US012437371B2

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

(10) **Patent No.: US 12,437,371 B2**
(45) **Date of Patent: Oct. 7, 2025**

(54) **ILLUMINATION RESAMPLING USING TEMPORAL GRADIENTS IN LIGHT TRANSPORT SIMULATION SYSTEMS AND APPLICATIONS**

(71) Applicant: **Nvidia Corporation**, Santa Clara, CA (US)

(72) Inventors: **Alexey Panteleev**, El Dorado Hills, CA (US); **Chris Wyman**, Redmond, WA (US)

(73) Assignee: **Nvidia Corporation**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 267 days.

(21) Appl. No.: **17/975,450**

(22) Filed: **Oct. 27, 2022**

(65) **Prior Publication Data**

US 2023/0138718 A1 May 4, 2023

Related U.S. Application Data

(60) Provisional application No. 63/273,834, filed on Oct. 29, 2021.

(51) **Int. Cl.**
G06T 5/70 (2024.01)
G06T 3/4053 (2024.01)
(Continued)

(52) **U.S. Cl.**
CPC ***G06T 5/70*** (2024.01); ***G06T 3/4053*** (2013.01); ***G06T 5/20*** (2013.01); ***G06T 15/005*** (2013.01); ***G06T 15/506*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2004/0096106 A1* 5/2004 Demi G06T 7/66 382/199
2010/0253779 A1* 10/2010 Itoh G06V 20/52 348/143

(Continued)

OTHER PUBLICATIONS

International Search Report and Written Opinion issued in PCT Application No. PCT/US2022/048169, dated Feb. 22, 2023.

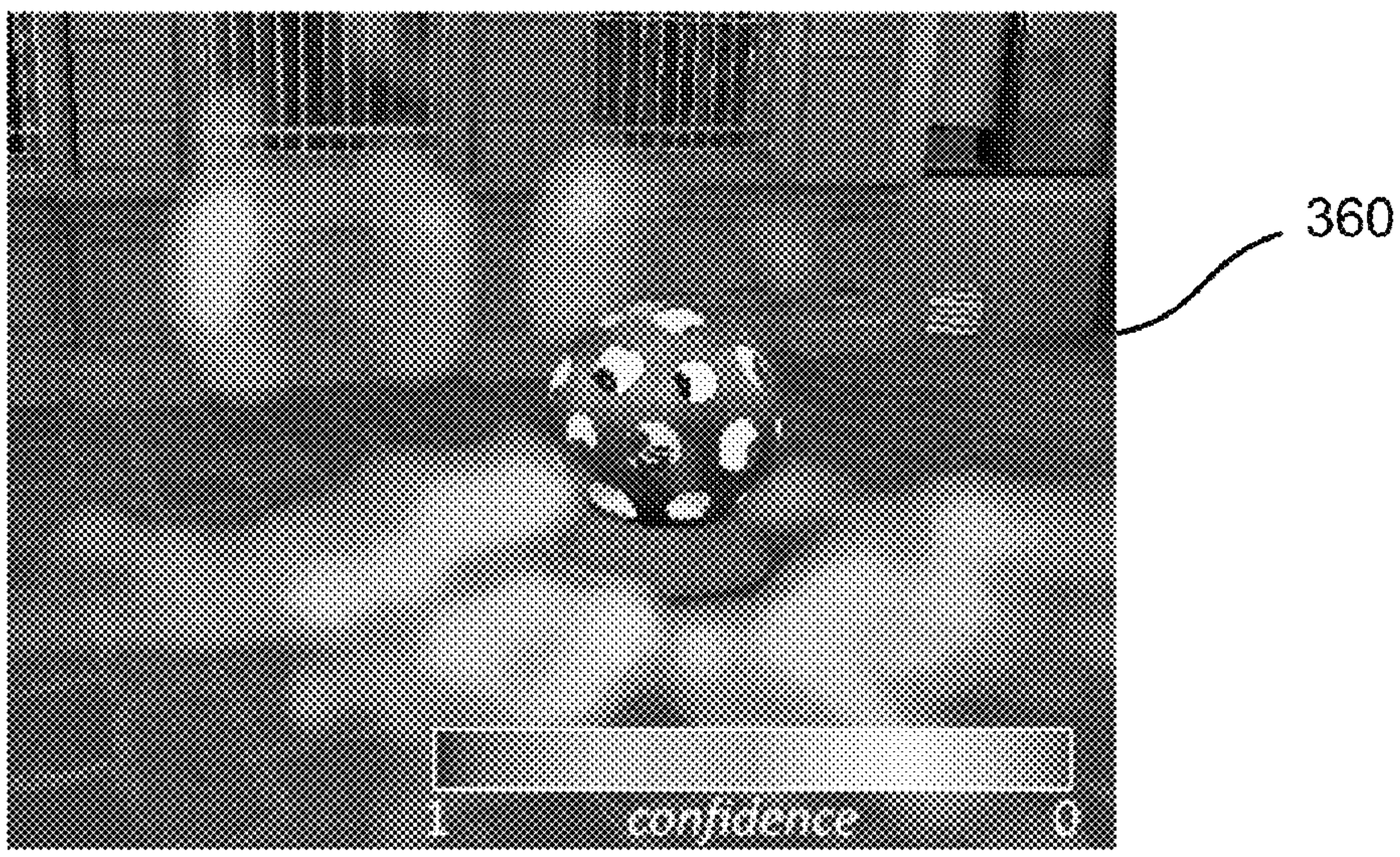
(Continued)

Primary Examiner — Jeffrey J Chow
(74) *Attorney, Agent, or Firm* — Hogan Lovells US LLP

(57) **ABSTRACT**

Systems and methods described relate to the generation of image content. In order to provide for smoothing between sequential images, but avoid introducing lag into lighting effects, light information can be compared for regions between consecutive rendered frames. Shading can be performed and the results compared for tiles of pixels to compute gradient values, such as by using a single light sample for each tile. A filtering pass can be performed with respect to these gradients, and this filtered, lower-resolution grid version can be upscaled into a full resolution, screen-sized image and the gradients transformed into confidence values. These confidence values can be used to determine an extent to which to keep lighting data from the previous frame with respect to the current frame. For example, less lighting information can be used from the prior frame for a given pixel location if the confidence for that location is lower.

20 Claims, 36 Drawing Sheets



(51) **Int. Cl.**
G06T 5/20 (2006.01)
G06T 15/00 (2011.01)
G06T 15/50 (2011.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

2011/0261207	A1*	10/2011	Strandemar	H04N 23/20 382/284
2012/0183196	A1*	7/2012	Dasgupta	G06T 1/20 382/132
2013/0266057	A1*	10/2013	Kokaram	G06T 5/70 375/E7.227
2016/0037059	A1*	2/2016	Lim	G06T 5/50 348/241
2020/0058103	A1*	2/2020	Liu	G06T 15/50

OTHER PUBLICATIONS

Christoph Schied et al: "Gradient Estimation for Real-time Adaptive Temporal Filtering", Proceedings of the ACM On Computer Graphics and Interactive Techniques, vol. 1, No. 2, Aug. 24, 2018 (Aug. 24, 2018) pp. 1-16, XP058413761, DOI: 10.1145/3233301, Section 3 Gradient Estimation, Section 4 Applications.

Mueller Joerg et al: "Temporally Adaptive Shading Reuse for Real-Time Rendering and Virtual Reality", ACM Transactions On Graphics, ACM, NY, US, vol. 40, No. 2, Apr. 27, 2021 (Apr. 27, 2021) , pp. 1-14, XP058683701, ISSN: 0730-0301, DOI: 10.1145/3446790, the whole document.

Christoph Schied et al: "Spatiotemporal variance-guided filtering : real-time reconstruction for path-traced global illumination", Proceedings of High Performance Graphics On , HPG '17, Jul. 28, 2017 (Jul. 28, 2017), pp. 1-12, XP055817590, DOI: 10.1145/3105762.3105770 ISBN: 978-1-4503-5101-0, the whole document.

Bitterli Benedikt et al: "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting", ACM Transactions On Graphics, ACM, NY, US, vol. 39, No. 4, Jul. 8, 2020 (Jul. 8, 2020), pp. 148:1-148:17, XP058683417, ISSN: 0730-0301, DOI: 10.1145/3386569.3392481, the whole document.

* cited by examiner

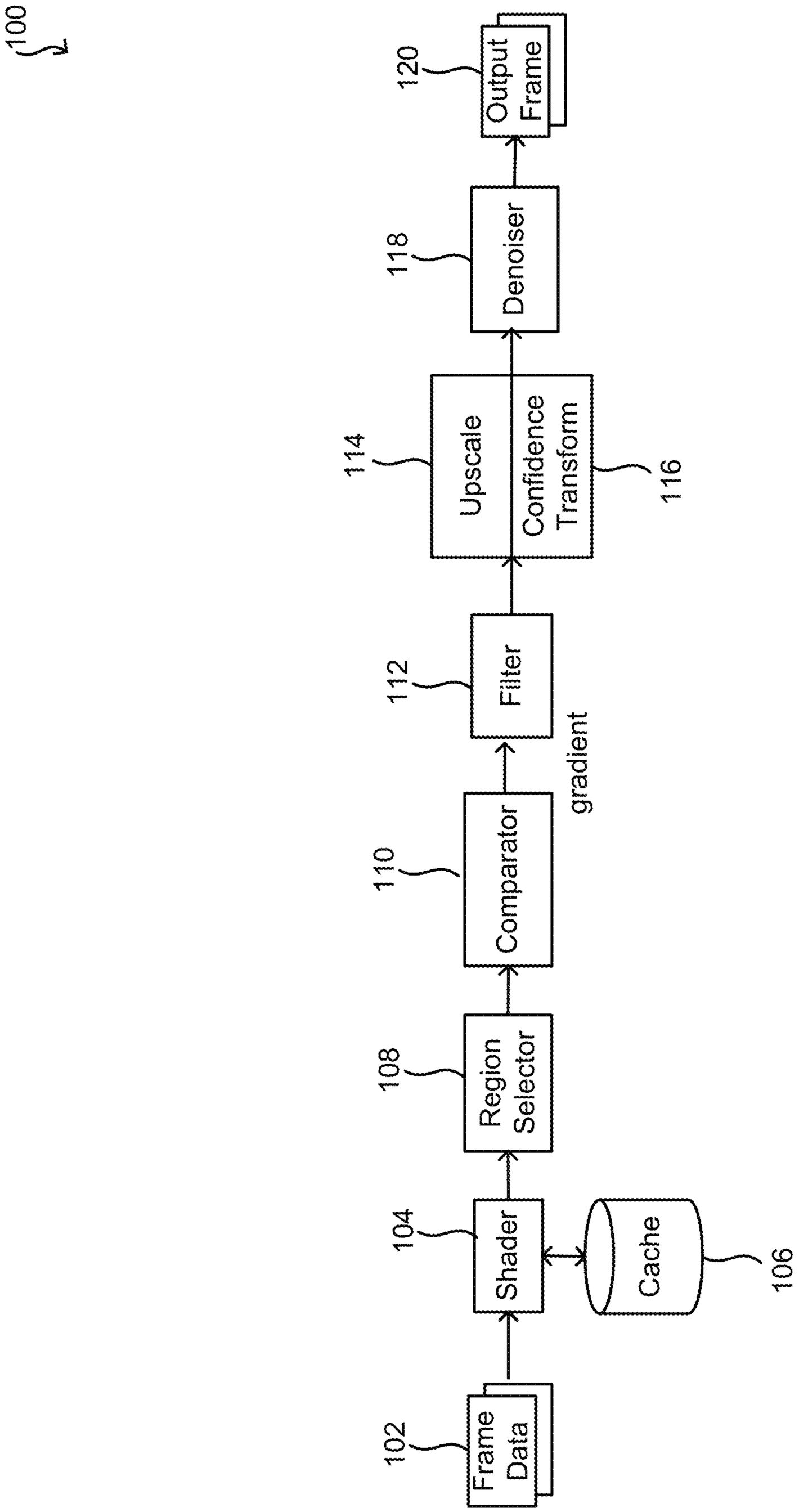

100
120
Output Frame
118
Denoiser
114
Upscale
Confidence Transform
116
112
Filter
gradient
110
Comparator
108
Region Selector
104
Shader
Cache
106
102
Frame Data

