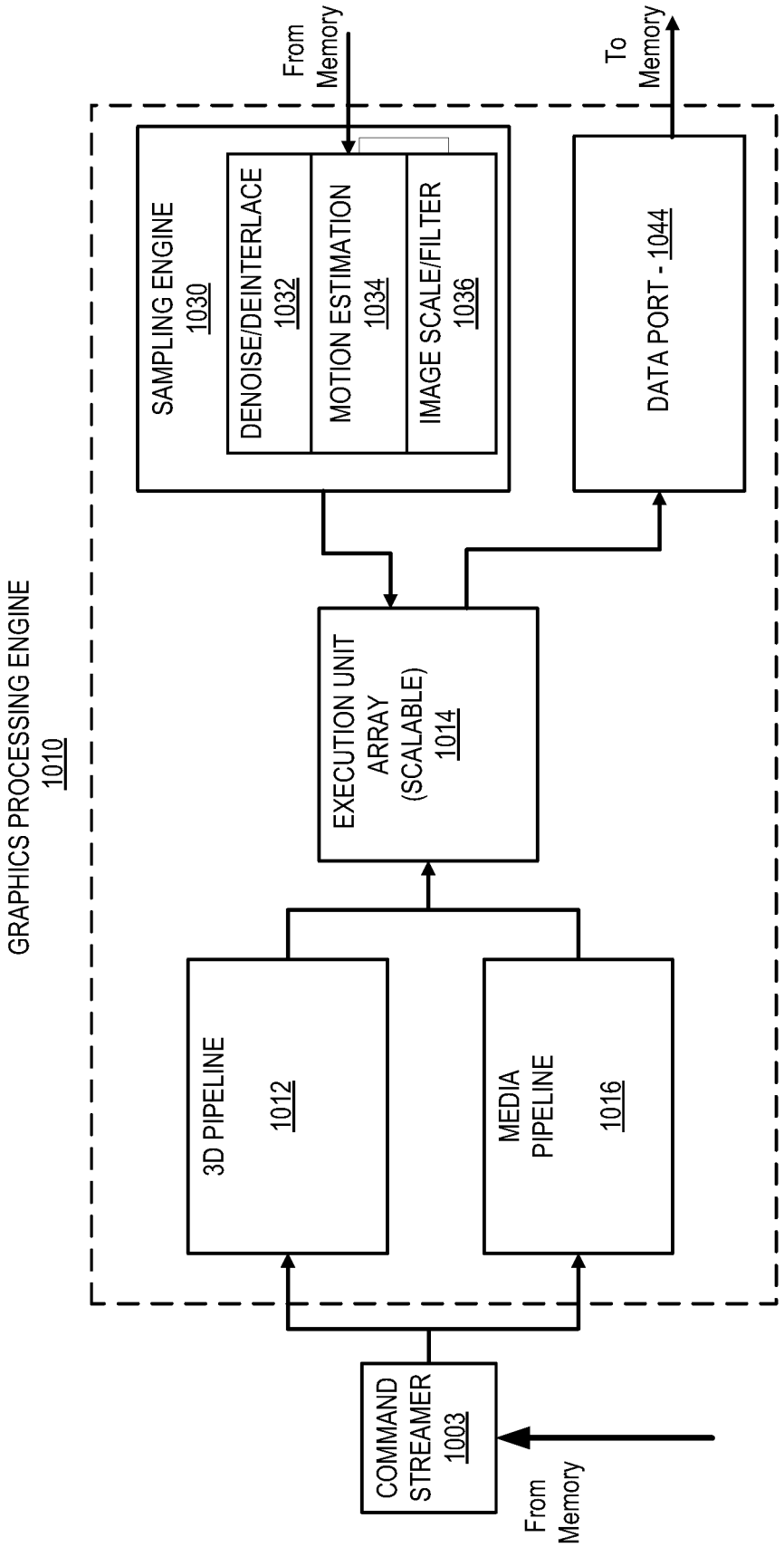

GRAPHICS PROCESSING ENGINE
1010
From Memory
To Memory
SAMPLING ENGINE
1030
DENOISE/DEINTERLACE
1032
MOTION ESTIMATION
1034
IMAGE SCALE/FILTER
1036
DATA PORT - 1044
EXECUTION UNIT ARRAY (SCALABLE)
1014
3D PIPELINE
1012
MEDIA PIPELINE
1016
COMMAND STREAMER
1003
From Memory