FIG. 1

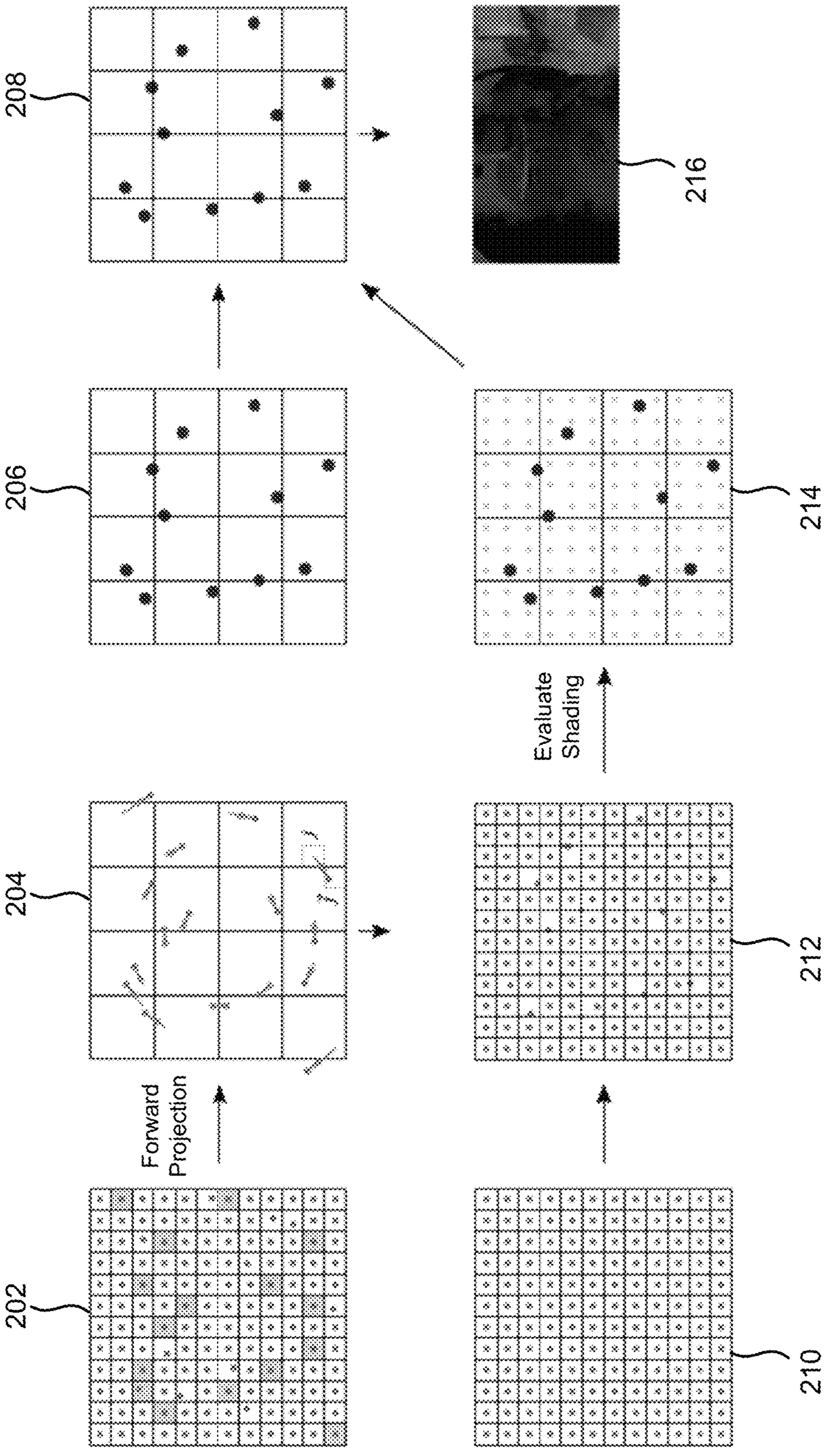

208
216
206
214
204
212
Evaluate Shading
Forward Projection
202
210

FIG. 2

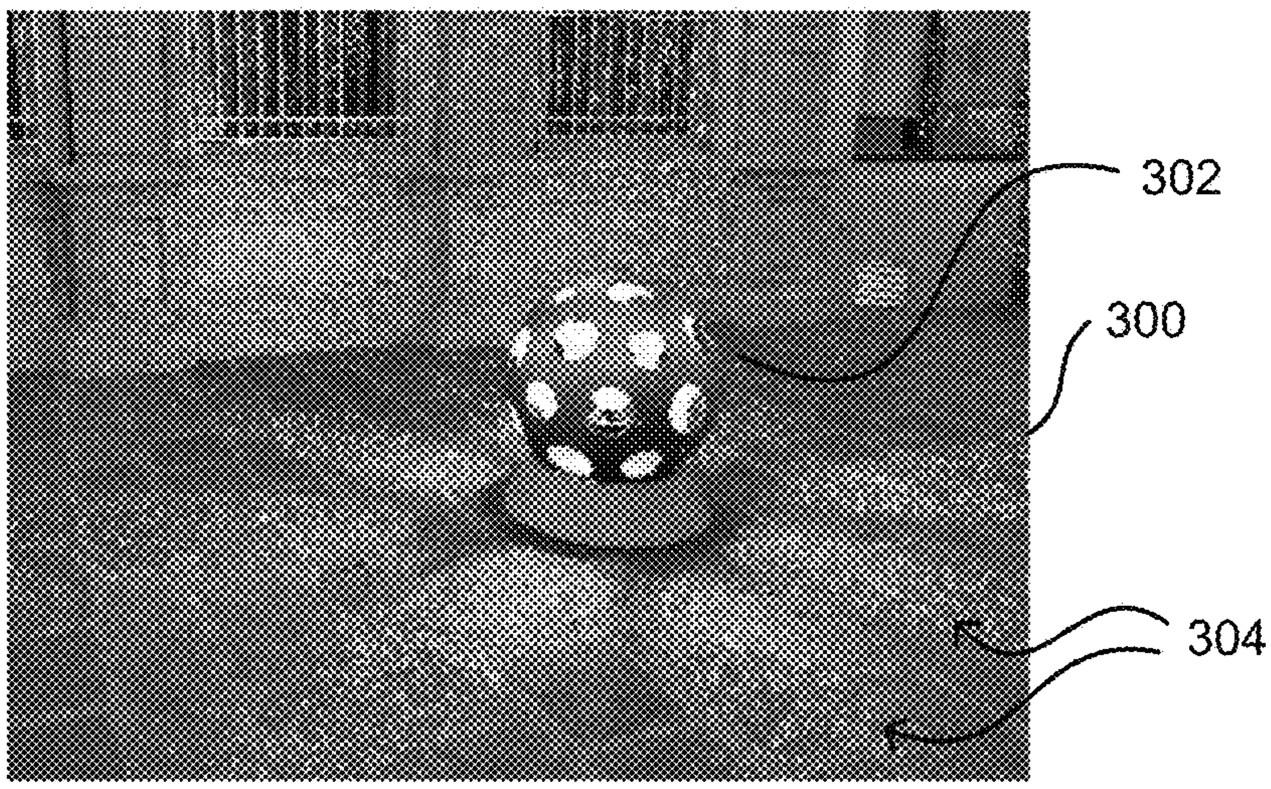


FIG. 3A

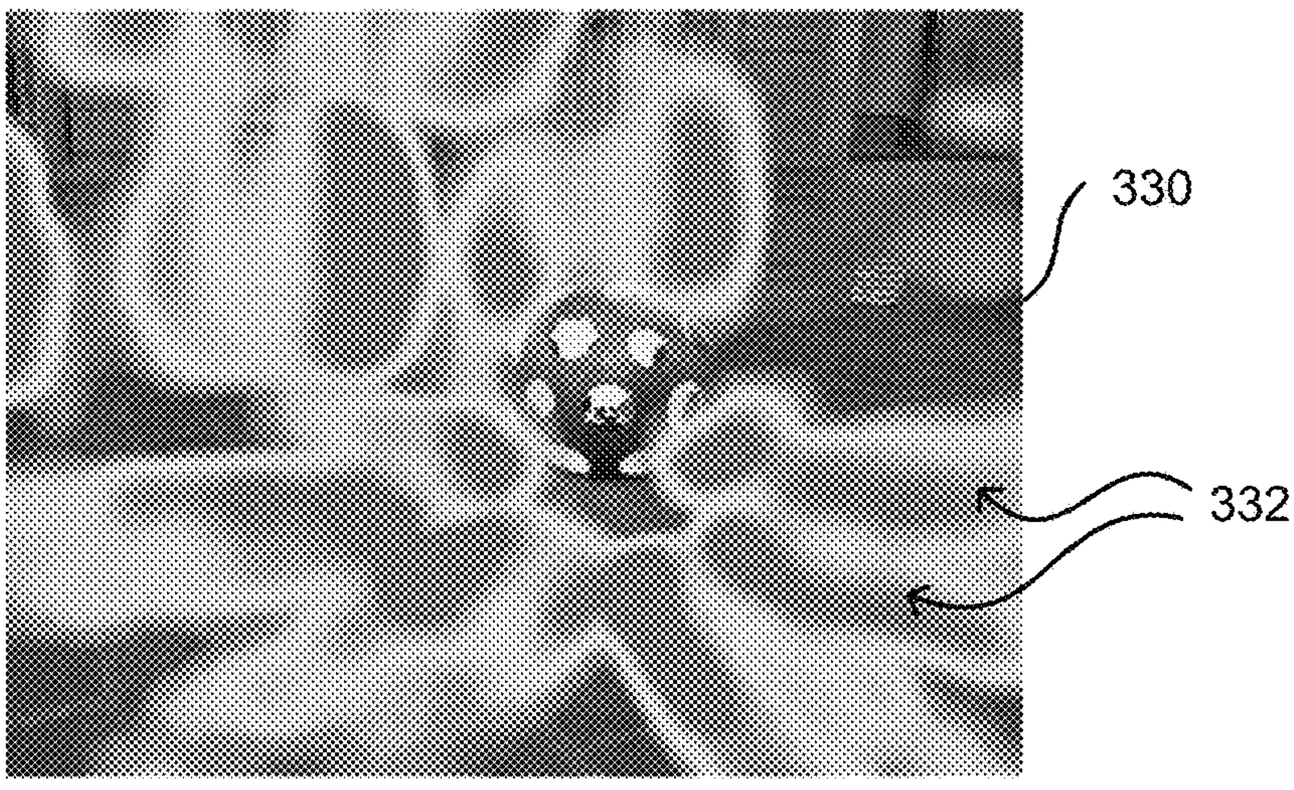


FIG. 3B

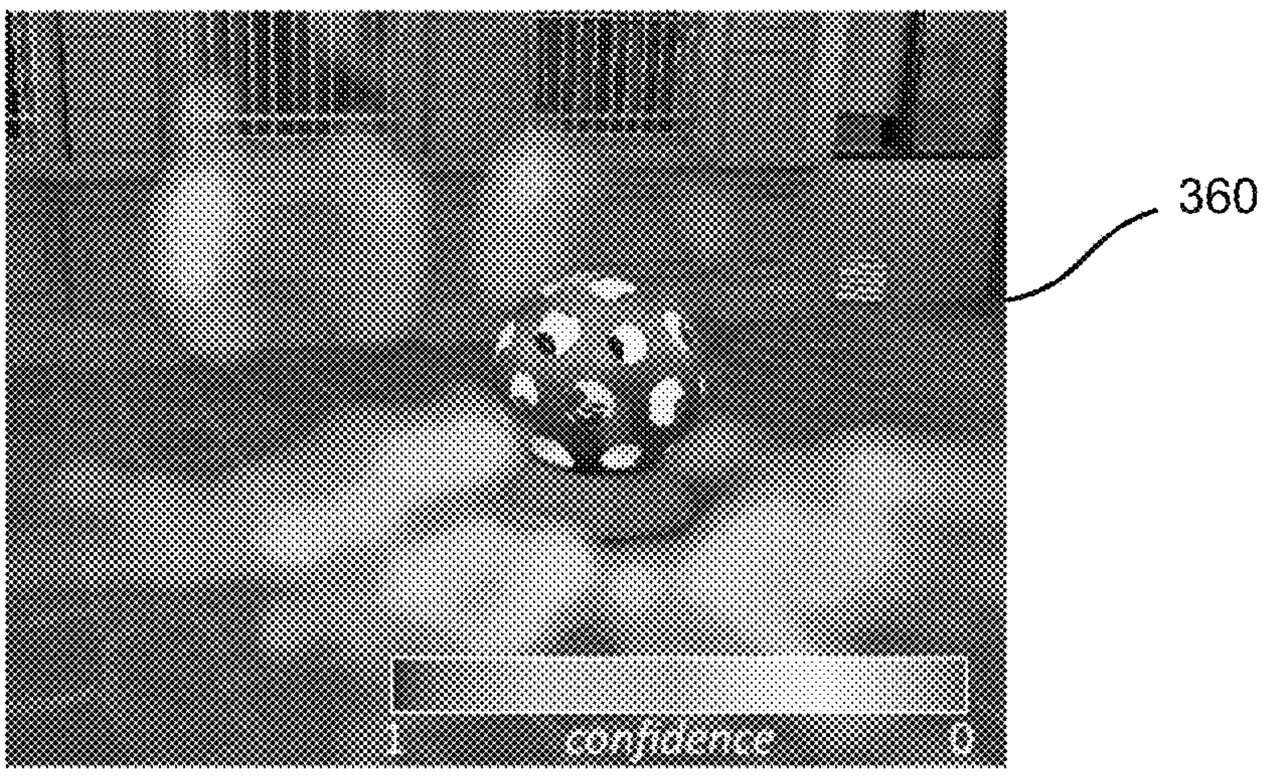


FIG. 3C

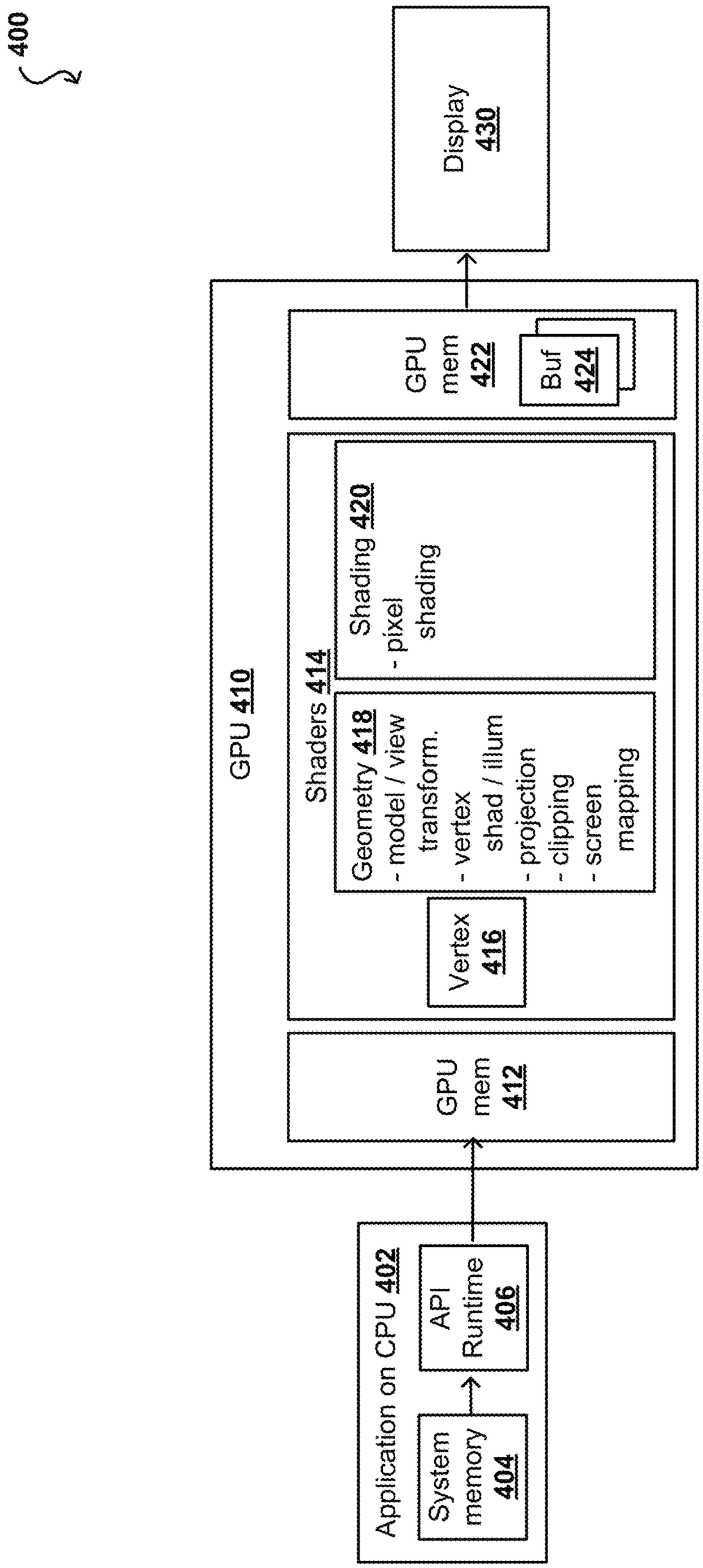

400
Display 430
GPU 410
Shaders 414
Geometry 418
- model / view transform.
- vertex shad / illum
- projection
- clipping
- screen mapping
Vertex 416
Shading 420
- pixel shading
GPU mem 422
Buf 424
GPU mem 412
Application on CPU 402
System memory 404
API Runtime 406

FIG. 4

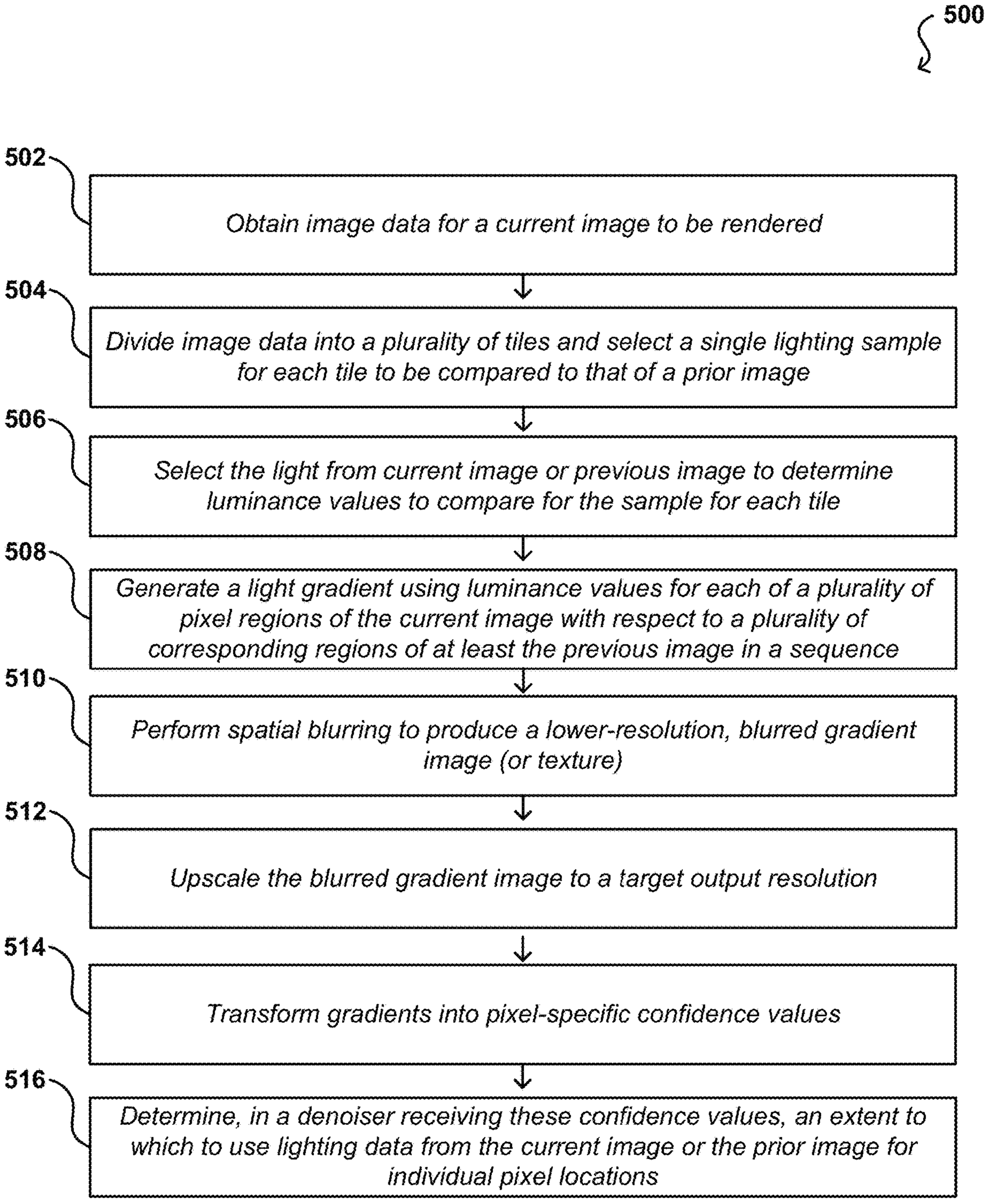

500
502
Obtain image data for a current image to be rendered
504
Divide image data into a plurality of tiles and select a single lighting sample for each tile to be compared to that of a prior image
506
Select the light from current image or previous image to determine luminance values to compare for the sample for each tile
508
Generate a light gradient using luminance values for each of a plurality of pixel regions of the current image with respect to a plurality of corresponding regions of at least the previous image in a sequence
510
Perform spatial blurring to produce a lower-resolution, blurred gradient image (or texture)
512
Upscale the blurred gradient image to a target output resolution
514
Transform gradients into pixel-specific confidence values
516
Determine, in a denoiser receiving these confidence values, an extent to which to use lighting data from the current image or the prior image for individual pixel locations

FIG. 5

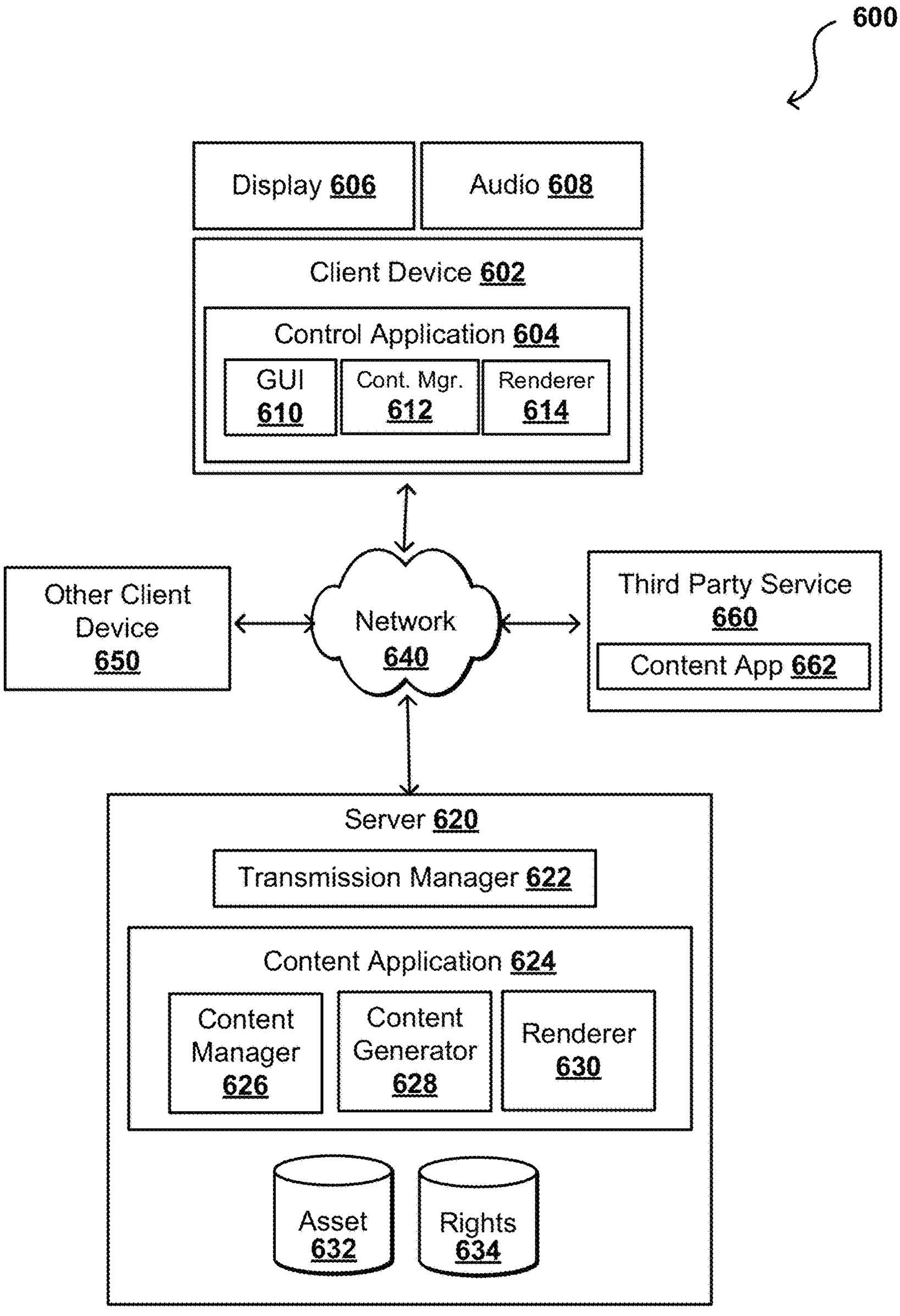

600
Display 606
Audio 608
Client Device 602
Control Application 604
GUI 610
Cont. Mgr. 612
Renderer 614
Other Client Device 650
Network 640
Third Party Service 660
Content App 662
Server 620
Transmission Manager 622
Content Application 624
Content Manager 626
Content Generator 628
Renderer 630
Asset 632
Rights 634

FIG. 6

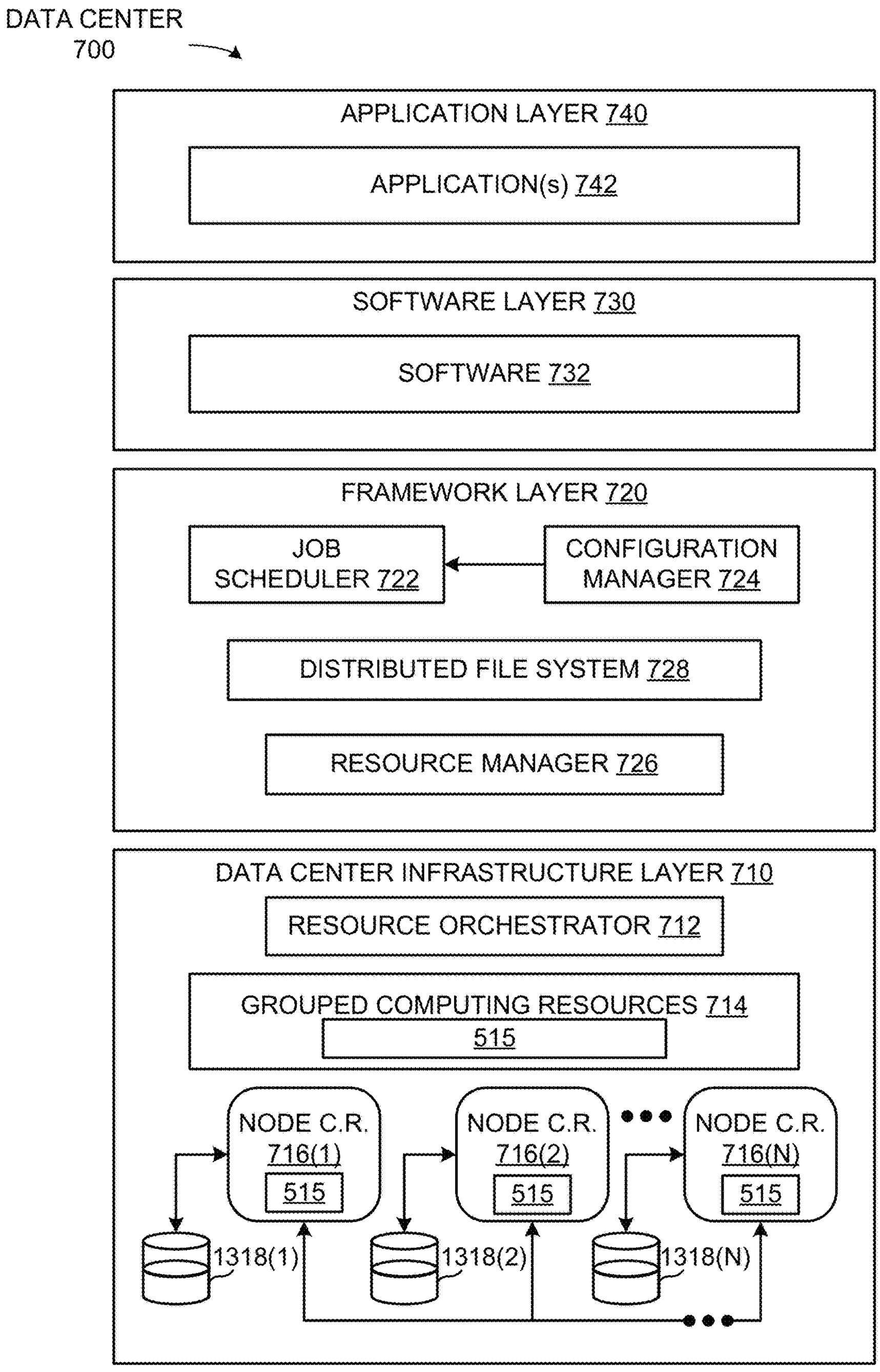

DATA CENTER
700
APPLICATION LAYER 740
APPLICATION(s) 742
SOFTWARE LAYER 730
SOFTWARE 732
FRAMEWORK LAYER 720
JOB SCHEDULER 722
CONFIGURATION MANAGER 724
DISTRIBUTED FILE SYSTEM 728
RESOURCE MANAGER 726
DATA CENTER INFRASTRUCTURE LAYER 710
RESOURCE ORCHESTRATOR 712
GROUPED COMPUTING RESOURCES 714
515
NODE C.R. 716(1)
515
NODE C.R. 716(2)
515
NODE C.R. 716(N)
515
1318(1)
1318(2)
1318(N)

FIG. 7

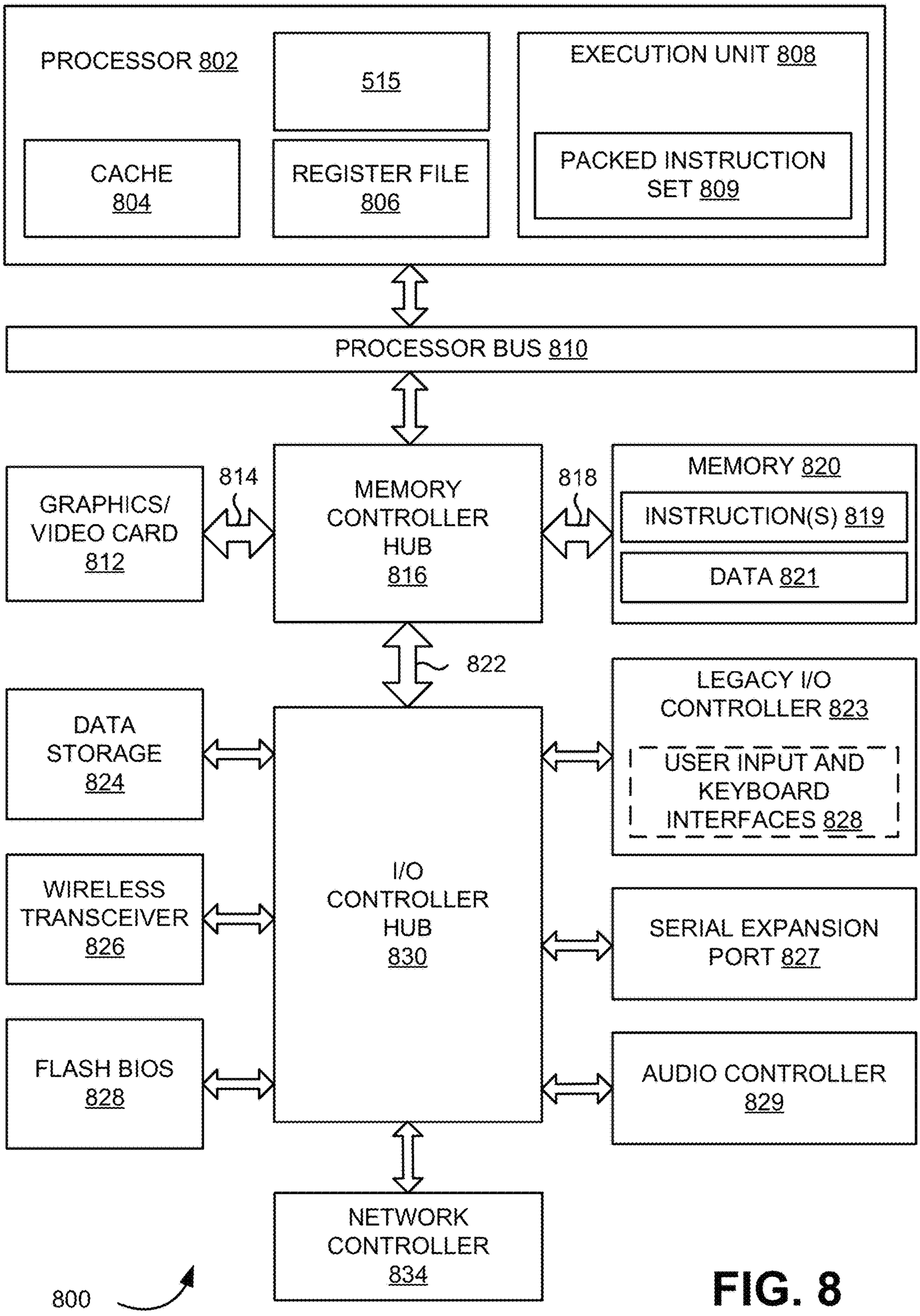

PROCESSOR 802
515
EXECUTION UNIT 808
CACHE 804
REGISTER FILE 806
PACKED INSTRUCTION SET 809
PROCESSOR BUS 810
GRAPHICS/ VIDEO CARD 812
814
MEMORY CONTROLLER HUB 816
818
MEMORY 820
INSTRUCTION(S) 819
DATA 821
822
DATA STORAGE 824
LEGACY I/O CONTROLLER 823
USER INPUT AND KEYBOARD INTERFACES 828
WIRELESS TRANSCEIVER 826
I/O CONTROLLER HUB 830
SERIAL EXPANSION PORT 827
FLASH BIOS 828
AUDIO CONTROLLER 829
NETWORK CONTROLLER 834
800

FIG. 8

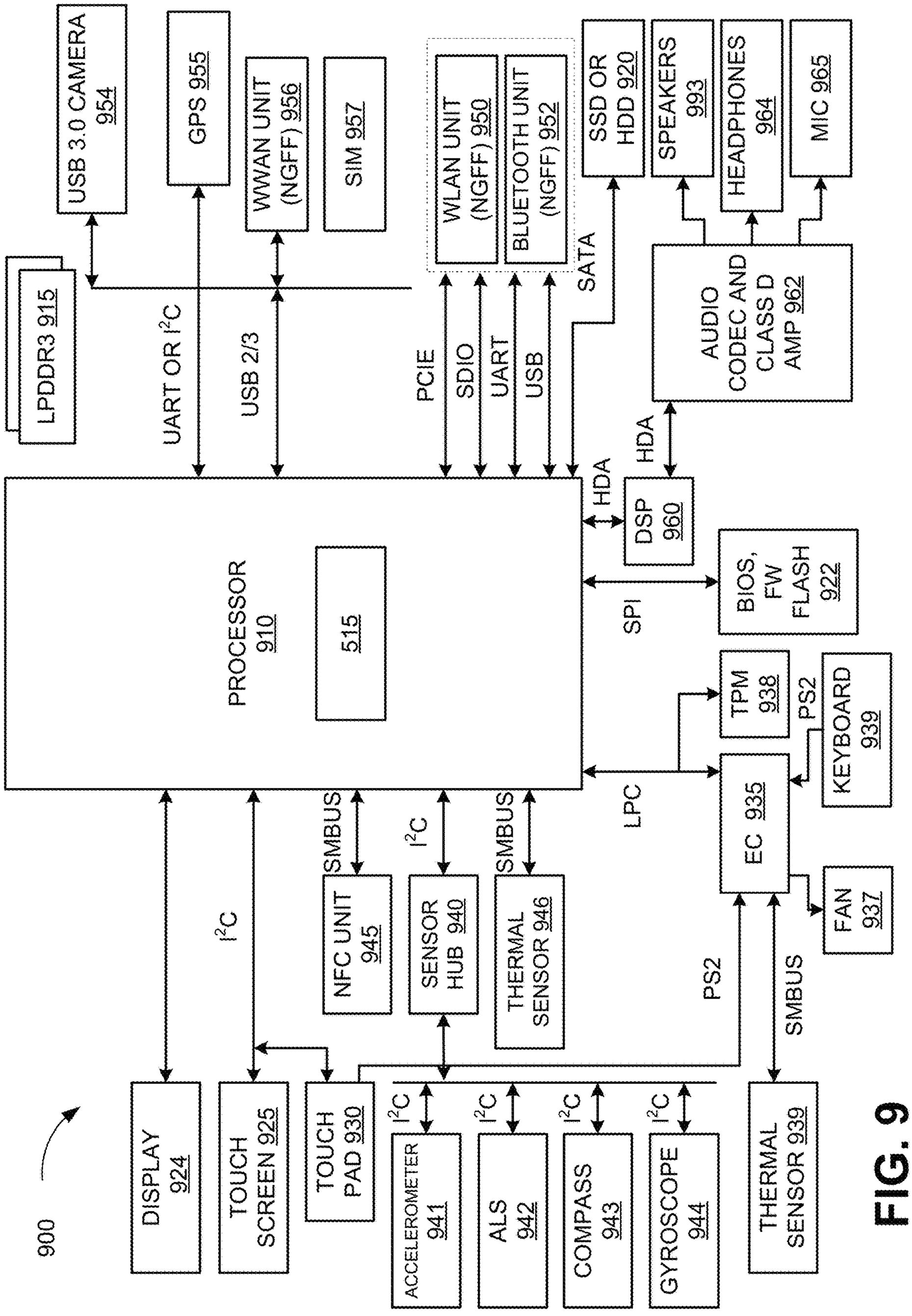

900
DISPLAY 924
TOUCH SCREEN 925
TOUCH PAD 930
ACCELEROMETER 941
ALS 942
COMPASS 943
GYROSCOPE 944
THERMAL SENSOR 939
FAN 937
NFC UNIT 945
SENSOR HUB 940
THERMAL SENSOR 946
EC 935
KEYBOARD 939
TPM 938
BIOS, FW FLASH 922
DSP 960
AUDIO CODEC AND CLASS D AMP 962
SPEAKERS 993
HEADPHONES 964
MIC 965
SSD OR HDD 920
WLAN UNIT (NGFF) 950
BLUETOOTH UNIT (NGFF) 952
WWAN UNIT (NGFF) 956
SIM 957
GPS 955
USB 3.0 CAMERA 954
LPDDR3 915
PROCESSOR 910
515
UART OR I2C
USB 2/3
PCIE
SDIO
UART
USB
SATA
HDA
SPI
LPC
PS2
SMBUS
I2C

FIG. 9

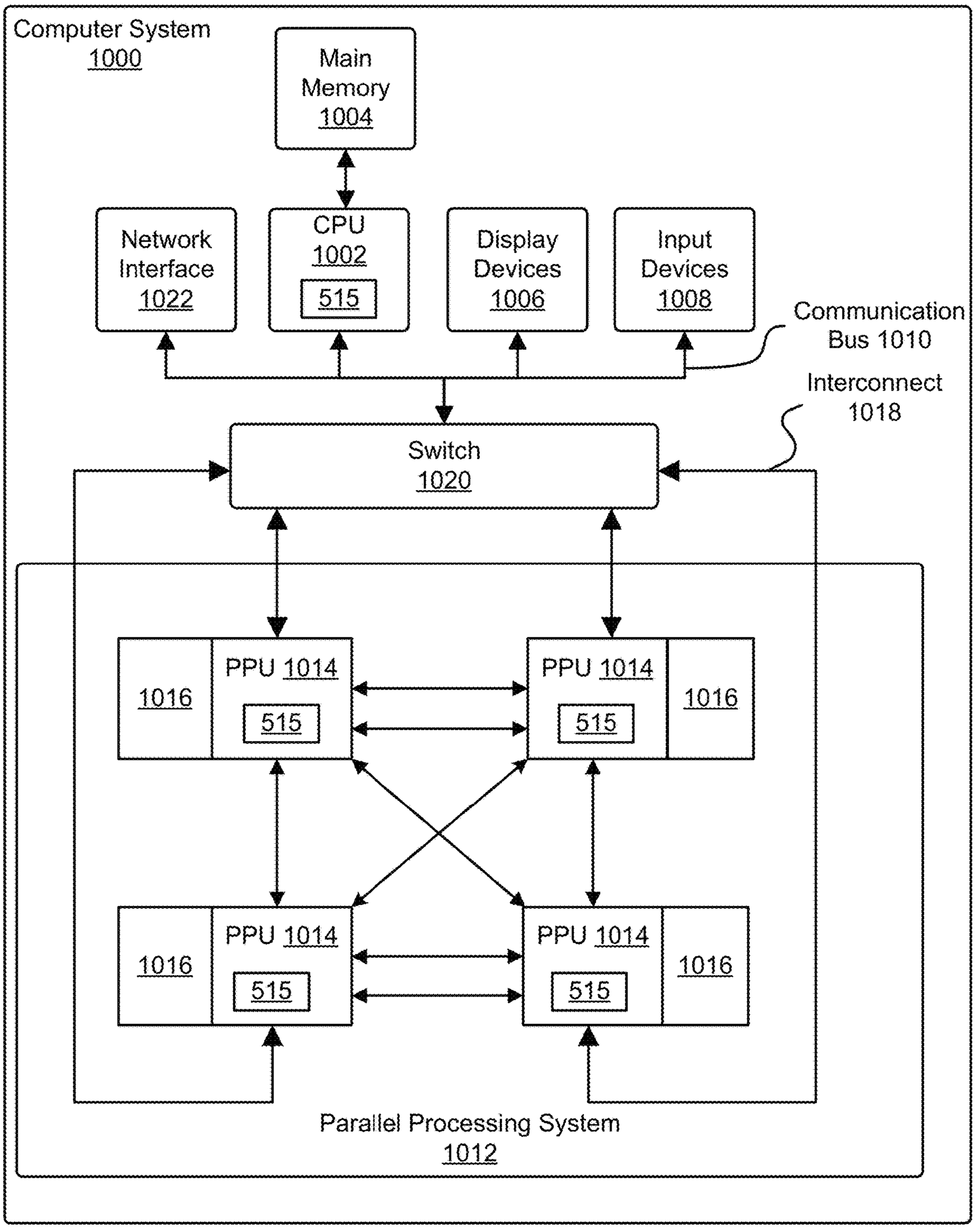

Computer System
1000
Main
Memory
1004
Network
Interface
1022
CPU
1002
515
Display
Devices
1006
Input
Devices
1008
Communication
Bus 1010
Interconnect
1018
Switch
1020
1016
PPU 1014
515
PPU 1014
515
1016
1016
PPU 1014
515
PPU 1014
515
1016
Parallel Processing System
1012