FIG. 10

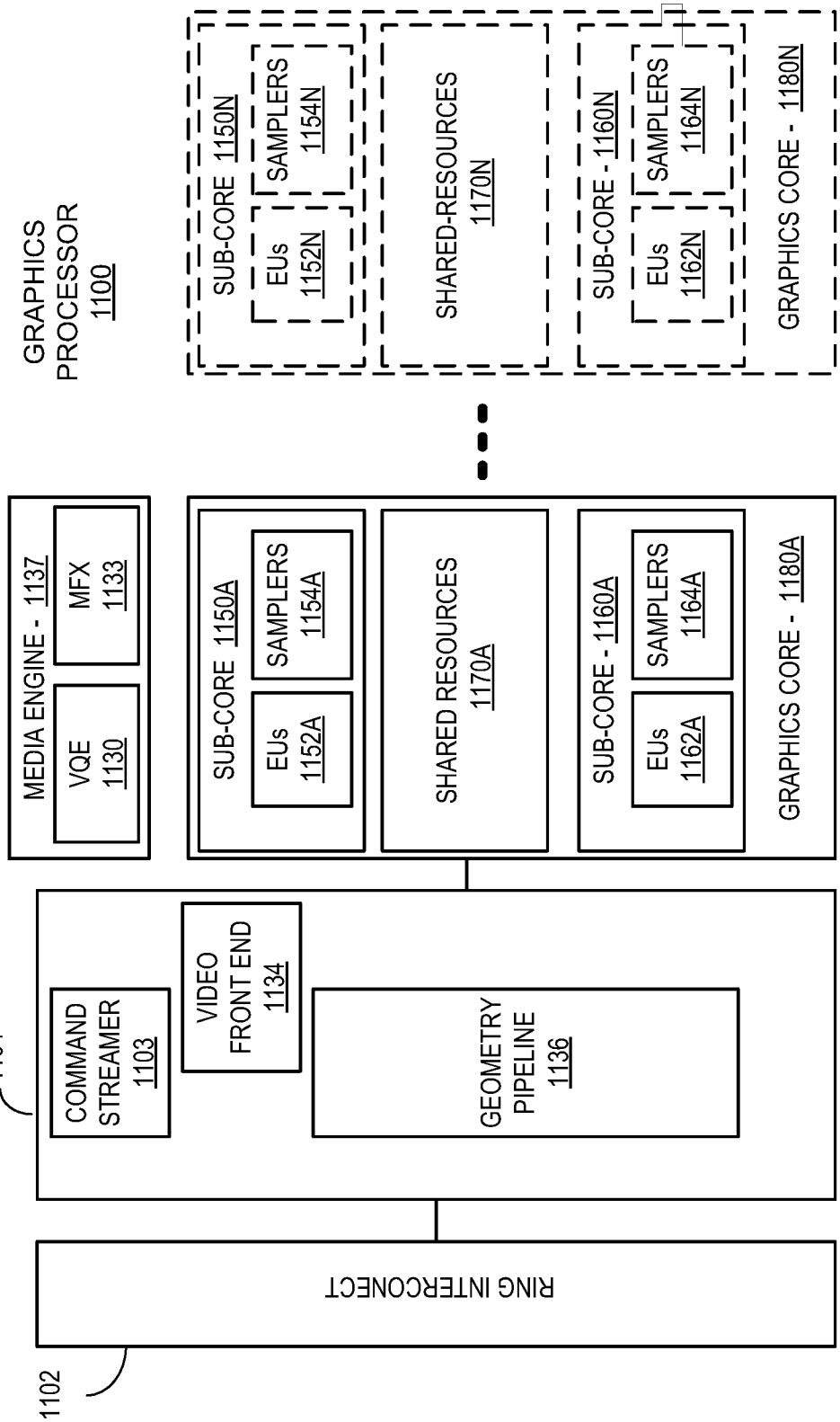

GRAPHICS PROCESSOR 1100
SUB-CORE 1150N
EUs 1152N
SAMPLERS 1154N
SHARED-RESOURCES 1170N
SUB-CORE - 1160N
EUs 1162N
SAMPLERS 1164N
GRAPHICS CORE - 1180N
MEDIA ENGINE - 1137
VQE 1130
MFX 1133
SUB-CORE 1150A
EUs 1152A
SAMPLERS 1154A
SHARED RESOURCES 1170A
SUB-CORE - 1160A
EUs 1162A
SAMPLERS 1164A
GRAPHICS CORE - 1180A
1104
COMMAND STREAMER 1103
VIDEO FRONT END 1134
GEOMETRY PIPELINE 1136
RING INTERCONECT
1102