FIG. 10

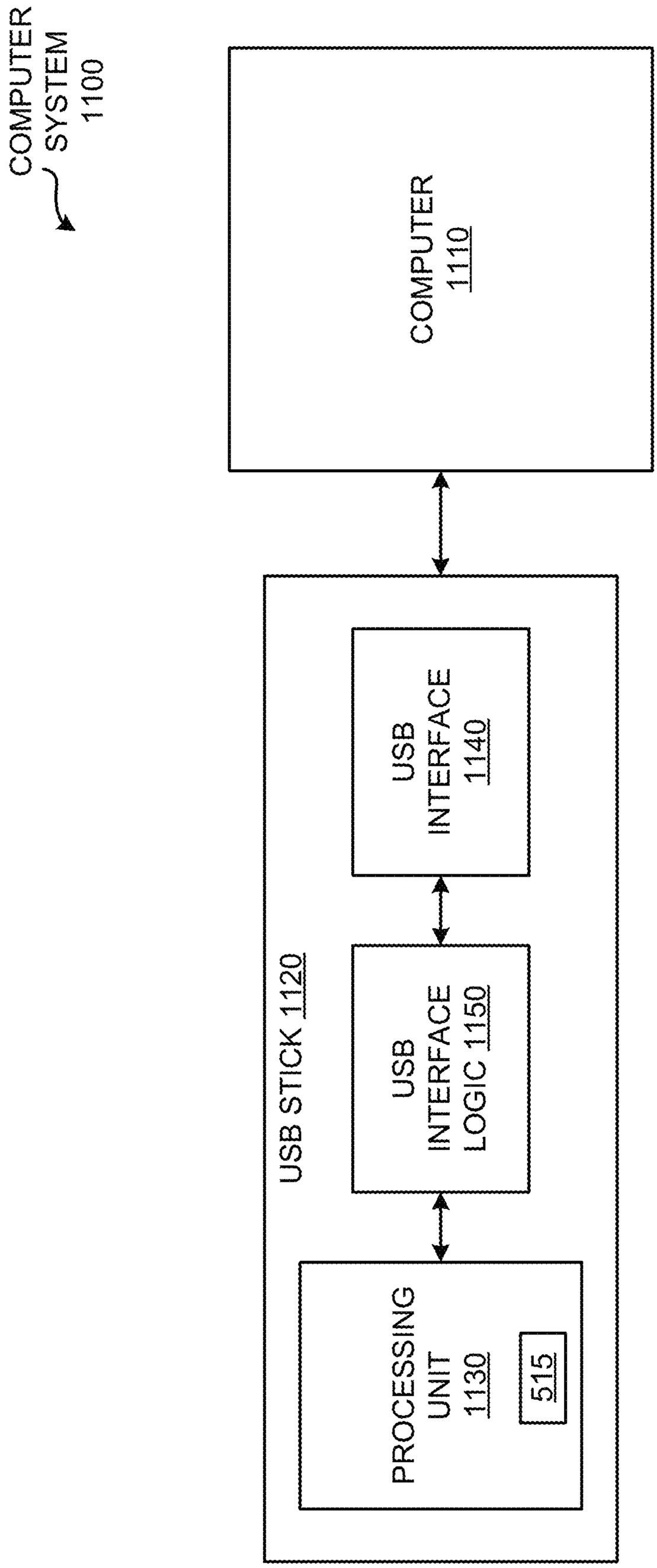

COMPUTER SYSTEM 1100
COMPUTER 1110
USB STICK 1120
USB INTERFACE 1140
USB INTERFACE LOGIC 1150
PROCESSING UNIT 1130
515

FIG. 11

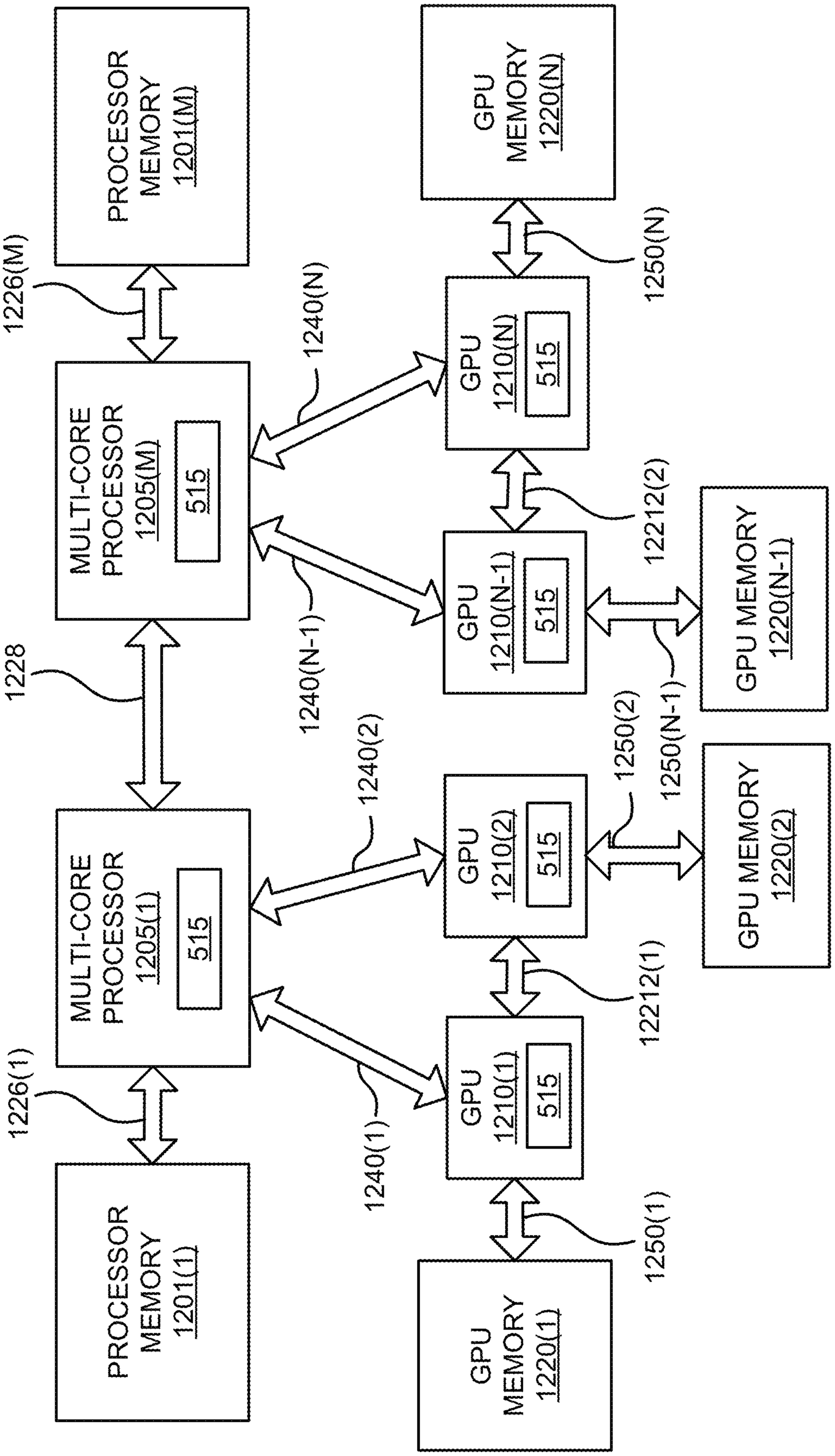

PROCESSOR MEMORY 1201(M)
1226(M)
MULTI-CORE PROCESSOR 1205(M)
515
1228
GPU MEMORY 1220(N)
1250(N)
1240(N)
GPU 1210(N)
515
1240(N-1)
GPU 1210(N-1)
515
12212(2)
GPU MEMORY 1220(N-1)
1250(2)
1250(N-1)
1240(2)
MULTI-CORE PROCESSOR 1205(1)
515
GPU 1210(2)
515
GPU MEMORY 1220(2)
12212(1)
1226(1)
1240(1)
GPU 1210(1)
515
PROCESSOR MEMORY 1201(1)
GPU MEMORY 1220(1)
1250(1)

FIG. 12A

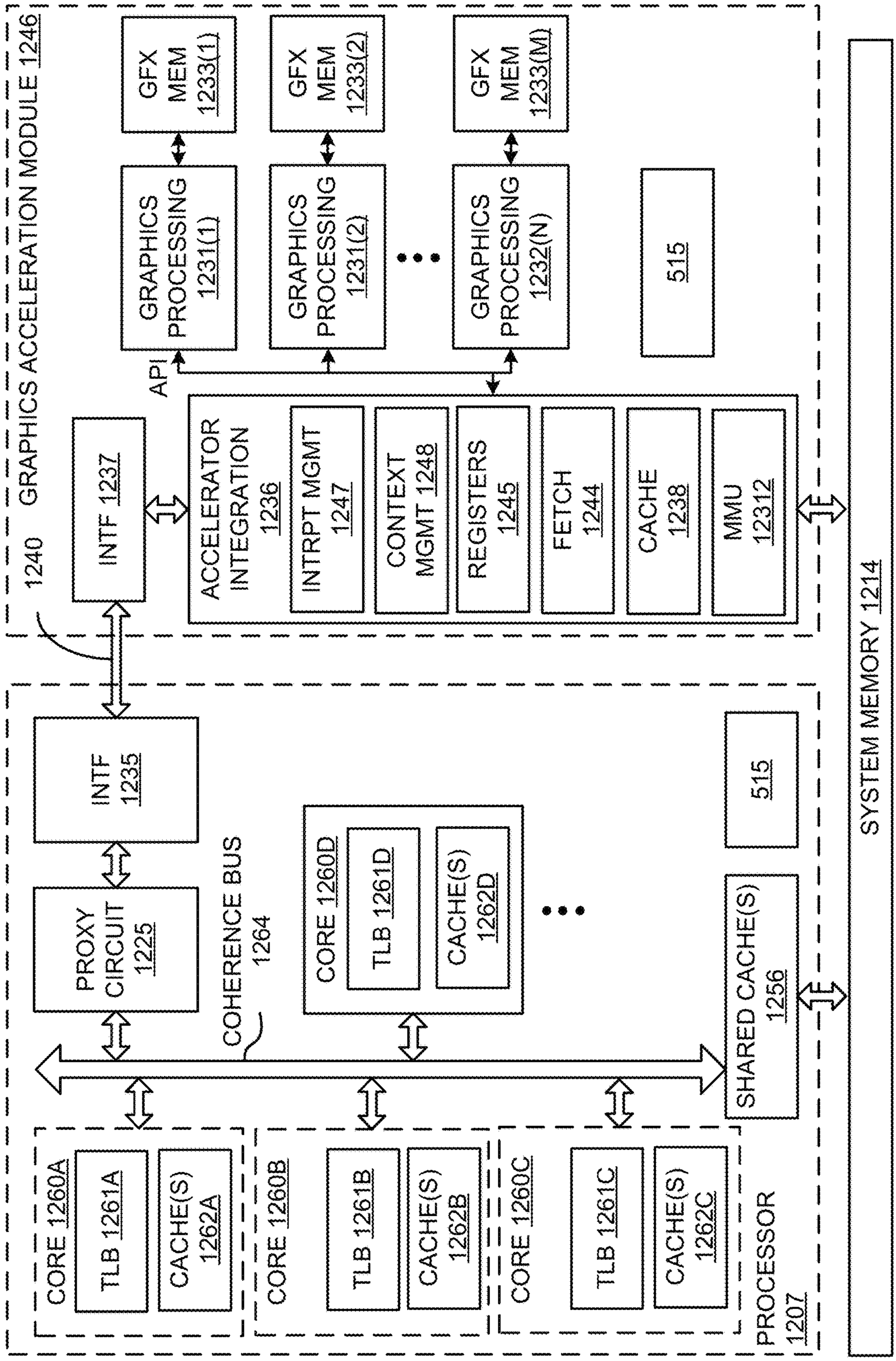

GRAPHICS ACCELERATION MODULE 1246
GFX MEM 1233(1)
GFX MEM 1233(2)
GFX MEM 1233(M)
GRAPHICS PROCESSING 1231(1)
GRAPHICS PROCESSING 1231(2)
GRAPHICS PROCESSING 1232(N)
515
API
1240
INTF 1237
ACCELERATOR INTEGRATION 1236
INTRPT MGMT 1247
CONTEXT MGMT 1248
REGISTERS 1245
FETCH 1244
CACHE 1238
MMU 12312
INTF 1235
PROXY CIRCUIT 1225
COHERENCE BUS 1264
CORE 1260D
TLB 1261D
CACHE(S) 1262D
515
SHARED CACHE(S) 1256
CORE 1260A
TLB 1261A
CACHE(S) 1262A
CORE 1260B
TLB 1261B
CACHE(S) 1262B
CORE 1260C
TLB 1261C
CACHE(S) 1262C
PROCESSOR 1207
SYSTEM MEMORY 1214

FIG. 12 B

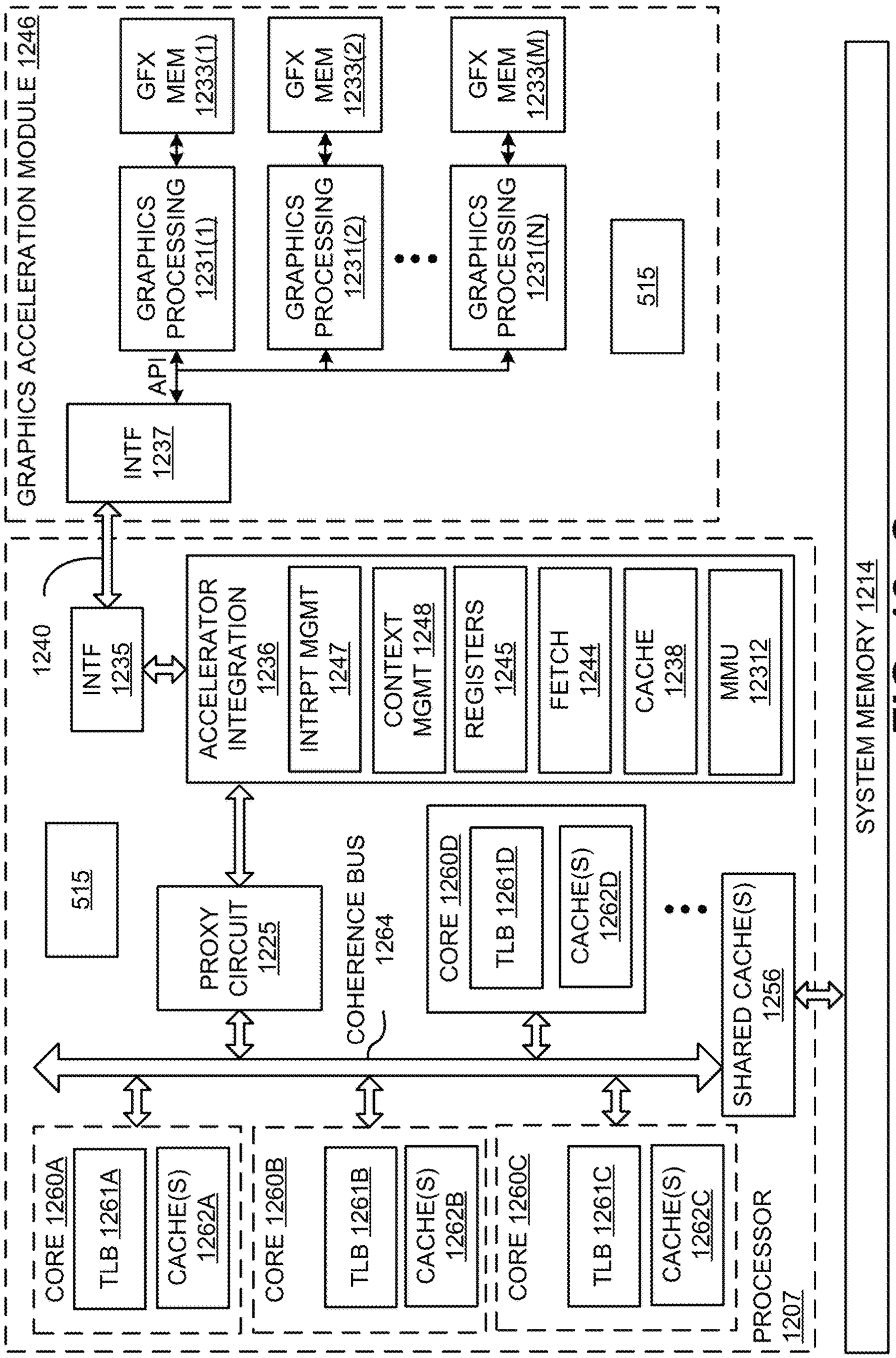

SYSTEM MEMORY 1214
GRAPHICS ACCELERATION MODULE 1246
GRAPHICS PROCESSING 1231(1)
GRAPHICS PROCESSING 1231(2)
GRAPHICS PROCESSING 1231(N)
GFX MEM 1233(1)
GFX MEM 1233(2)
GFX MEM 1233(M)
515
INTF 1237
APP
1240
INTF 1235
ACCELERATOR INTEGRATION 1236
INTRPT MGMT 1247
CONTEXT MGMT 1248
REGISTERS 1245
FETCH 1244
CACHE 1238
MMU 12312
515
PROXY CIRCUIT 1225
COHERENCE BUS 1264
CORE 1260D
TLB 1261D
CACHE(S) 1262D
SHARED CACHE(S) 1256
CORE 1260A
TLB 1261A
CACHE(S) 1262A
CORE 1260B
TLB 1261B
CACHE(S) 1262B
CORE 1260C
TLB 1261C
CACHE(S) 1262C
PROCESSOR 1207

FIG. 12 C

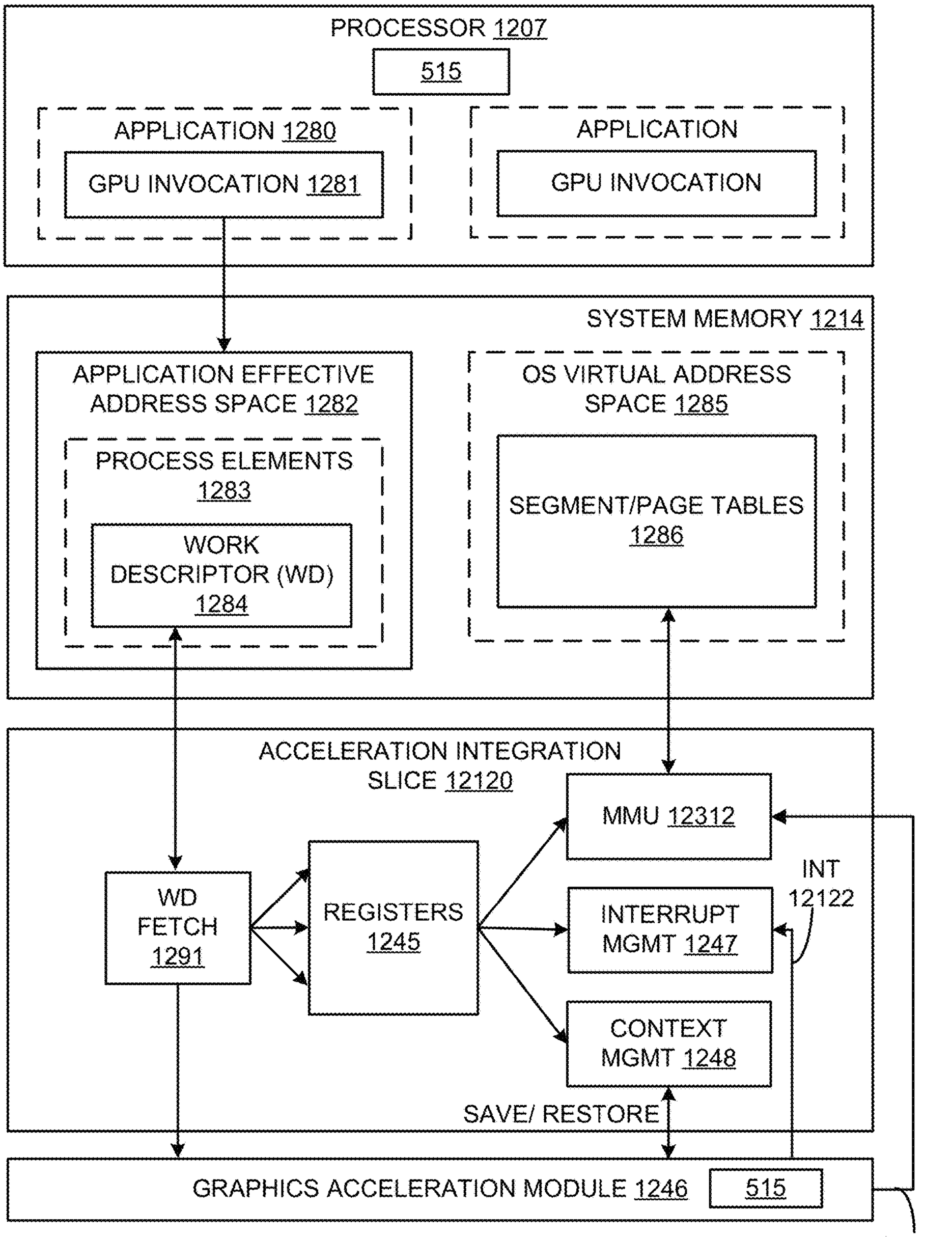

PROCESSOR 1207
515
APPLICATION 1280
GPU INVOCATION 1281
APPLICATION
GPU INVOCATION
SYSTEM MEMORY 1214
APPLICATION EFFECTIVE ADDRESS SPACE 1282
PROCESS ELEMENTS 1283
WORK DESCRIPTOR (WD) 1284
OS VIRTUAL ADDRESS SPACE 1285
SEGMENT/PAGE TABLES 1286
ACCELERATION INTEGRATION SLICE 12120
MMU 12312
INT 12122
WD FETCH 1291
REGISTERS 1245
INTERRUPT MGMT 1247
CONTEXT MGMT 1248
SAVE/ RESTORE
GRAPHICS ACCELERATION MODULE 1246
515
EFFECTIVE ADDRESS 1293

FIG. 12 D

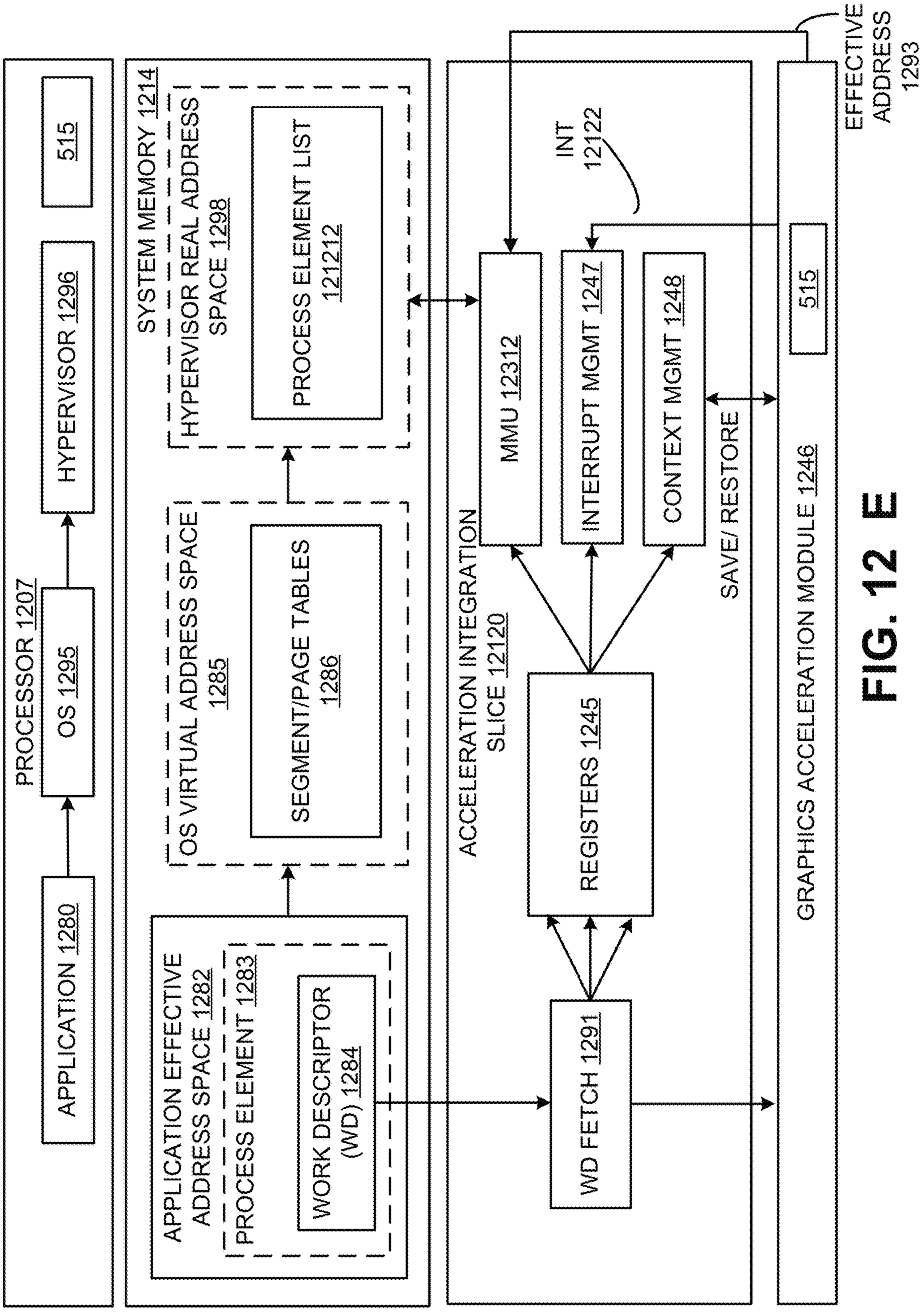

APPLICATION 1280
PROCESSOR 1207
OS 1295
HYPERVISOR 1296
515
SYSTEM MEMORY 1214
HYPERVISOR REAL ADDRESS SPACE 1298
PROCESS ELEMENT LIST 121212
OS VIRTUAL ADDRESS SPACE 1285
SEGMENT/PAGE TABLES 1286
APPLICATION EFFECTIVE ADDRESS SPACE 1282
PROCESS ELEMENT 1283
WORK DESCRIPTOR (WD) 1284
ACCELERATION INTEGRATION SLICE 12120
MMU 12312
INTERRUPT MGMT 1247
CONTEXT MGMT 1248
REGISTERS 1245
SAVE/ RESTORE
WD FETCH 1291
INT 12122
EFFECTIVE ADDRESS 1293
515
GRAPHICS ACCELERATION MODULE 1246

FIG. 12 E

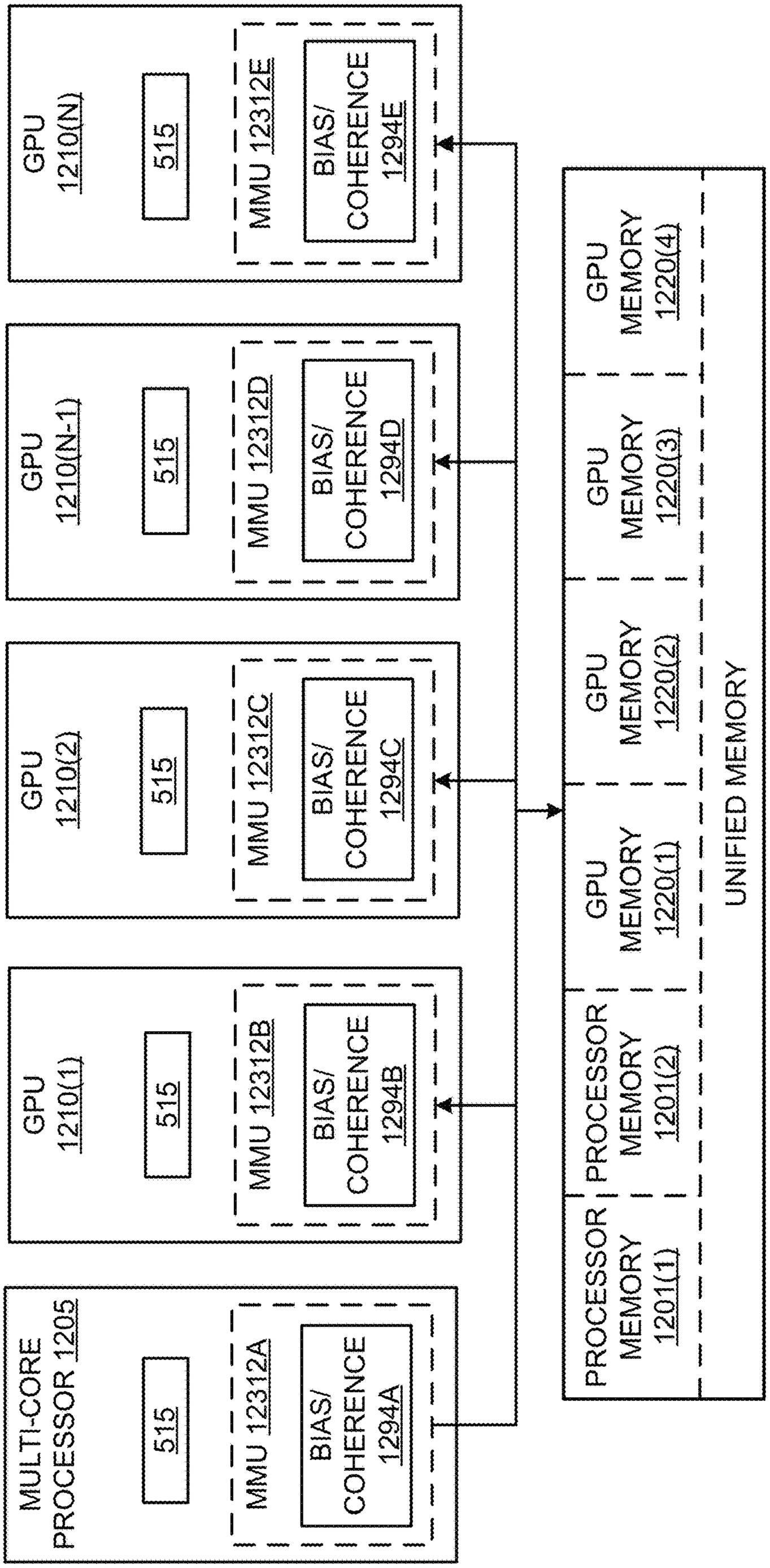

GPU 1210(N)
515
MMU 12312E
BIAS/ COHERENCE 1294E
GPU 1210(N-1)
515
MMU 12312D
BIAS/ COHERENCE 1294D
GPU 1210(2)
515
MMU 12312C
BIAS/ COHERENCE 1294C
GPU 1210(1)
515
MMU 12312B
BIAS/ COHERENCE 1294B
MULTI-CORE PROCESSOR 1205
515
MMU 12312A
BIAS/ COHERENCE 1294A
GPU MEMORY 1220(4)
GPU MEMORY 1220(3)
GPU MEMORY 1220(2)
GPU MEMORY 1220(1)
UNIFIED MEMORY
PROCESSOR MEMORY 1201(2)
PROCESSOR MEMORY 1201(1)

FIG. 12 F

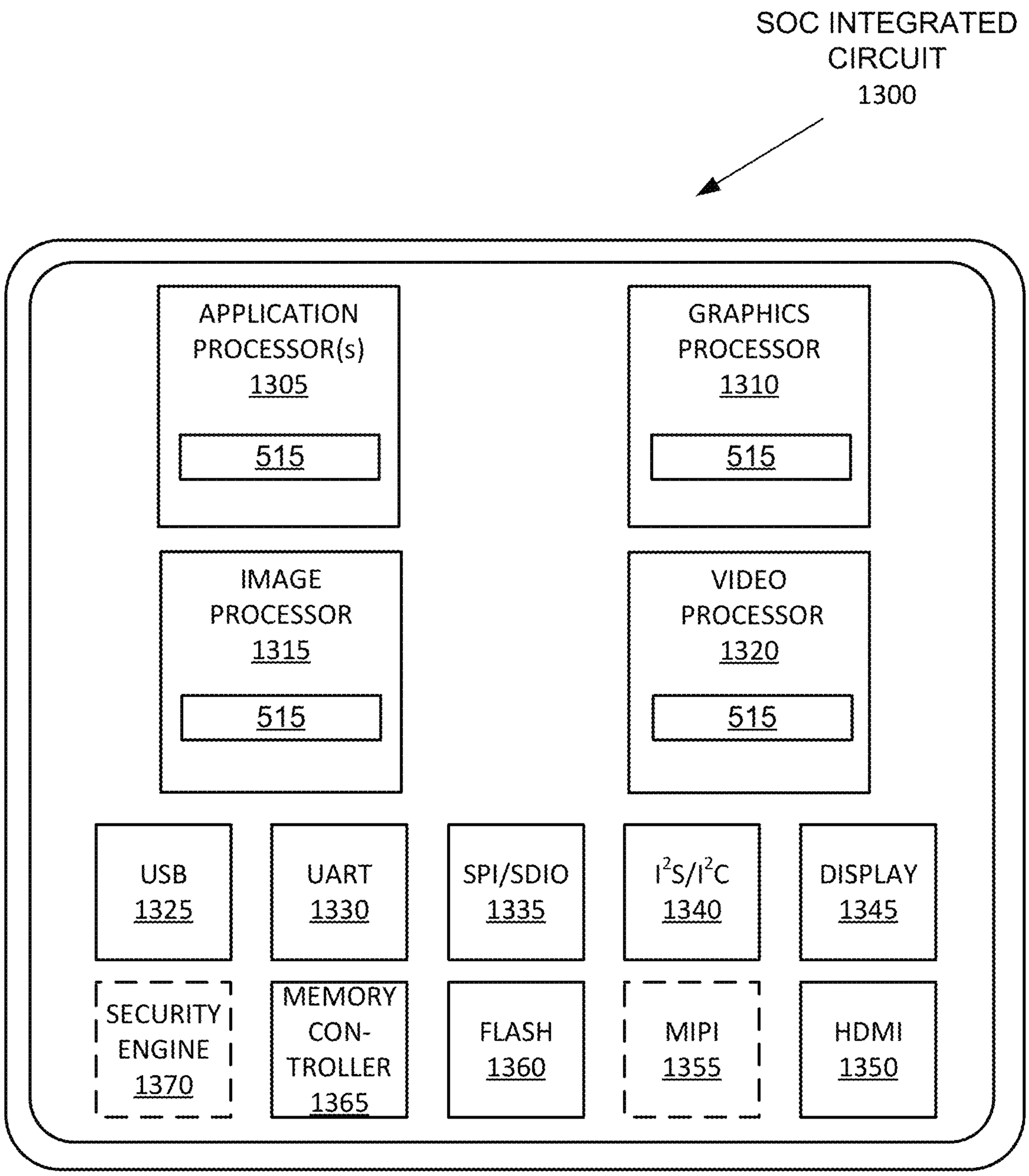

SOC INTEGRATED CIRCUIT 1300
APPLICATION PROCESSOR(s) 1305
515
GRAPHICS PROCESSOR 1310
515
IMAGE PROCESSOR 1315
515
VIDEO PROCESSOR 1320
515
USB 1325
UART 1330
SPI/SDIO 1335
I2S/I2C 1340
DISPLAY 1345
SECURITY ENGINE 1370
MEMORY CON-TROLLER 1365
FLASH 1360
MIPI 1355
HDMI 1350

FIG. 13

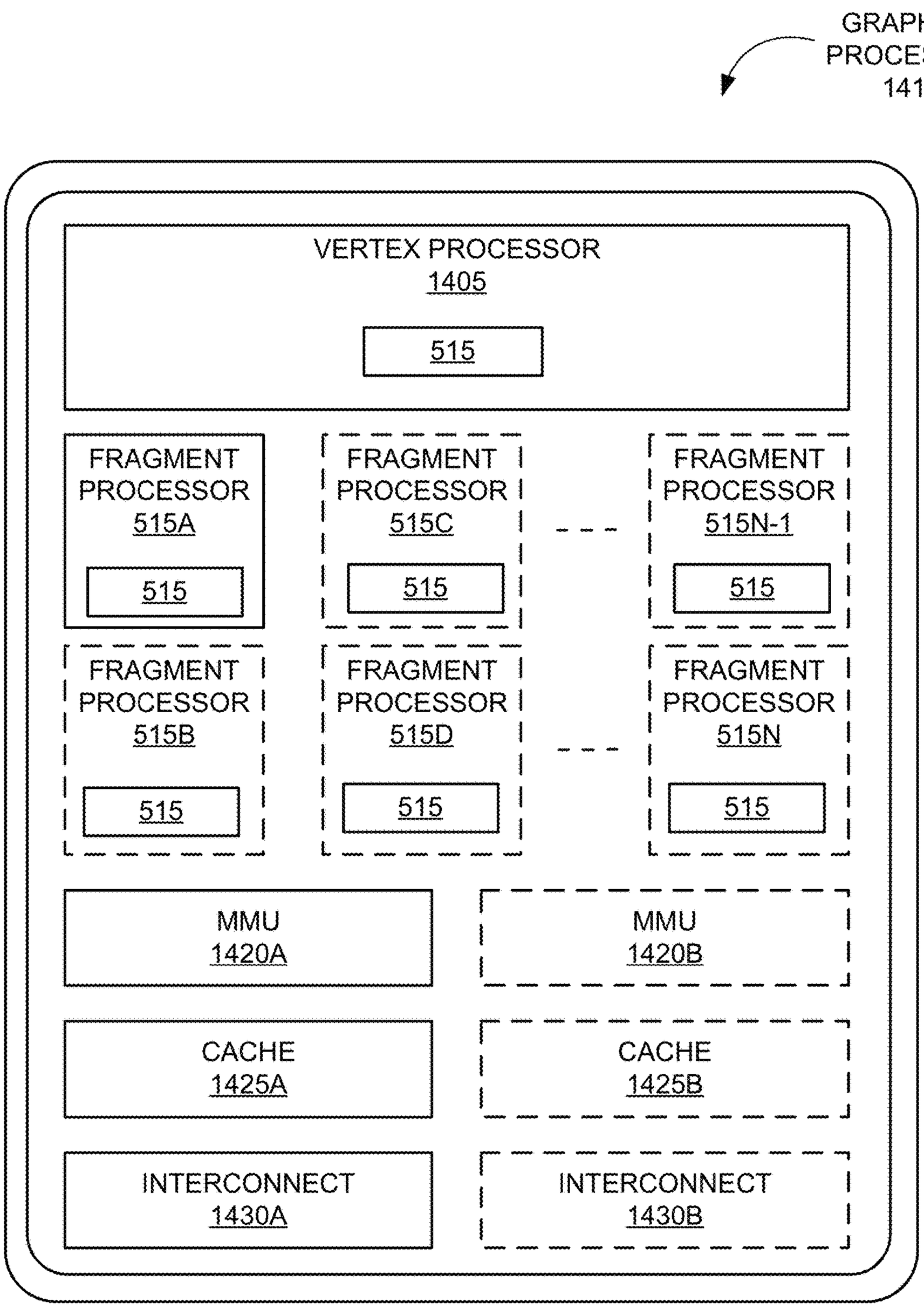

GRAPHICS PROCESSOR 1410
VERTEX PROCESSOR 1405
515
FRAGMENT PROCESSOR 515A
515
FRAGMENT PROCESSOR 515C
515
FRAGMENT PROCESSOR 515N-1
515
FRAGMENT PROCESSOR 515B
515
FRAGMENT PROCESSOR 515D
515
FRAGMENT PROCESSOR 515N
515
MMU 1420A
MMU 1420B
CACHE 1425A
CACHE 1425B
INTERCONNECT 1430A
INTERCONNECT 1430B

FIG. 14A

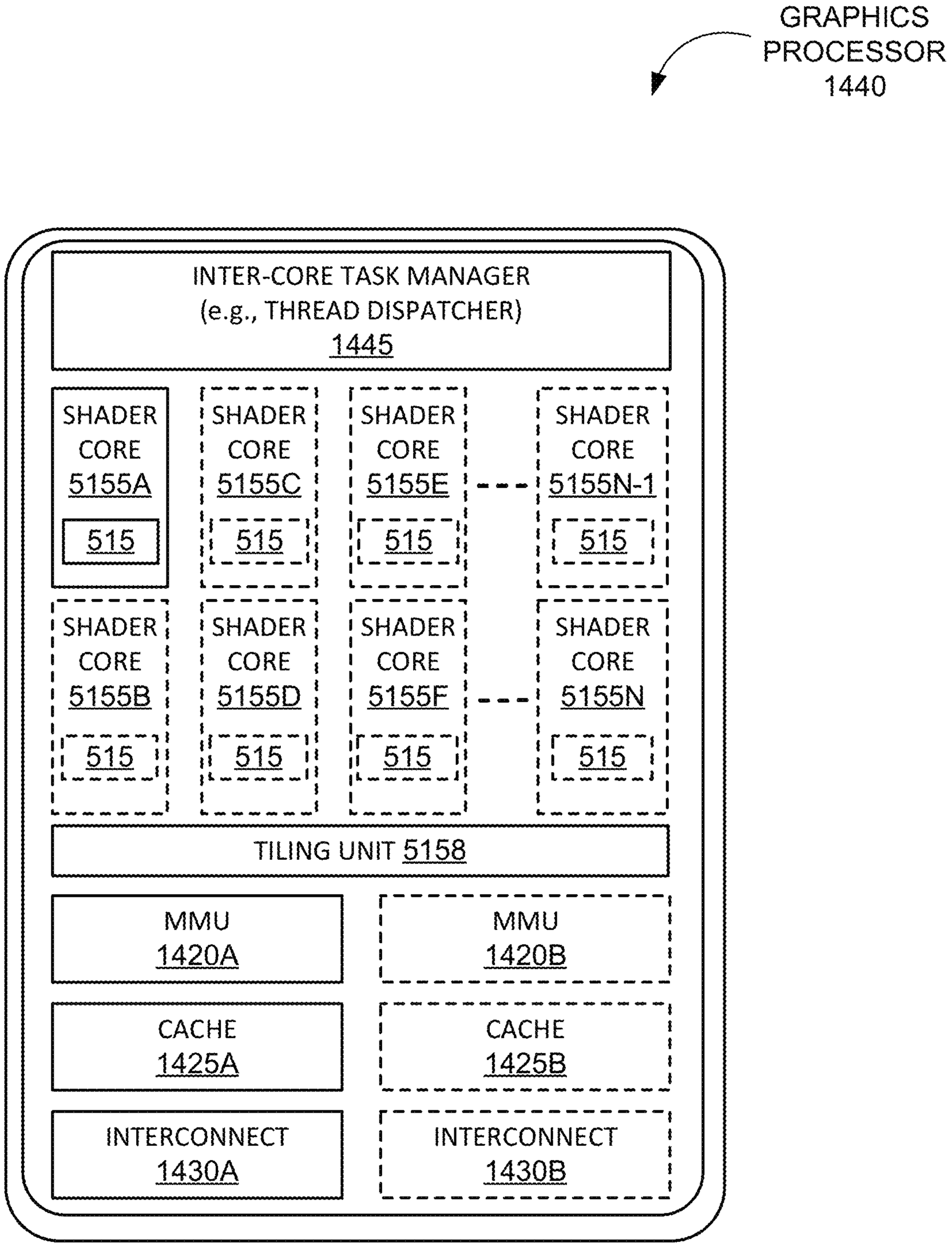

GRAPHICS PROCESSOR 1440
INTER-CORE TASK MANAGER (e.g., THREAD DISPATCHER) 1445
SHADER CORE 5155A
515
SHADER CORE 5155C
515
SHADER CORE 5155E
515
SHADER CORE 5155N-1
515
SHADER CORE 5155B
515
SHADER CORE 5155D
515
SHADER CORE 5155F
515
SHADER CORE 5155N
515
TILING UNIT 5158
MMU 1420A
MMU 1420B
CACHE 1425A
CACHE 1425B
INTERCONNECT 1430A
INTERCONNECT 1430B