FIG. 11

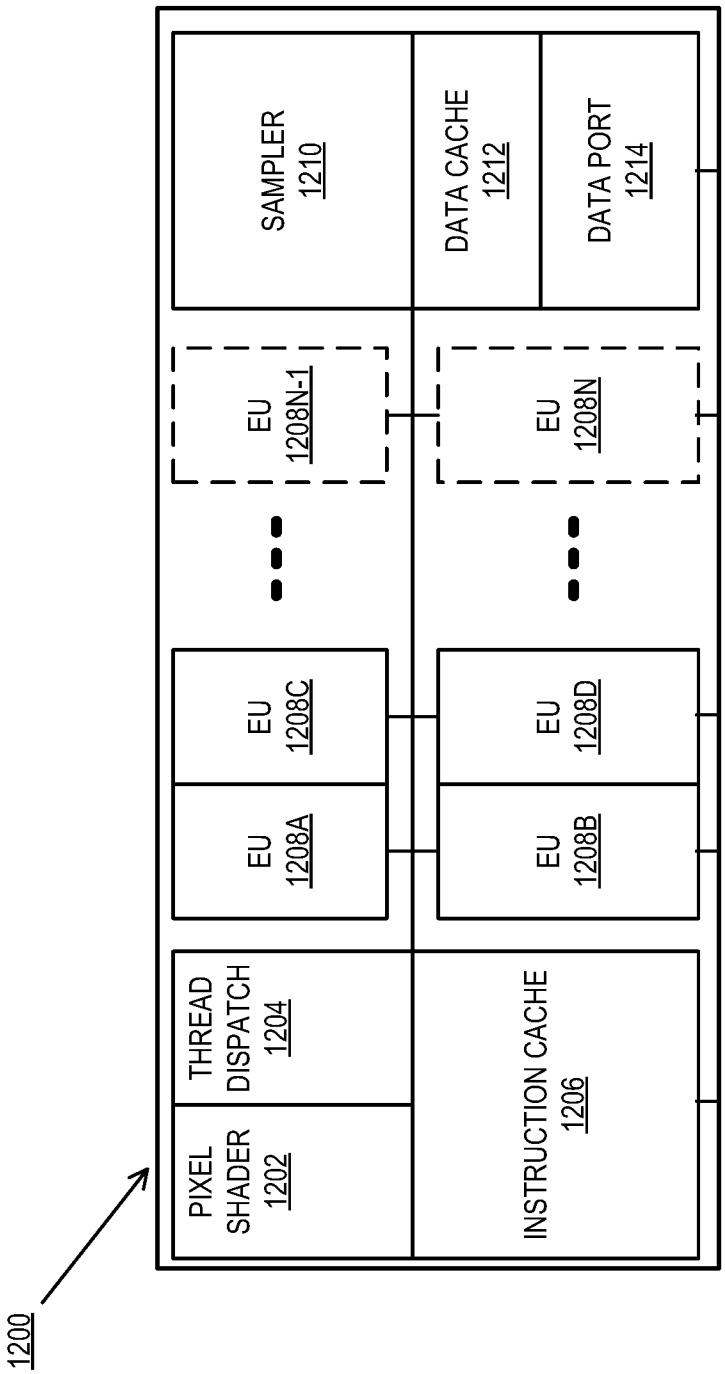

1200
PIXEL SHADER 1202
THREAD DISPATCH 1204
INSTRUCTION CACHE 1206
EU 1208A
EU 1208B
EU 1208C
EU 1208D
EU 1208N-1
EU 1208N
SAMPLER 1210
DATA CACHE 1212
DATA PORT 1214