FIG. 14B

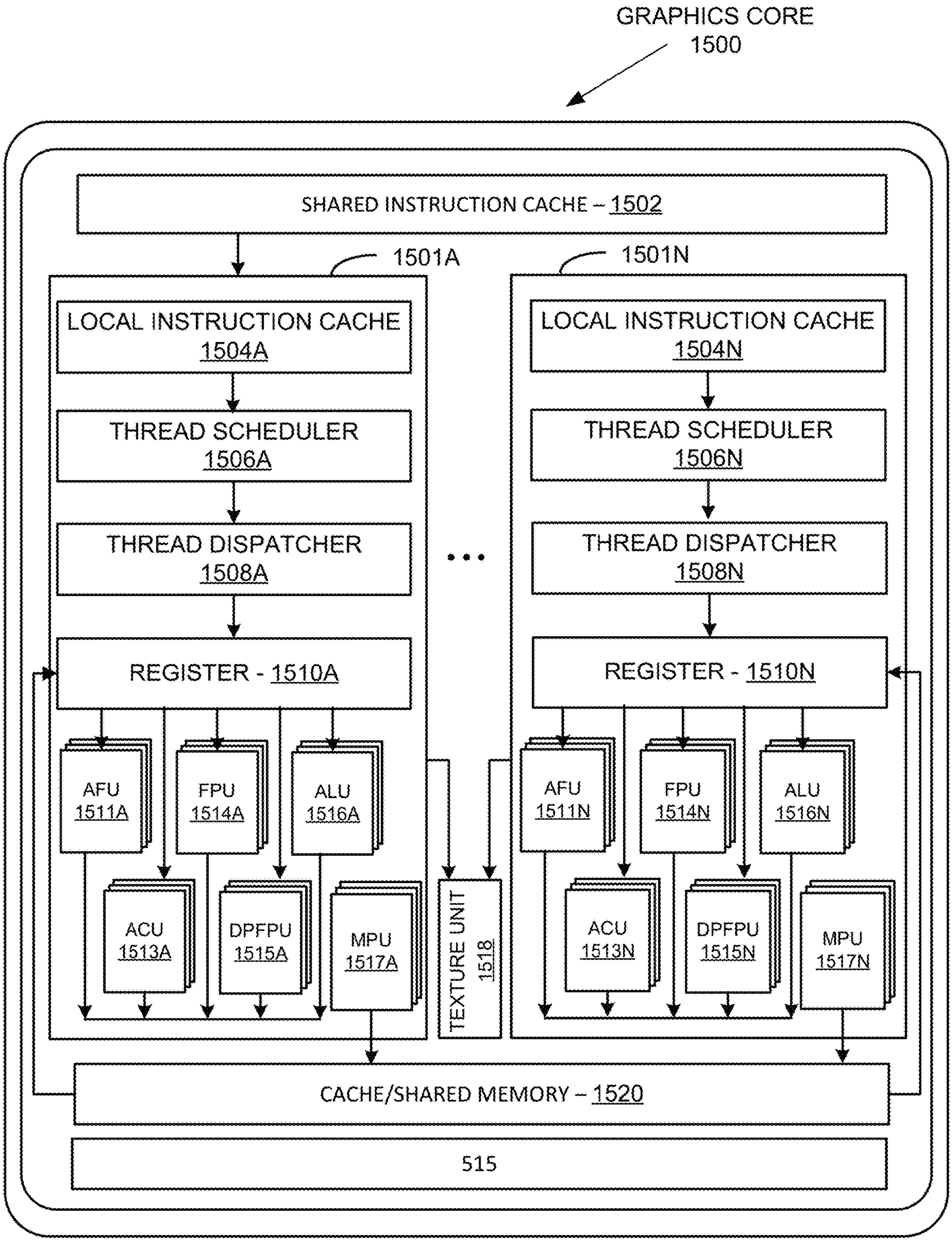

GRAPHICS CORE
1500
SHARED INSTRUCTION CACHE – 1502
1501A
1501N
LOCAL INSTRUCTION CACHE
1504A
LOCAL INSTRUCTION CACHE
1504N
THREAD SCHEDULER
1506A
THREAD SCHEDULER
1506N
THREAD DISPATCHER
1508A
THREAD DISPATCHER
1508N
REGISTER - 1510A
REGISTER - 1510N
AFU
1511A
FPU
1514A
ALU
1516A
AFU
1511N
FPU
1514N
ALU
1516N
ACU
1513A
DPFPU
1515A
MPU
1517A
TEXTURE UNIT
1518
ACU
1513N
DPFPU
1515N
MPU
1517N
CACHE/SHARED MEMORY – 1520
515

FIG. 15A

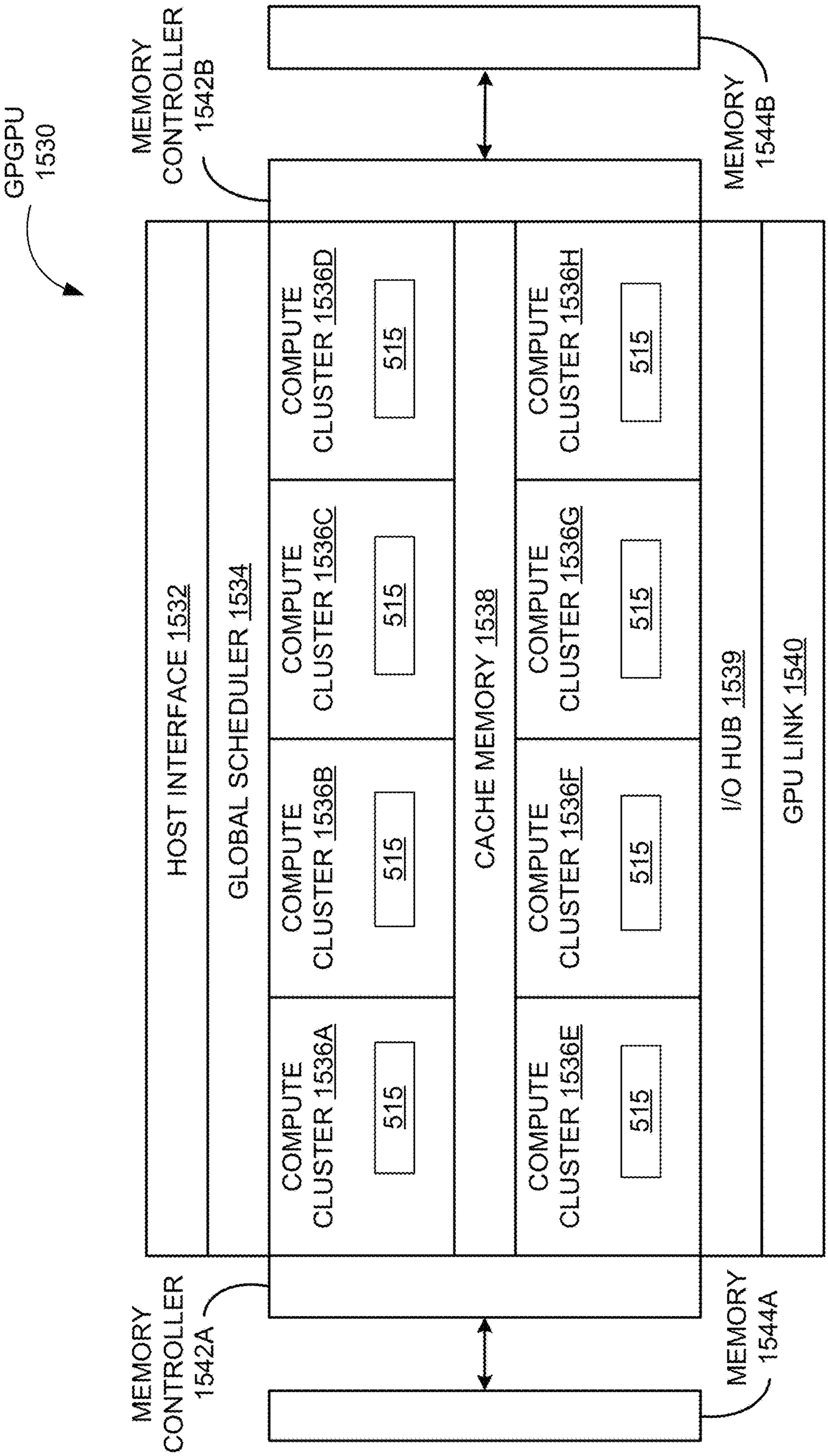

GPGPU 1530
MEMORY CONTROLLER 1542B
MEMORY 1544B
HOST INTERFACE 1532
GLOBAL SCHEDULER 1534
COMPUTE CLUSTER 1536D
COMPUTE CLUSTER 1536C
COMPUTE CLUSTER 1536B
COMPUTE CLUSTER 1536A
COMPUTE CLUSTER 1536H
COMPUTE CLUSTER 1536G
COMPUTE CLUSTER 1536F
COMPUTE CLUSTER 1536E
515
CACHE MEMORY 1538
I/O HUB 1539
GPU LINK 1540
MEMORY CONTROLLER 1542A
MEMORY 1544A

FIG. 15B

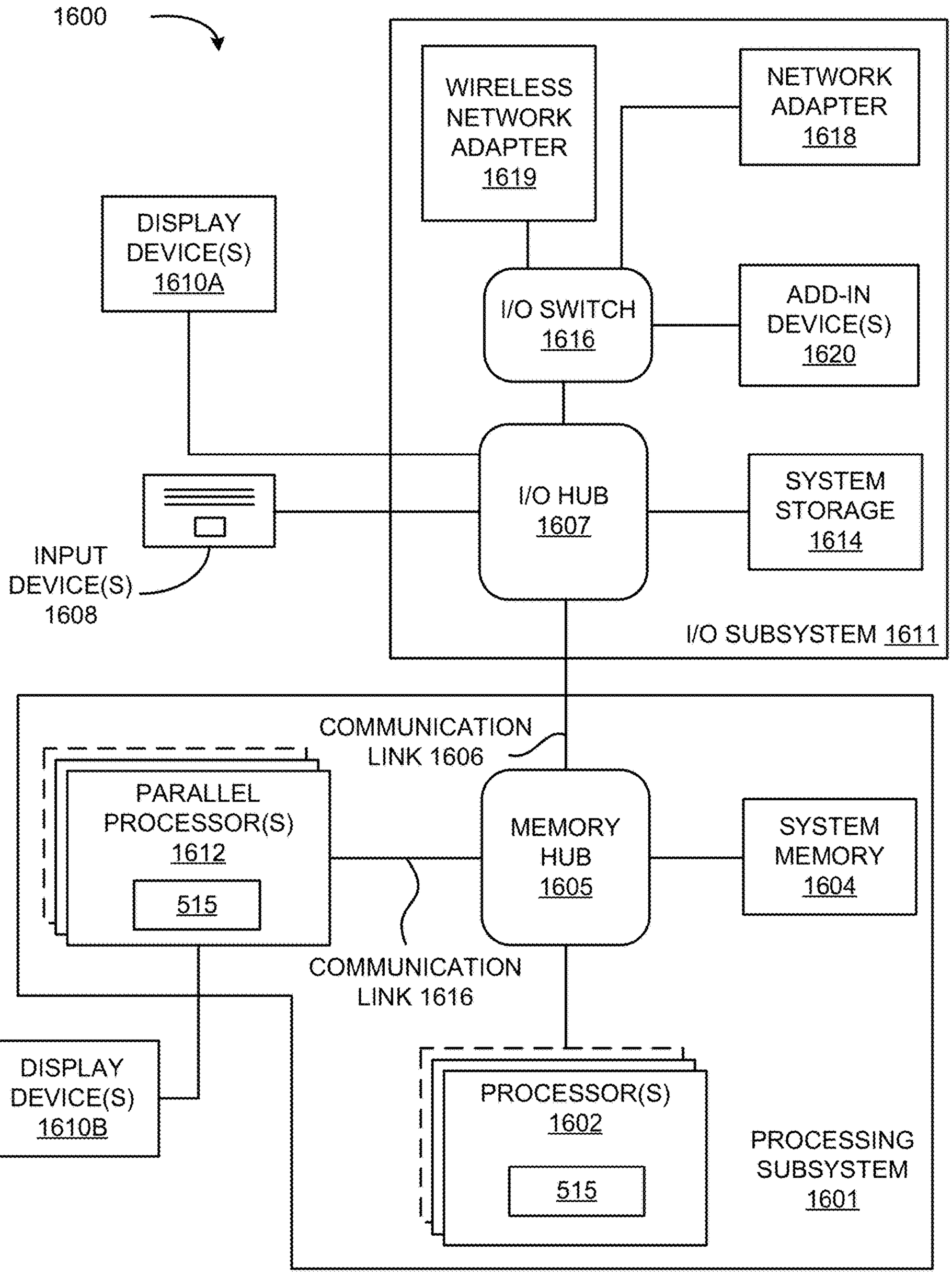

1600
WIRELESS NETWORK ADAPTER 1619
NETWORK ADAPTER 1618
DISPLAY DEVICE(S) 1610A
I/O SWITCH 1616
ADD-IN DEVICE(S) 1620
I/O HUB 1607
SYSTEM STORAGE 1614
INPUT DEVICE(S) 1608
I/O SUBSYSTEM 1611
COMMUNICATION LINK 1606
PARALLEL PROCESSOR(S) 1612
515
MEMORY HUB 1605
SYSTEM MEMORY 1604
COMMUNICATION LINK 1616
DISPLAY DEVICE(S) 1610B
PROCESSOR(S) 1602
515
PROCESSING SUBSYSTEM 1601

FIG. 16

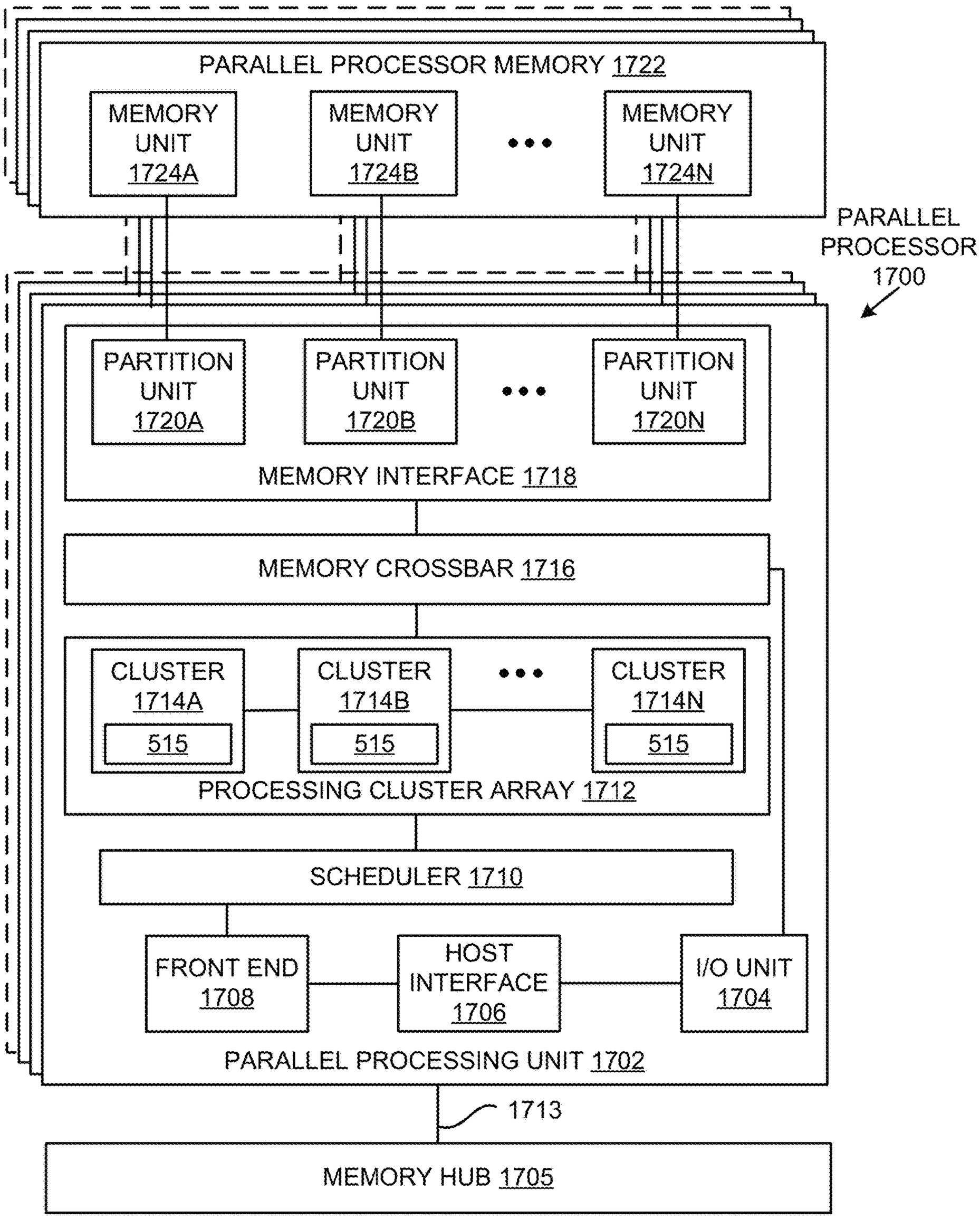

PARALLEL PROCESSOR MEMORY 1722
MEMORY UNIT 1724A
MEMORY UNIT 1724B
MEMORY UNIT 1724N
PARALLEL PROCESSOR 1700
PARTITION UNIT 1720A
PARTITION UNIT 1720B
PARTITION UNIT 1720N
MEMORY INTERFACE 1718
MEMORY CROSSBAR 1716
CLUSTER 1714A
CLUSTER 1714B
CLUSTER 1714N
515
515
515
PROCESSING CLUSTER ARRAY 1712
SCHEDULER 1710
FRONT END 1708
HOST INTERFACE 1706
I/O UNIT 1704
PARALLEL PROCESSING UNIT 1702
1713
MEMORY HUB 1705