FIG. 12

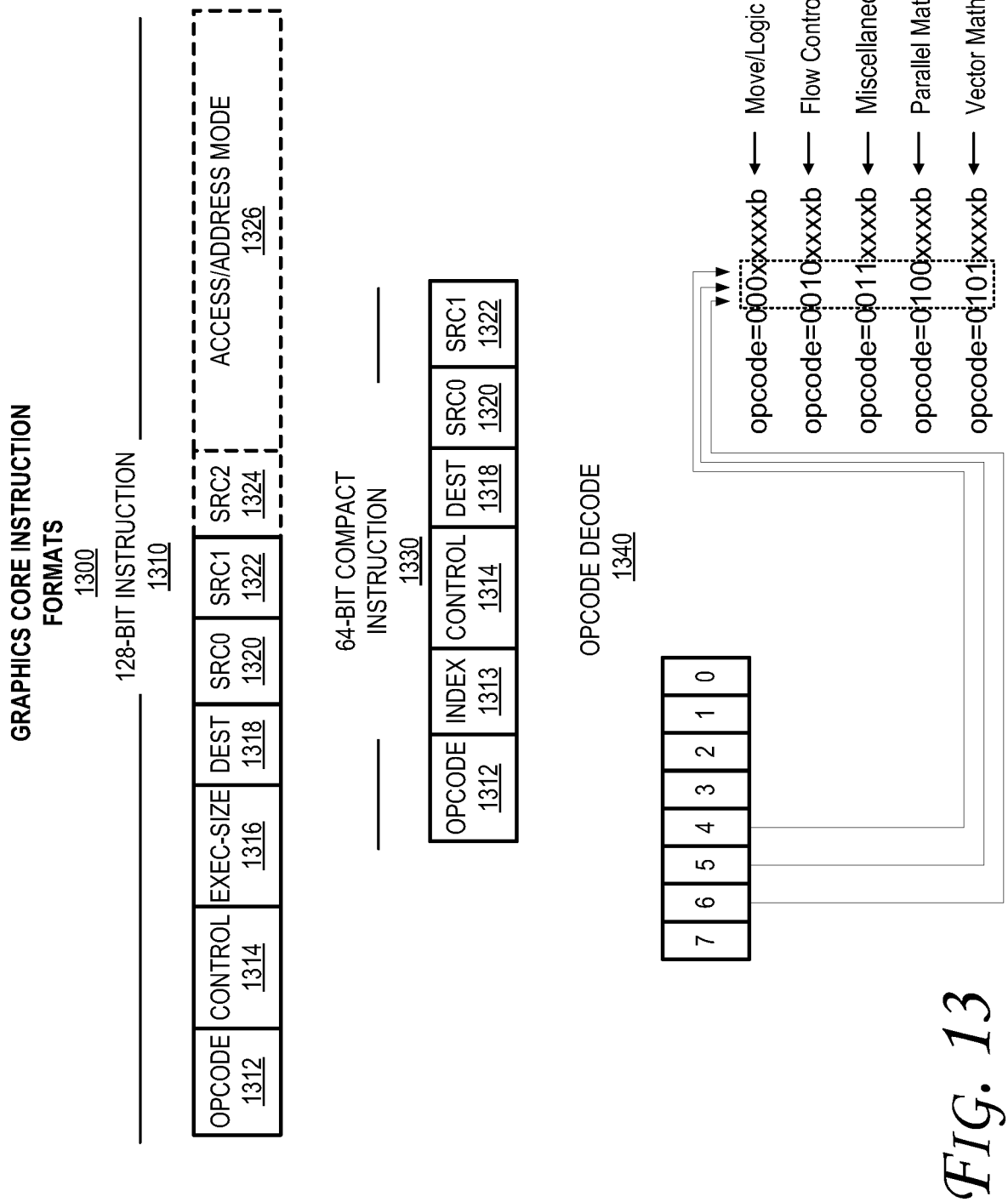

GRAPHICS CORE INSTRUCTION FORMATS
1300
128-BIT INSTRUCTION
1310
OPCODE 1312
CONTROL 1314
EXEC-SIZE 1316
DEST 1318
SRC0 1320
SRC1 1322
SRC2 1324
ACCESS/ADDRESS MODE 1326
64-BIT COMPACT INSTRUCTION
1330
OPCODE 1312
INDEX 1313
CONTROL 1314
DEST 1318
SRC0 1320
SRC1 1322
OPCODE DECODE
1340
7 6 5 4 3 2 1 0
opcode=0000xxxxb
opcode=0010xxxxb
opcode=0011xxxxb
opcode=0100xxxxb
opcode=0101xxxxb
Move/Logic - 1342
Flow Control - 1344
Miscellaneous - 1346
Parallel Math - 1348
Vector Math - 1350

FIG. 13

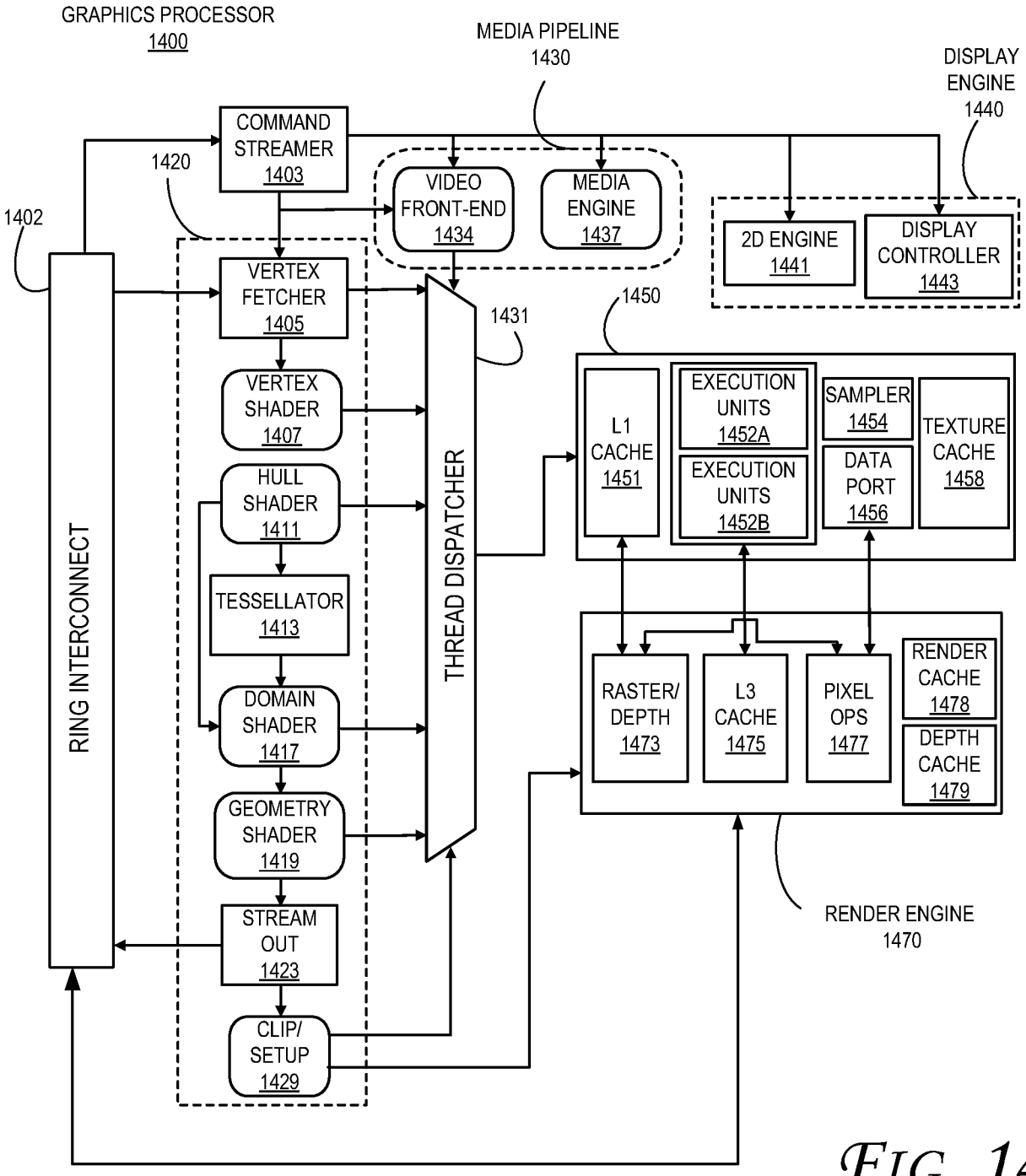

GRAPHICS PROCESSOR
1400
MEDIA PIPELINE
1430
DISPLAY ENGINE
1440
COMMAND STREAMER
1403
1420
1402
VIDEO FRONT-END
1434
MEDIA ENGINE
1437
2D ENGINE
1441
DISPLAY CONTROLLER
1443
VERTEX FETCHER
1405
1431
1450
VERTEX SHADER
1407
HULL SHADER
1411
TESSELLATOR
1413
DOMAIN SHADER
1417
GEOMETRY SHADER
1419
STREAM OUT
1423
CLIP/ SETUP
1429
RING INTERCONNECT
THREAD DISPATCHER
L1 CACHE
1451
EXECUTION UNITS
1452A
EXECUTION UNITS
1452B
SAMPLER
1454
DATA PORT
1456
TEXTURE CACHE
1458
RASTER/ DEPTH
1473
L3 CACHE
1475
PIXEL OPS
1477
RENDER CACHE
1478
DEPTH CACHE
1479
RENDER ENGINE
1470

FIG. 14

FIG. 15A

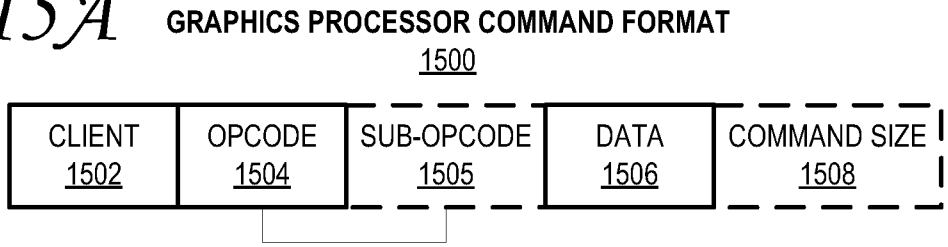
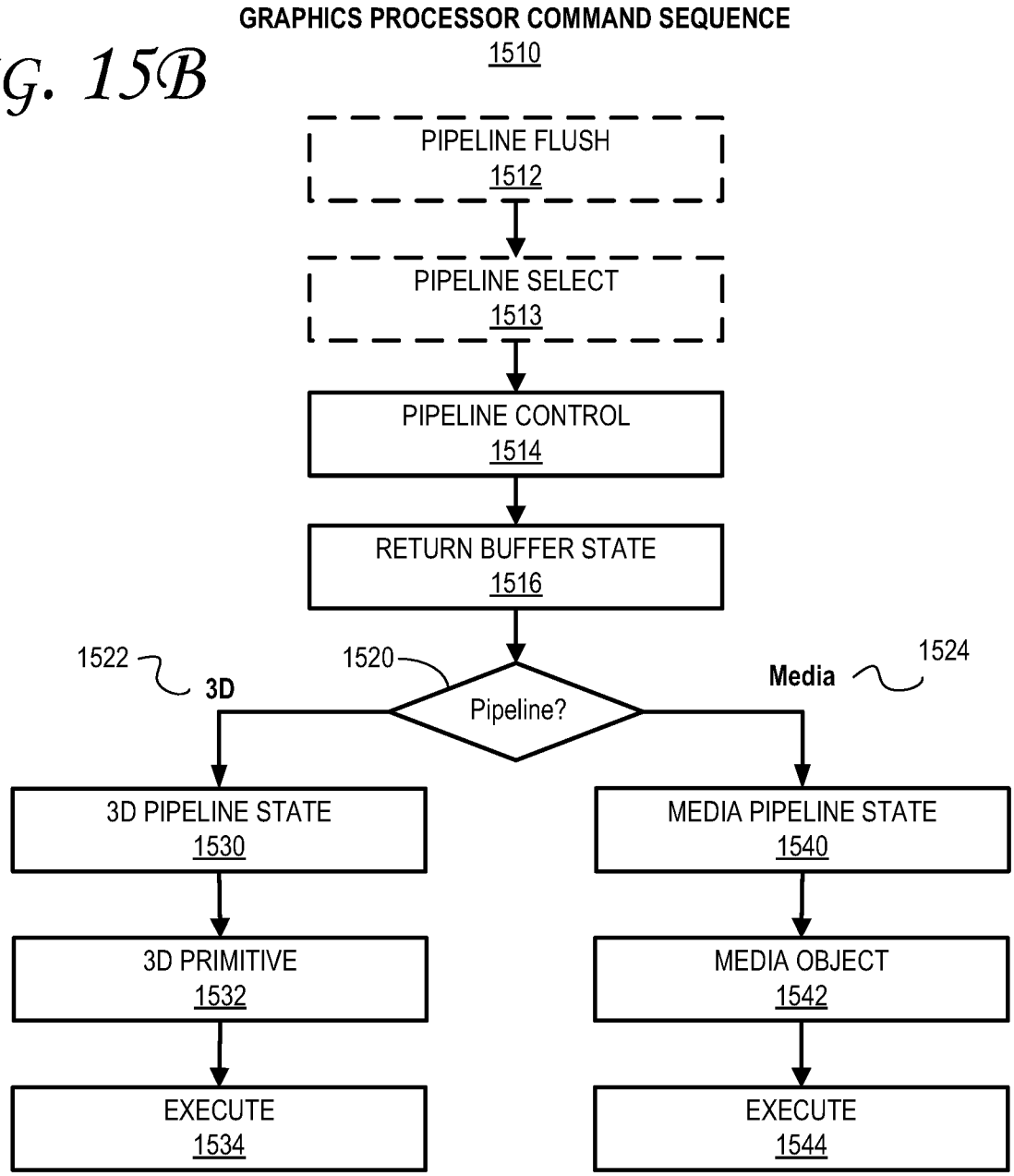