FIG. 17 A

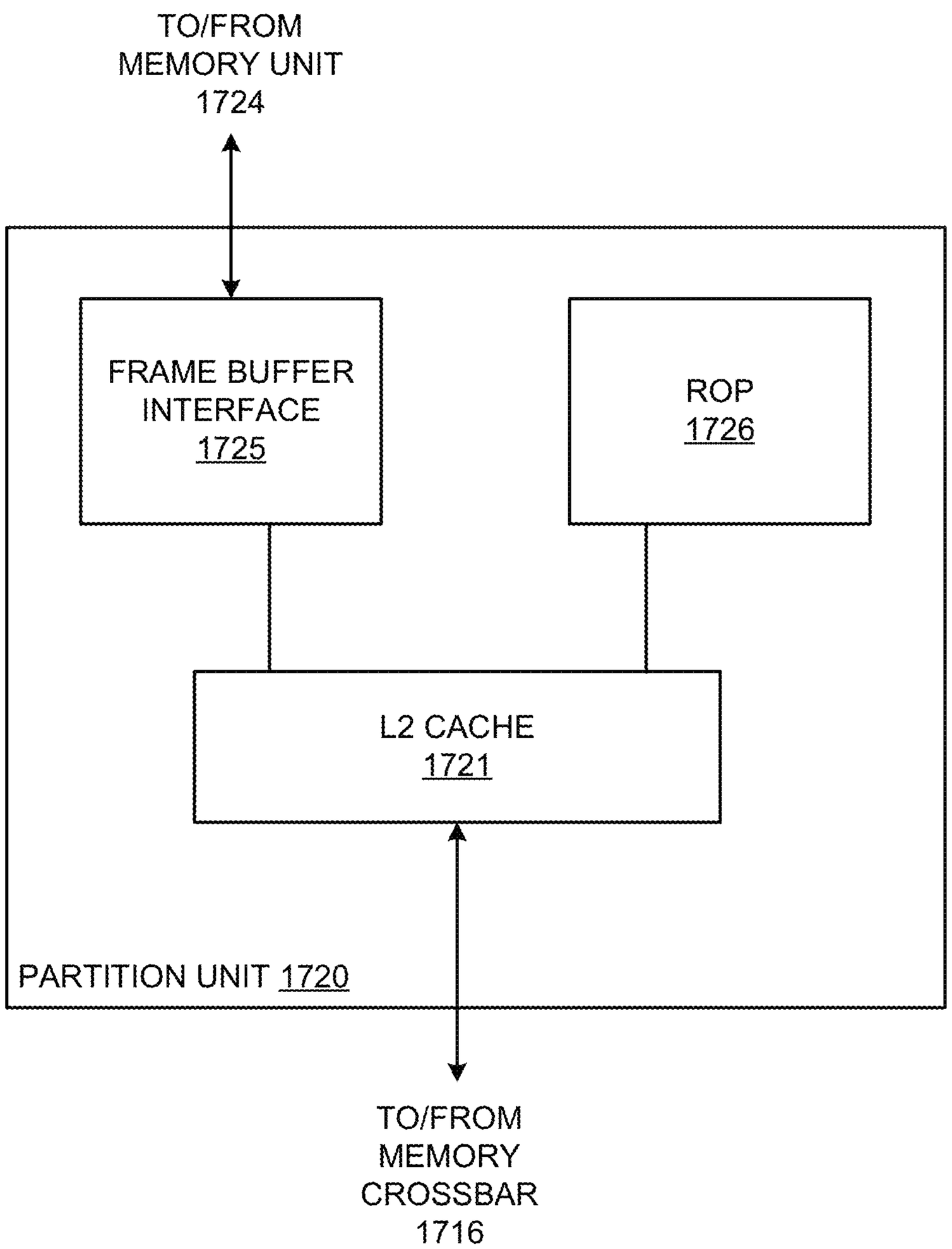

TO/FROM
MEMORY UNIT
1724
FRAME BUFFER
INTERFACE
1725
ROP
1726
L2 CACHE
1721
PARTITION UNIT 1720
TO/FROM
MEMORY
CROSSBAR
1716

FIG. 17 B

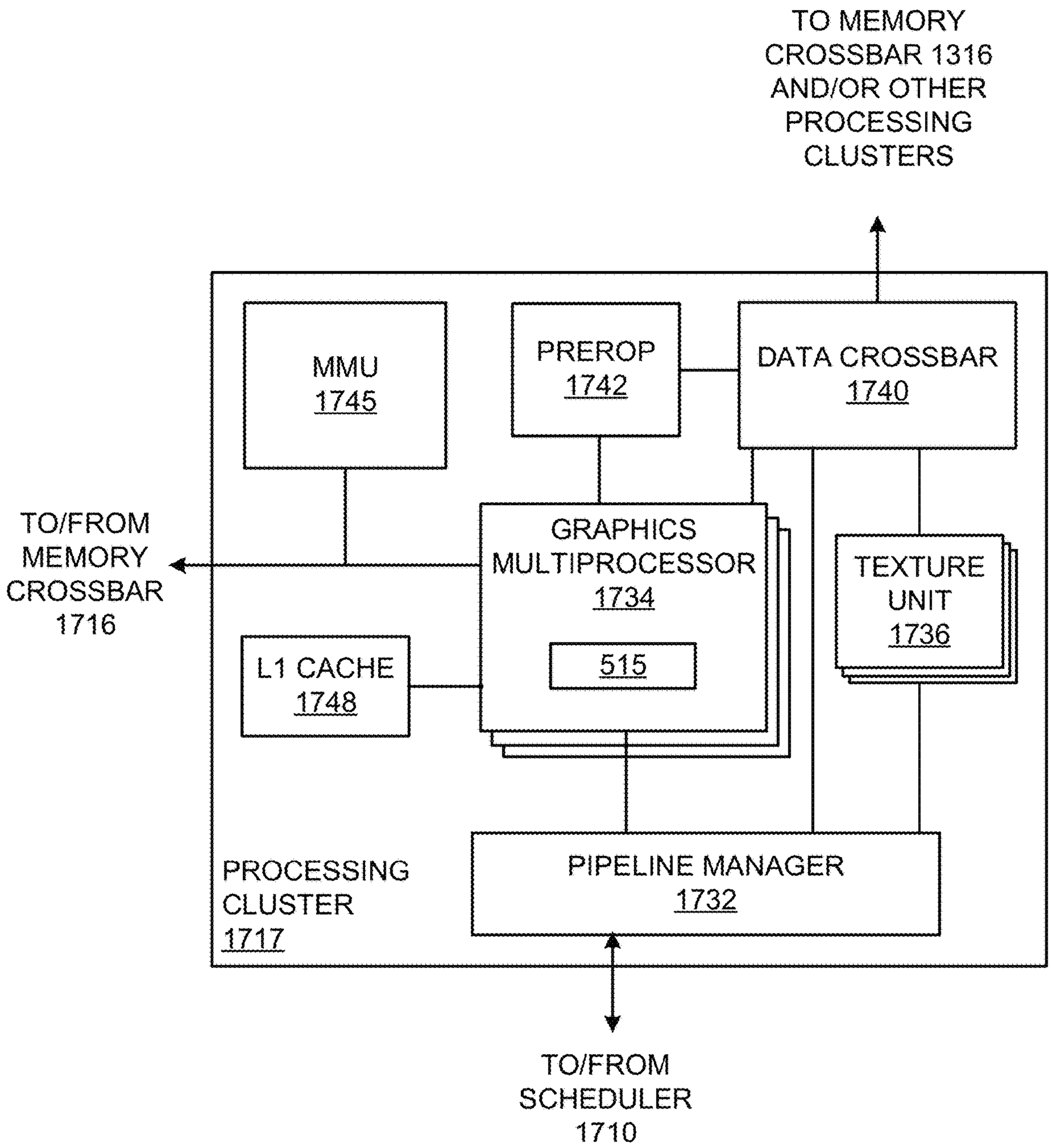

TO MEMORY CROSSBAR 1316 AND/OR OTHER PROCESSING CLUSTERS
MMU 1745
PREROP 1742
DATA CROSSBAR 1740
TO/FROM MEMORY CROSSBAR 1716
GRAPHICS MULTIPROCESSOR 1734
515
TEXTURE UNIT 1736
L1 CACHE 1748
PROCESSING CLUSTER 1717
PIPELINE MANAGER 1732
TO/FROM SCHEDULER 1710

FIG. 17 C

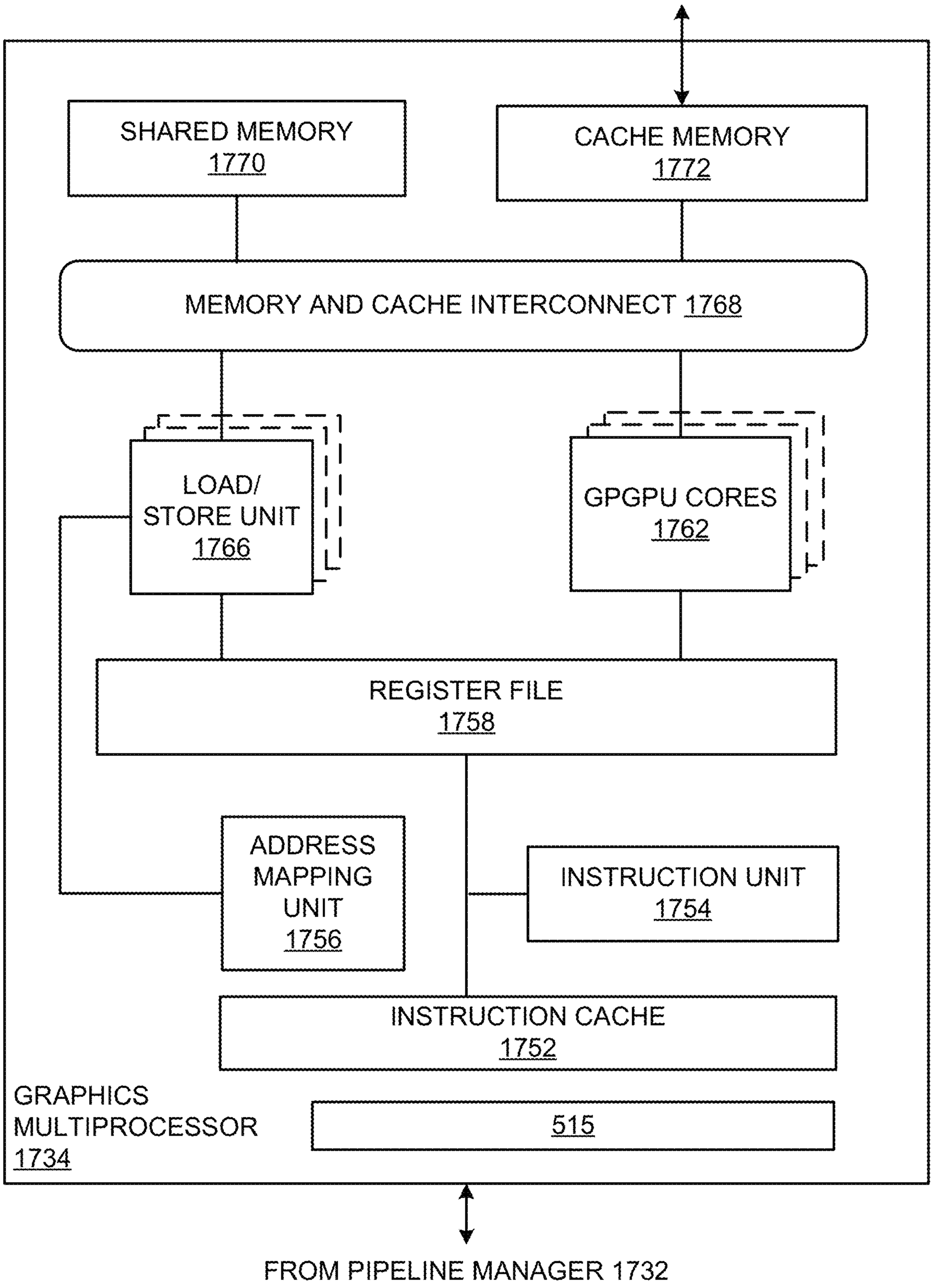

SHARED MEMORY
1770
CACHE MEMORY
1772
MEMORY AND CACHE INTERCONNECT 1768
LOAD/
STORE UNIT
1766
GPGPU CORES
1762
REGISTER FILE
1758
ADDRESS
MAPPING
UNIT
1756
INSTRUCTION UNIT
1754
INSTRUCTION CACHE
1752
GRAPHICS
MULTIPROCESSOR
1734
515
FROM PIPELINE MANAGER 1732

FIG. 17 D

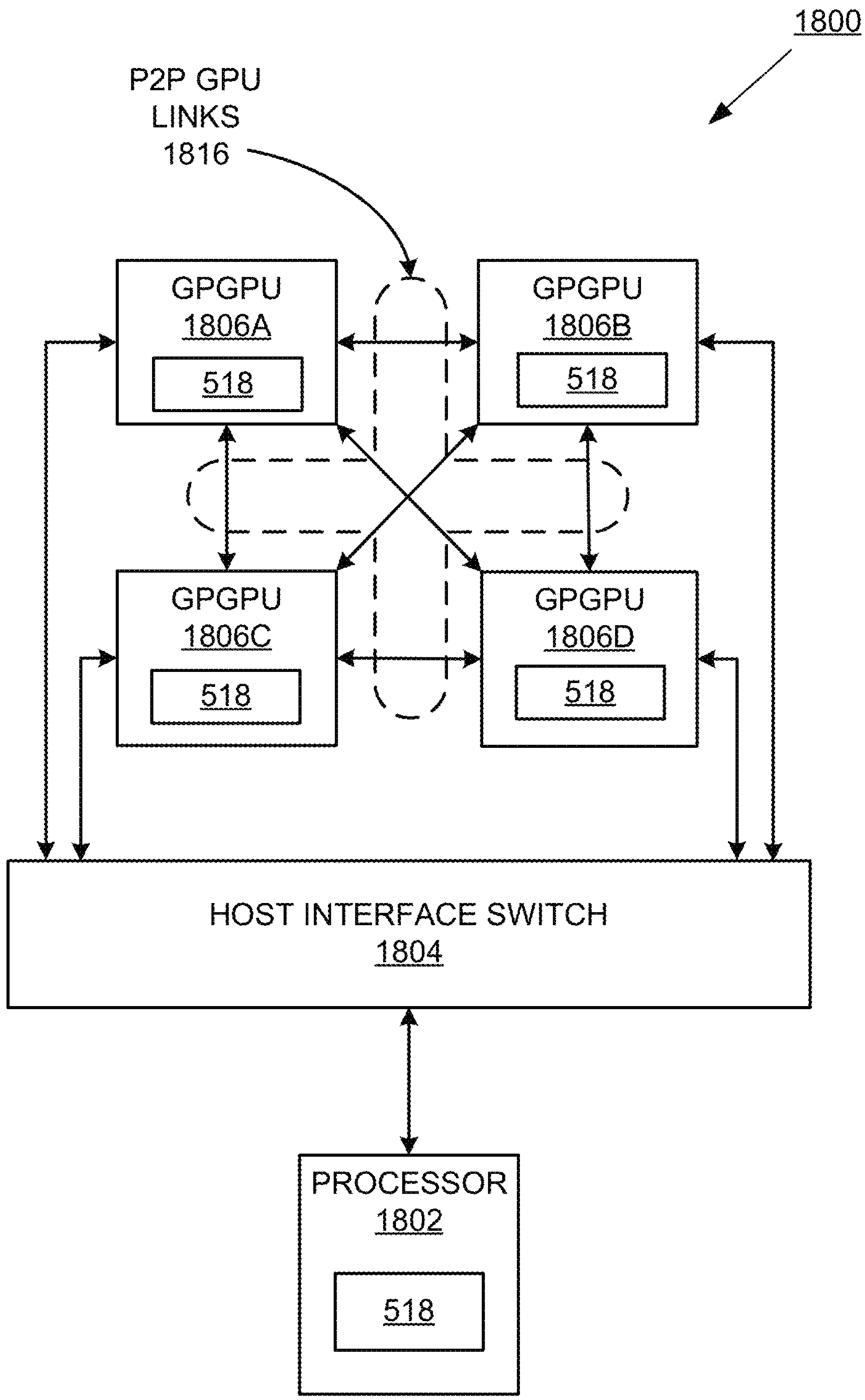

1800
P2P GPU LINKS 1816
GPGPU 1806A
518
GPGPU 1806B
518
GPGPU 1806C
518
GPGPU 1806D
518
HOST INTERFACE SWITCH 1804
PROCESSOR 1802
518

FIG. 18

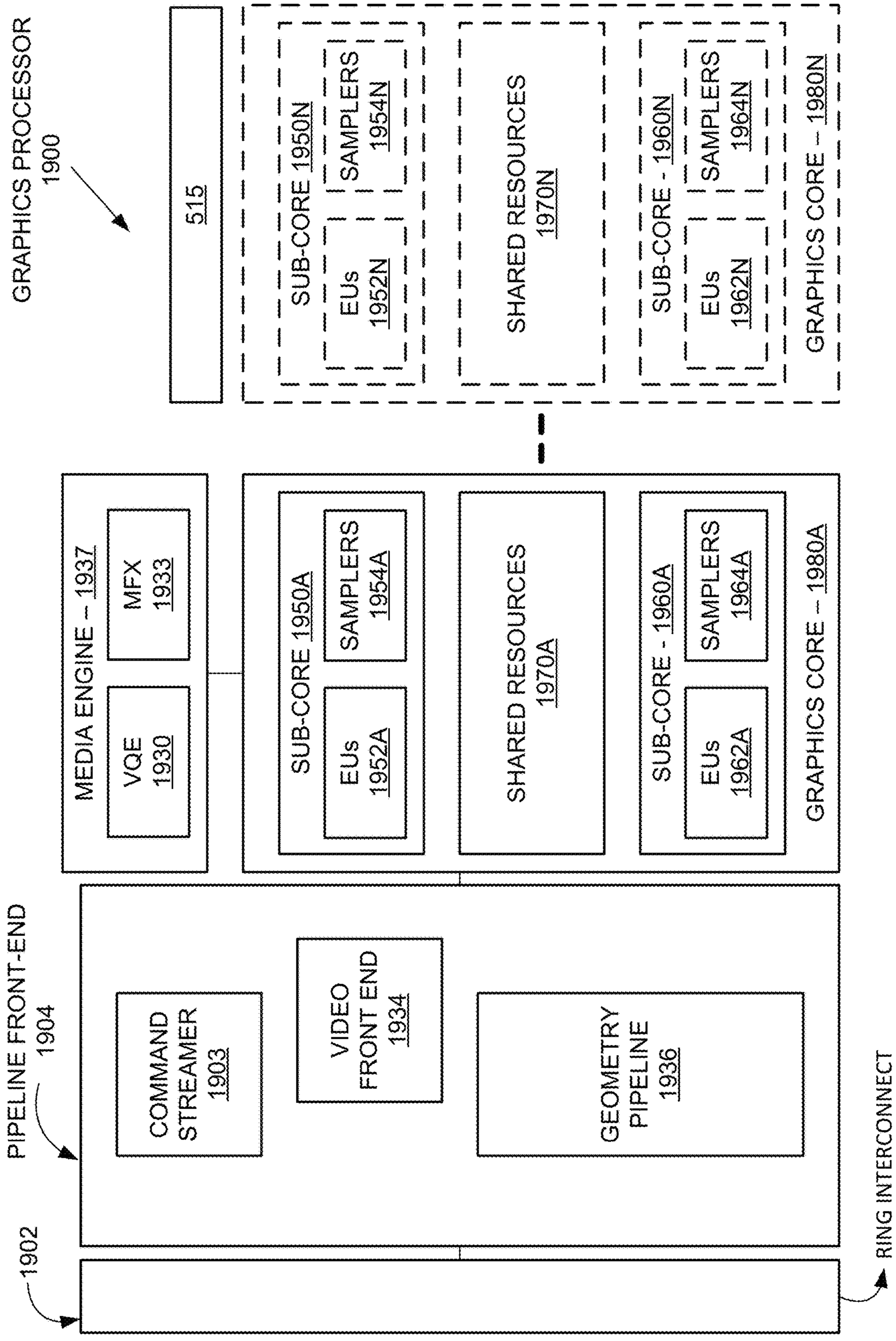

GRAPHICS PROCESSOR
1900
515
GRAPHICS CORE – 1980N
SUB-CORE 1950N
EUs
1952N
SAMPLERS
1954N
SHARED RESOURCES
1970N
SUB-CORE - 1960N
EUs
1962N
SAMPLERS
1964N
MEDIA ENGINE – 1937
VQE
1930
MFX
1933
GRAPHICS CORE – 1980A
SUB-CORE 1950A
EUs
1952A
SAMPLERS
1954A
SHARED RESOURCES
1970A
SUB-CORE - 1960A
EUs
1962A
SAMPLERS
1964A
PIPELINE FRONT-END
1904
COMMAND
STREAMER
1903
VIDEO
FRONT END
1934
GEOMETRY
PIPELINE
1936
1902
RING INTERCONNECT