FIG. 15B



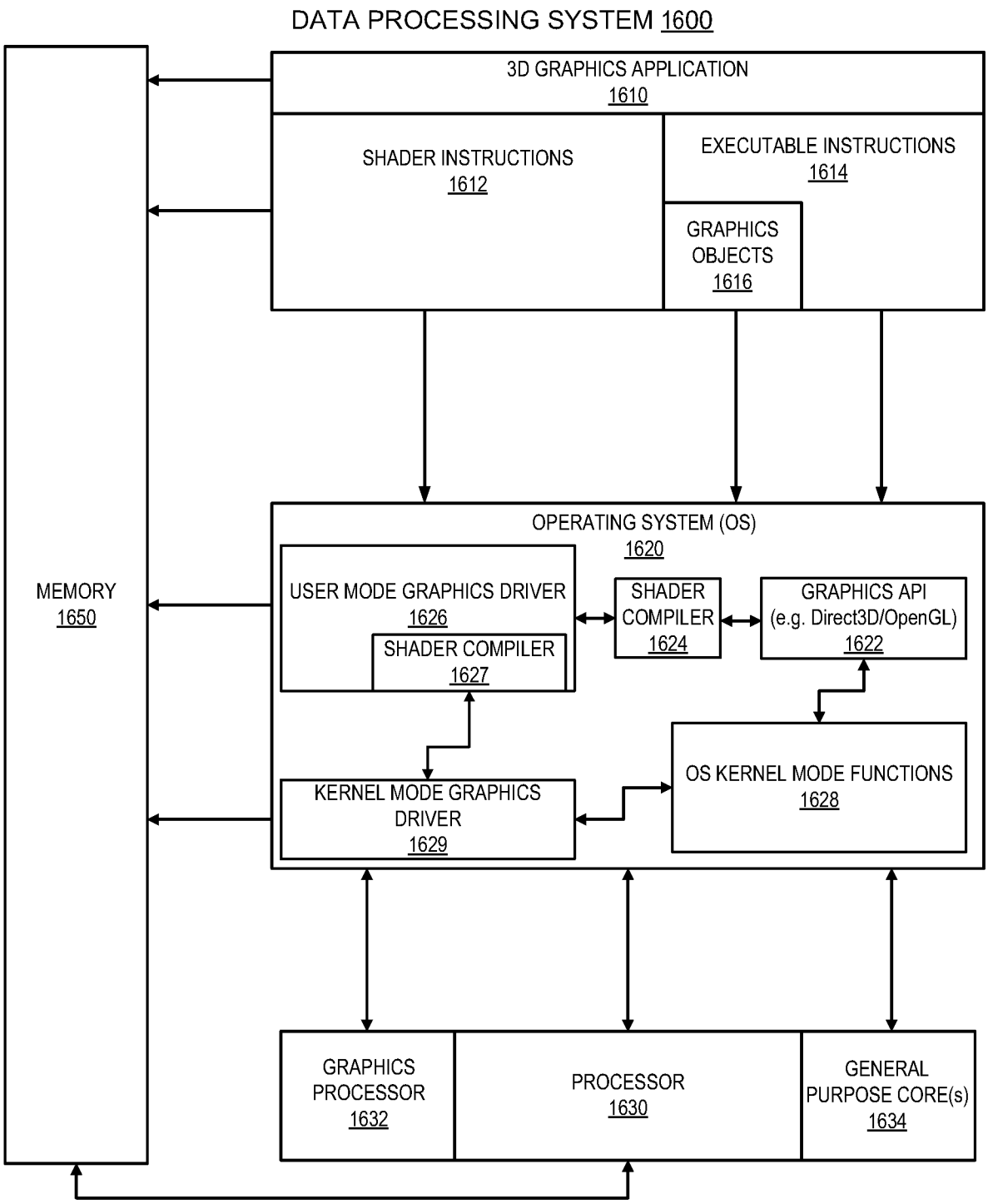

DATA PROCESSING SYSTEM 1600
3D GRAPHICS APPLICATION
1610
SHADER INSTRUCTIONS
1612
EXECUTABLE INSTRUCTIONS
1614
GRAPHICS OBJECTS
1616
MEMORY
1650
OPERATING SYSTEM (OS)
1620
USER MODE GRAPHICS DRIVER
1626
SHADER COMPILER
1627
SHADER COMPILER
1624
GRAPHICS API
(e.g. Direct3D/OpenGL)
1622
OS KERNEL MODE FUNCTIONS
1628
KERNEL MODE GRAPHICS DRIVER
1629
GRAPHICS PROCESSOR
1632
PROCESSOR
1630
GENERAL PURPOSE CORE(s)
1634

FIG. 16

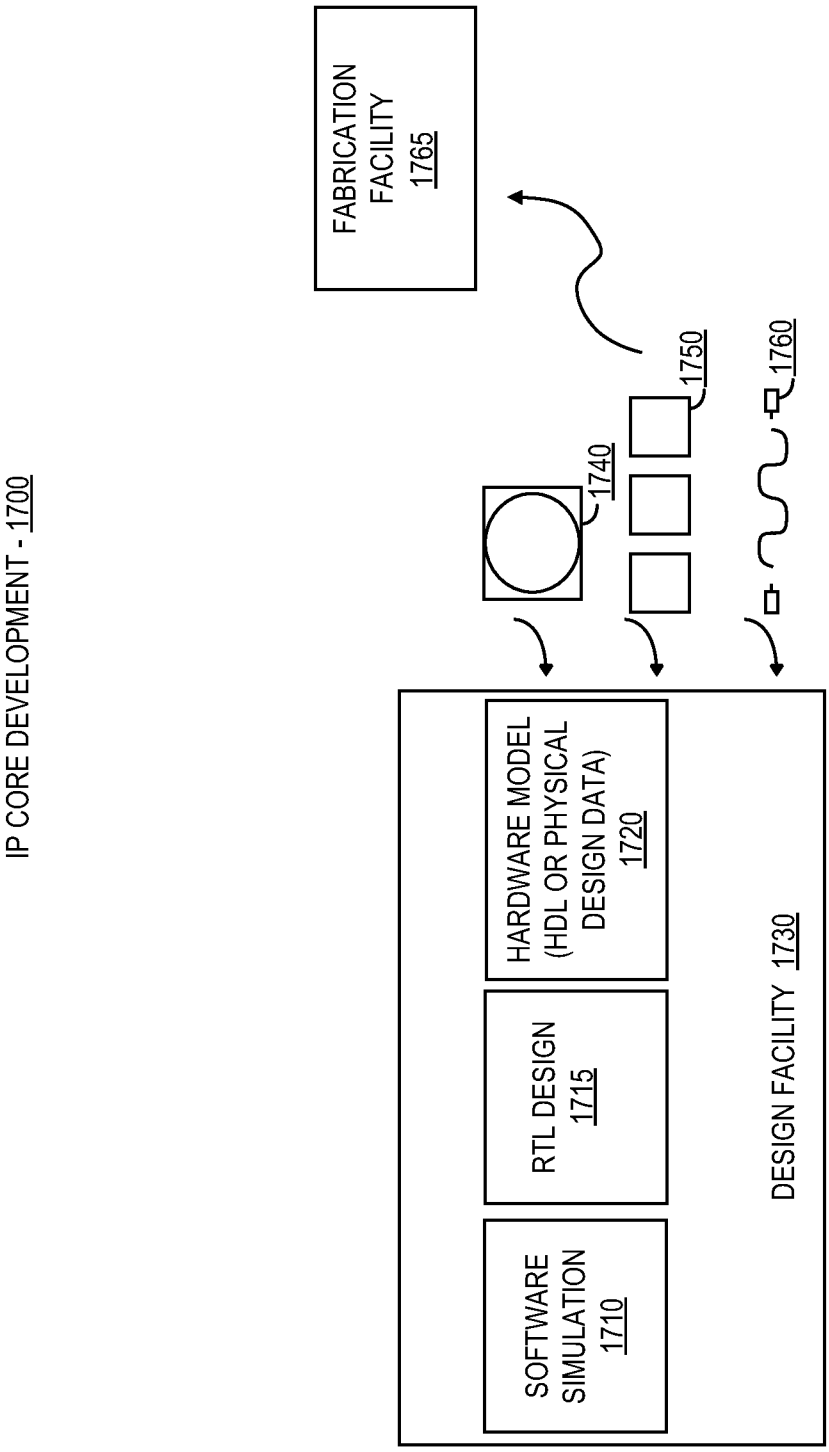

IP CORE DEVELOPMENT - 1700
FABRICATION FACILITY 1765
1740
1750
1760
HARDWARE MODEL (HDL OR PHYSICAL DESIGN DATA) 1720
RTL DESIGN 1715
SOFTWARE SIMULATION 1710
DESIGN FACILITY 1730

FIG. 17

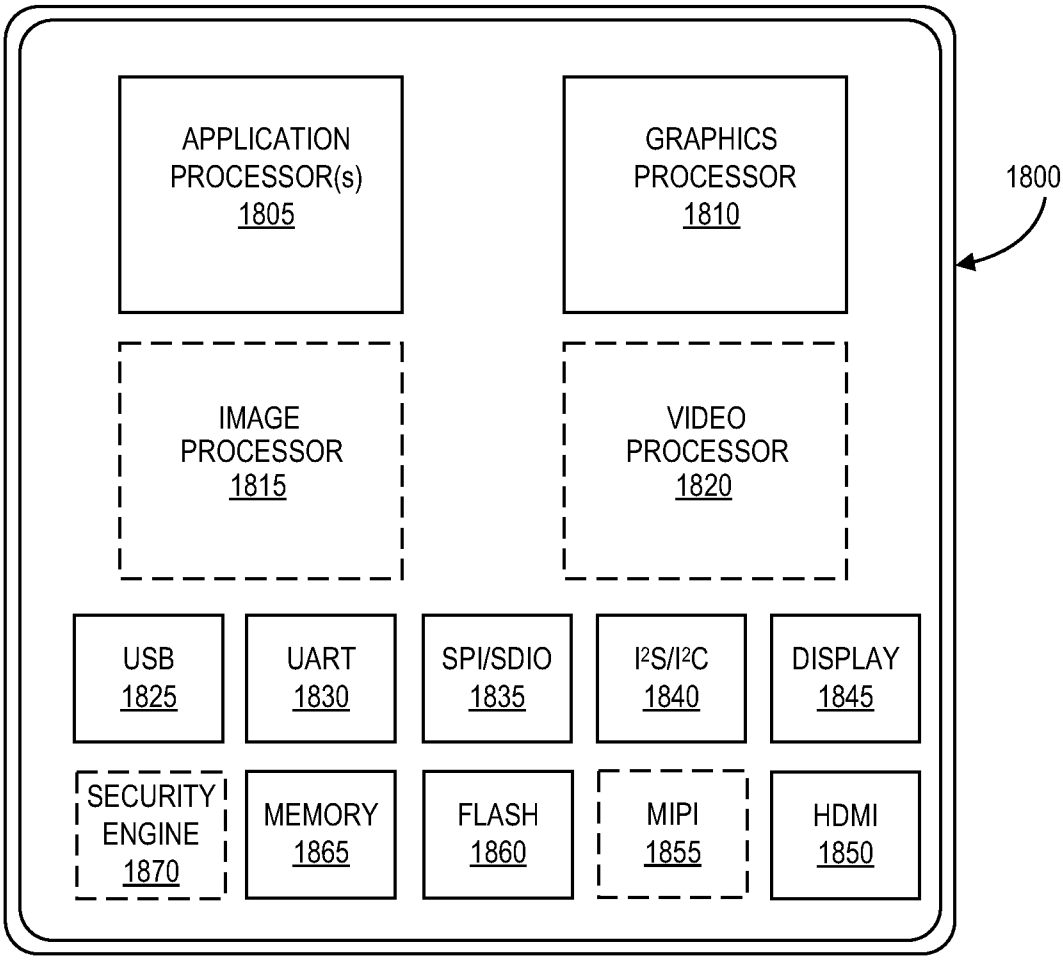

APPLICATION PROCESSOR(s) 1805
GRAPHICS PROCESSOR 1810
1800
IMAGE PROCESSOR 1815
VIDEO PROCESSOR 1820
USB 1825
UART 1830
SPI/SDIO 1835
I²S/I²C 1840
DISPLAY 1845
SECURITY ENGINE 1870
MEMORY 1865
FLASH 1860
MIPI 1855
HDMI 1850

FIG. 18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2015/090701

A. CLASSIFICATION OF SUBJECT MATTER

G06F 13/00(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)

EPODOC, WPI, CNPAT, CNKI, IEEE: GPU, graphic, process, CPU, central, copy, move, transfer, image, video, portion, part, source, destination, memory, remaind, buffer, align, complete, first, second

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014043347 A1 (ELECTRONICS AND TELECOMMUNICATIONS RESEARCH INSTITUTE) 13 February 2014 (2014-02-13) description, paragraphs [0020] to [0027], and figure 1	1-25
A	US 2014192069 A1 (RAIKIN, SHLOMO ET AL.) 10 July 2014 (2014-07-10) the whole document	1-25
A	US 2008117220 A1 (NEURALA LLC) 22 May 2008 (2008-05-22) the whole document	1-25
A	CN 103282888 A (HUAWEI TECHNOLOGIES CO., LTD.) 04 September 2013 (2013-09-04) the whole document	1-25

☐ 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	Date of mailing of the international search report
16 May 2016	**22 June 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 **YANG,Jishuang** Telephone No. **(86-10)62414422**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2015/090701

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	2014043347	A1	13 February 2014	KR	20140021388	A	20 February 2014
US	2014192069	A1	10 July 2014	TW	201346831	A	16 November 2013
				WO	2013095475	A1	27 June 2013
				CN	104025065	A	03 September 2014
US	2008117220	A1	22 May 2008	US	2014192073	A1	10 July 2014
CN	103282888	A	04 September 2013	WO	2013097098	A1	04 July 2013