FIG. 19

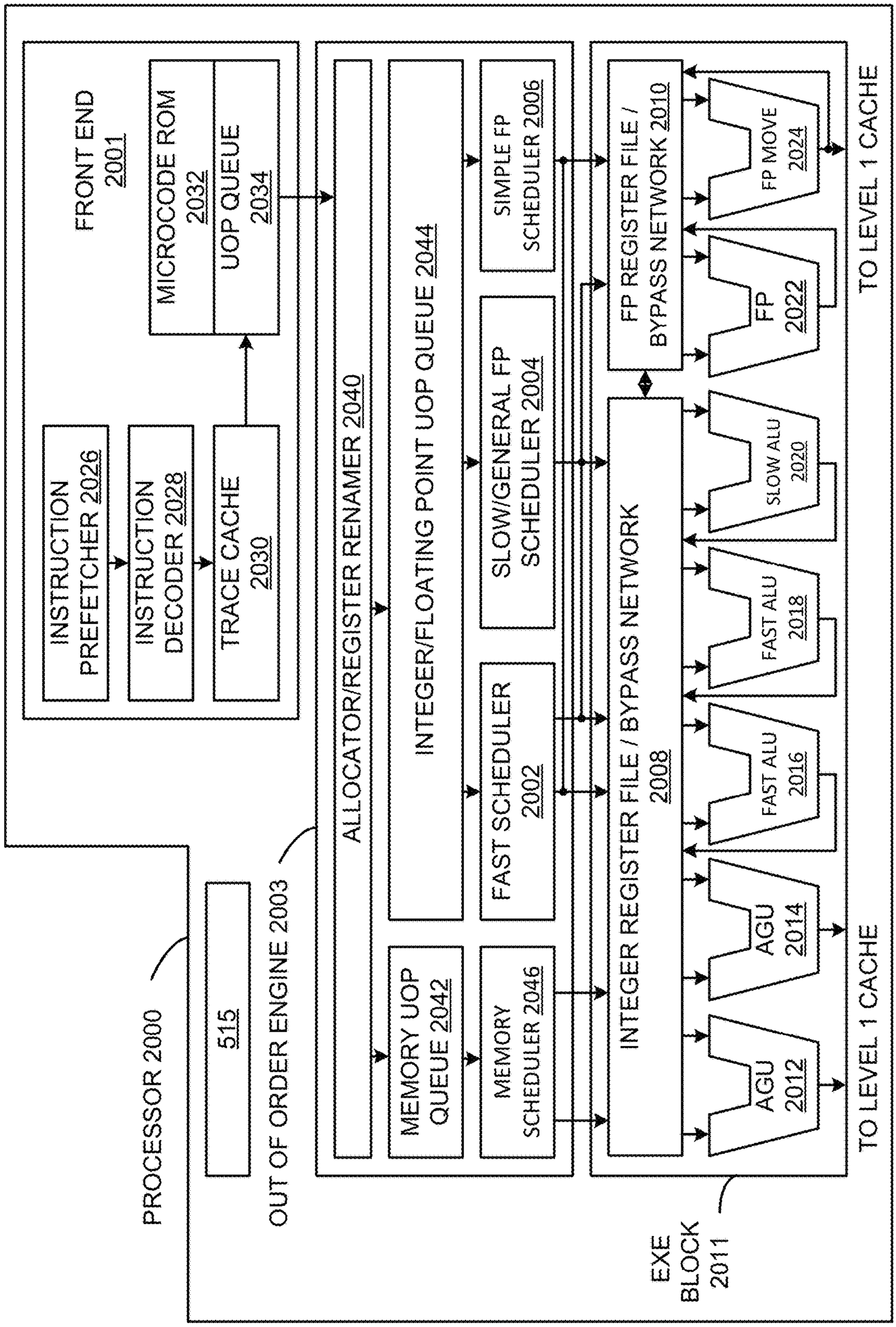

PROCESSOR 2000
FRONT END 2001
MICROCODE ROM 2032
UOP QUEUE 2034
INSTRUCTION PREFETCHER 2026
INSTRUCTION DECODER 2028
TRACE CACHE 2030
515
OUT OF ORDER ENGINE 2003
ALLOCATOR/REGISTER RENAMER 2040
INTEGER/FLOATING POINT UOP QUEUE 2044
MEMORY UOP QUEUE 2042
SIMPLE FP SCHEDULER 2006
SLOW/GENERAL FP SCHEDULER 2004
MEMORY SCHEDULER 2046
FP REGISTER FILE / BYPASS NETWORK 2010
FP MOVE 2024
FP 2022
SLOW ALU 2020
FAST ALU 2018
FAST ALU 2016
INTEGER REGISTER FILE / BYPASS NETWORK 2008
AGU 2014
AGU 2012
TO LEVEL 1 CACHE
EXE BLOCK 2011

FIG. 20

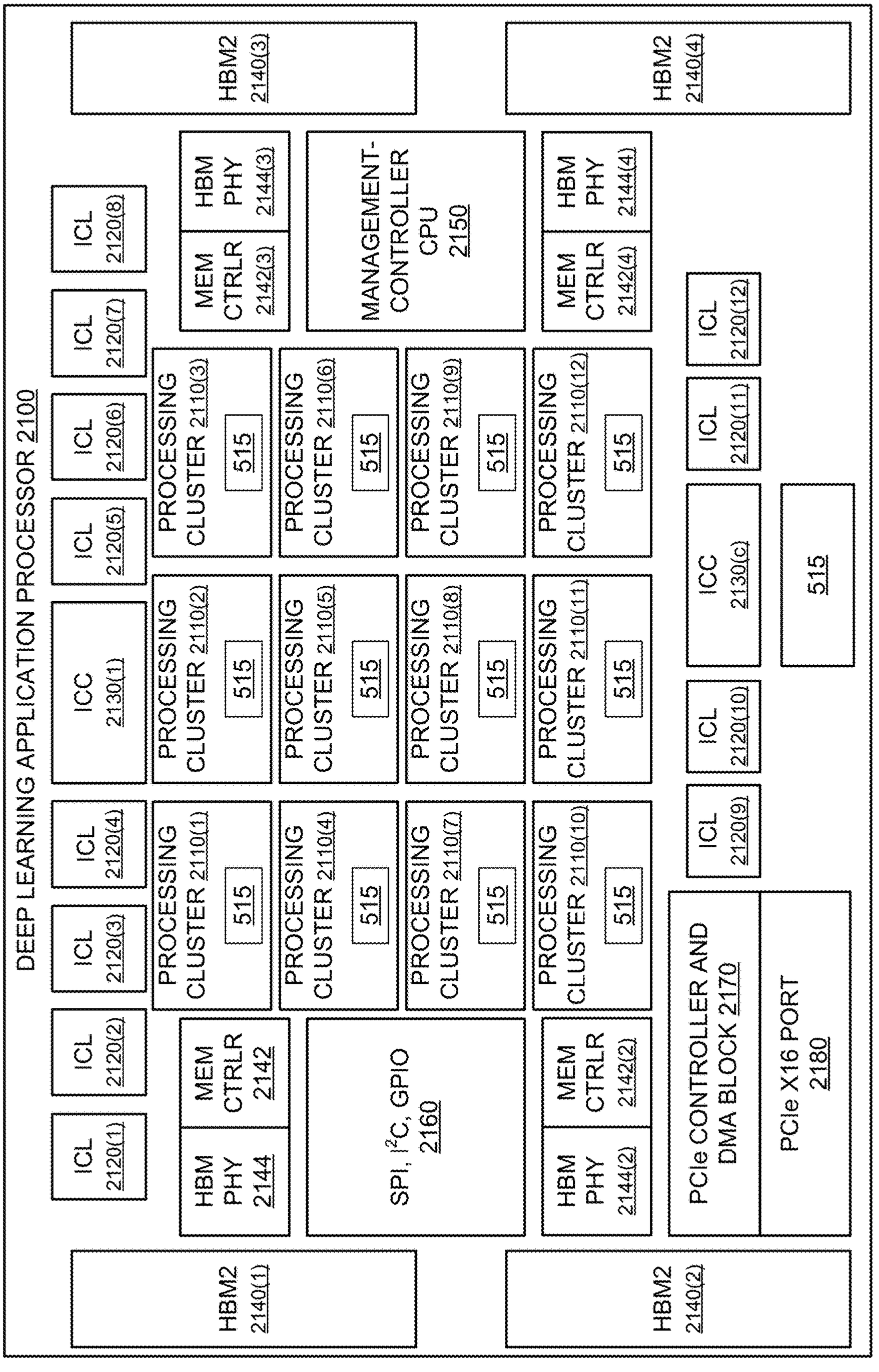

DEEP LEARNING APPLICATION PROCESSOR 2100
HBM2 2140(3)
HBM2 2140(4)
ICL 2120(8)
HBM PHY 2144(3)
MEM CTRLR 2142(3)
MANAGEMENT-CONTROLLER CPU 2150
HBM PHY 2144(4)
MEM CTRLR 2142(4)
ICL 2120(12)
ICL 2120(7)
ICL 2120(4)
ICC 2130(1)
ICL 2120(5)
ICL 2120(6)
PROCESSING CLUSTER 2110(3)
PROCESSING CLUSTER 2110(2)
PROCESSING CLUSTER 2110(1)
PROCESSING CLUSTER 2110(6)
PROCESSING CLUSTER 2110(5)
PROCESSING CLUSTER 2110(4)
PROCESSING CLUSTER 2110(9)
PROCESSING CLUSTER 2110(8)
PROCESSING CLUSTER 2110(7)
PROCESSING CLUSTER 2110(12)
PROCESSING CLUSTER 2110(11)
PROCESSING CLUSTER 2110(10)
515
ICL 2120(9)
ICL 2120(10)
ICC 2130(c)
ICL 2120(11)
ICL 2120(3)
ICL 2120(2)
ICL 2120(1)
HBM PHY 2144
MEM CTRLR 2142
SPI, I2C, GPIO 2160
HBM PHY 2144(2)
MEM CTRLR 2142(2)
PCIe CONTROLLER AND DMA BLOCK 2170
PCIe X16 PORT 2180
HBM2 2140(1)
HBM2 2140(2)

FIG. 21

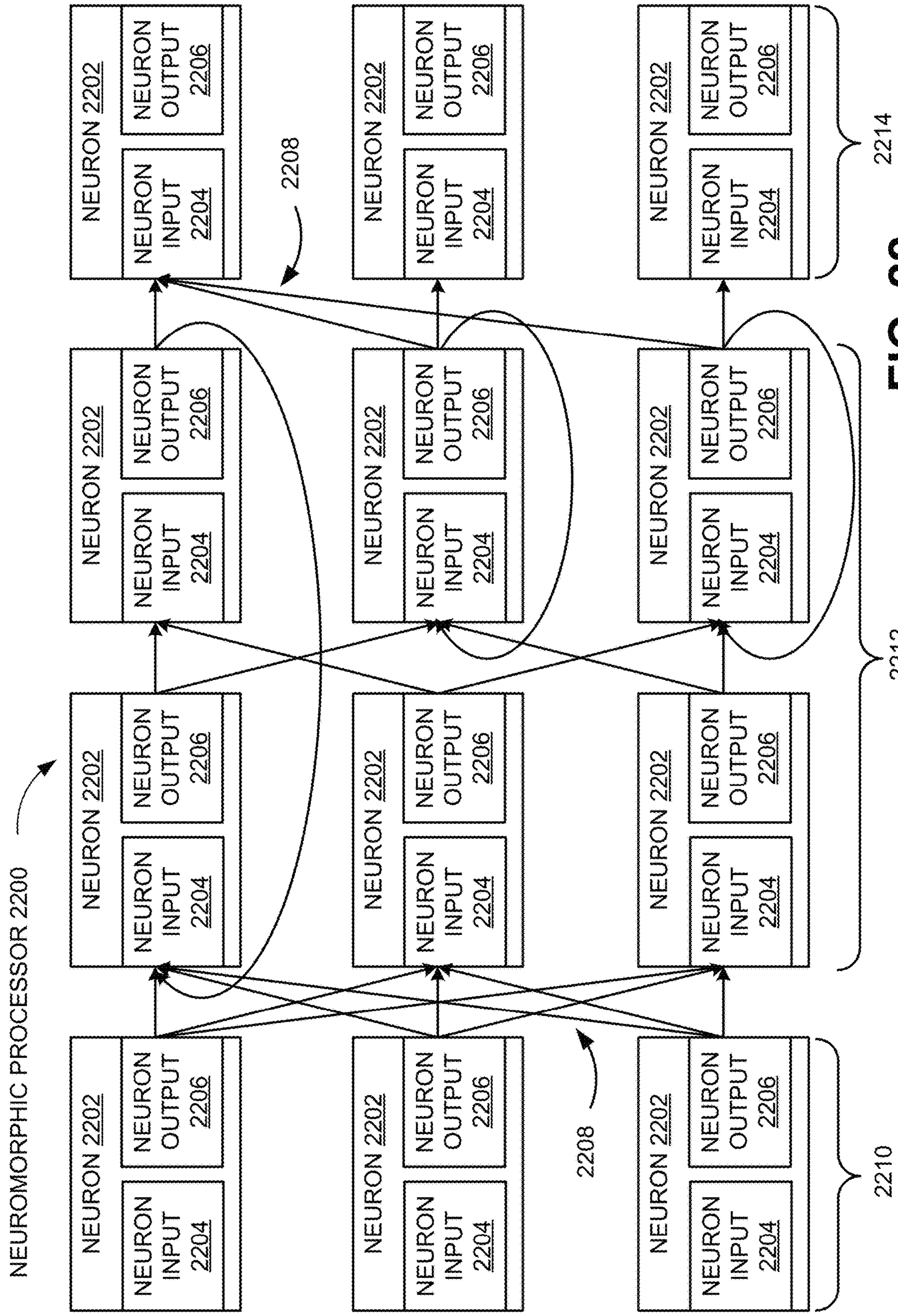

NEUROMORPHIC PROCESSOR 2200
NEURON 2202
NEURON INPUT 2204
NEURON OUTPUT 2206
2208
2210
2212
2214

FIG. 22

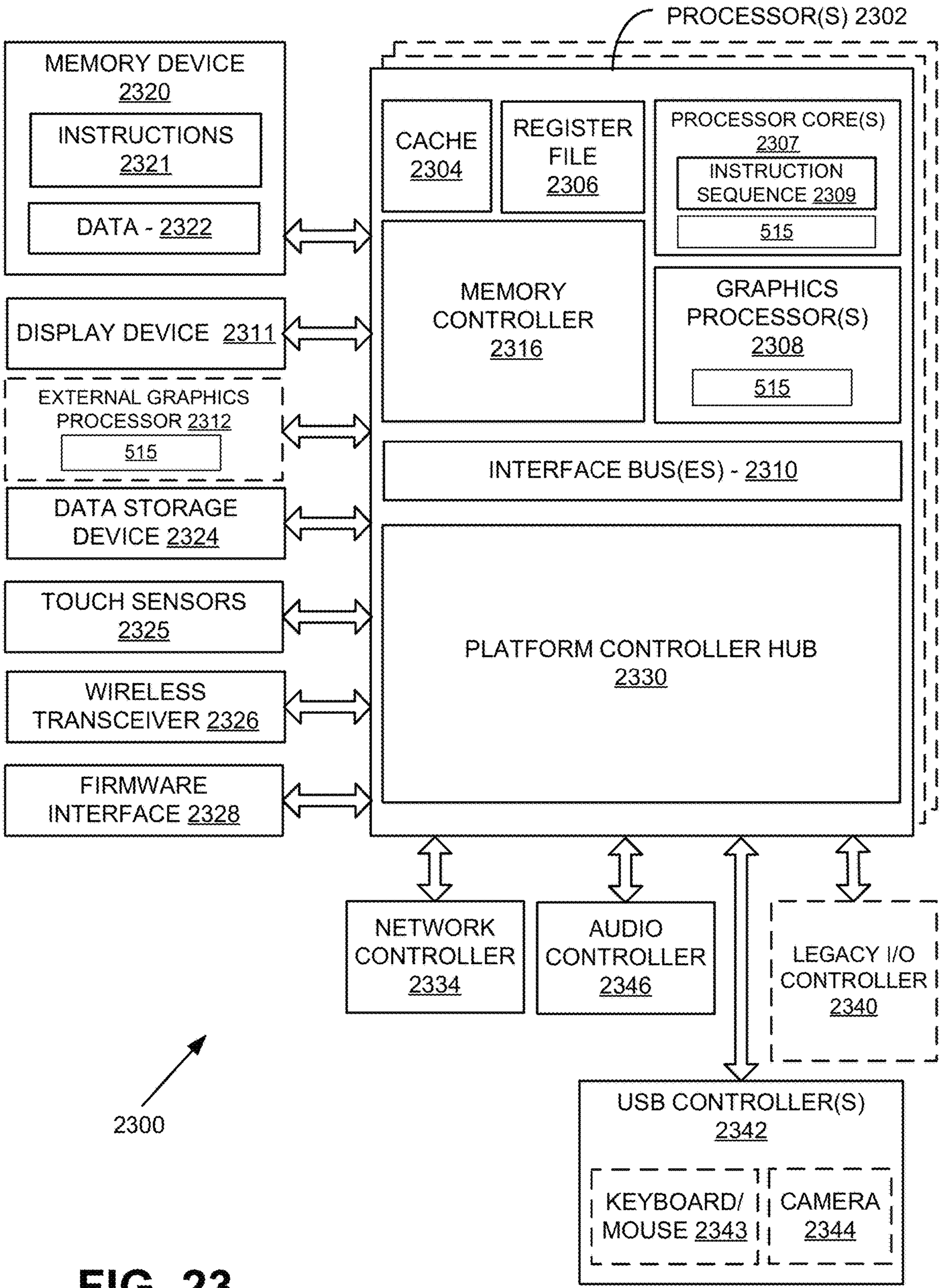

PROCESSOR(S) 2302
MEMORY DEVICE 2320
INSTRUCTIONS 2321
DATA - 2322
CACHE 2304
REGISTER FILE 2306
PROCESSOR CORE(S) 2307
INSTRUCTION SEQUENCE 2309
515
MEMORY CONTROLLER 2316
GRAPHICS PROCESSOR(S) 2308
515
DISPLAY DEVICE 2311
EXTERNAL GRAPHICS PROCESSOR 2312
515
INTERFACE BUS(ES) - 2310
DATA STORAGE DEVICE 2324
TOUCH SENSORS 2325
PLATFORM CONTROLLER HUB 2330
WIRELESS TRANSCEIVER 2326
FIRMWARE INTERFACE 2328
NETWORK CONTROLLER 2334
AUDIO CONTROLLER 2346
LEGACY I/O CONTROLLER 2340
USB CONTROLLER(S) 2342
KEYBOARD/ MOUSE 2343
CAMERA 2344
2300

FIG. 23

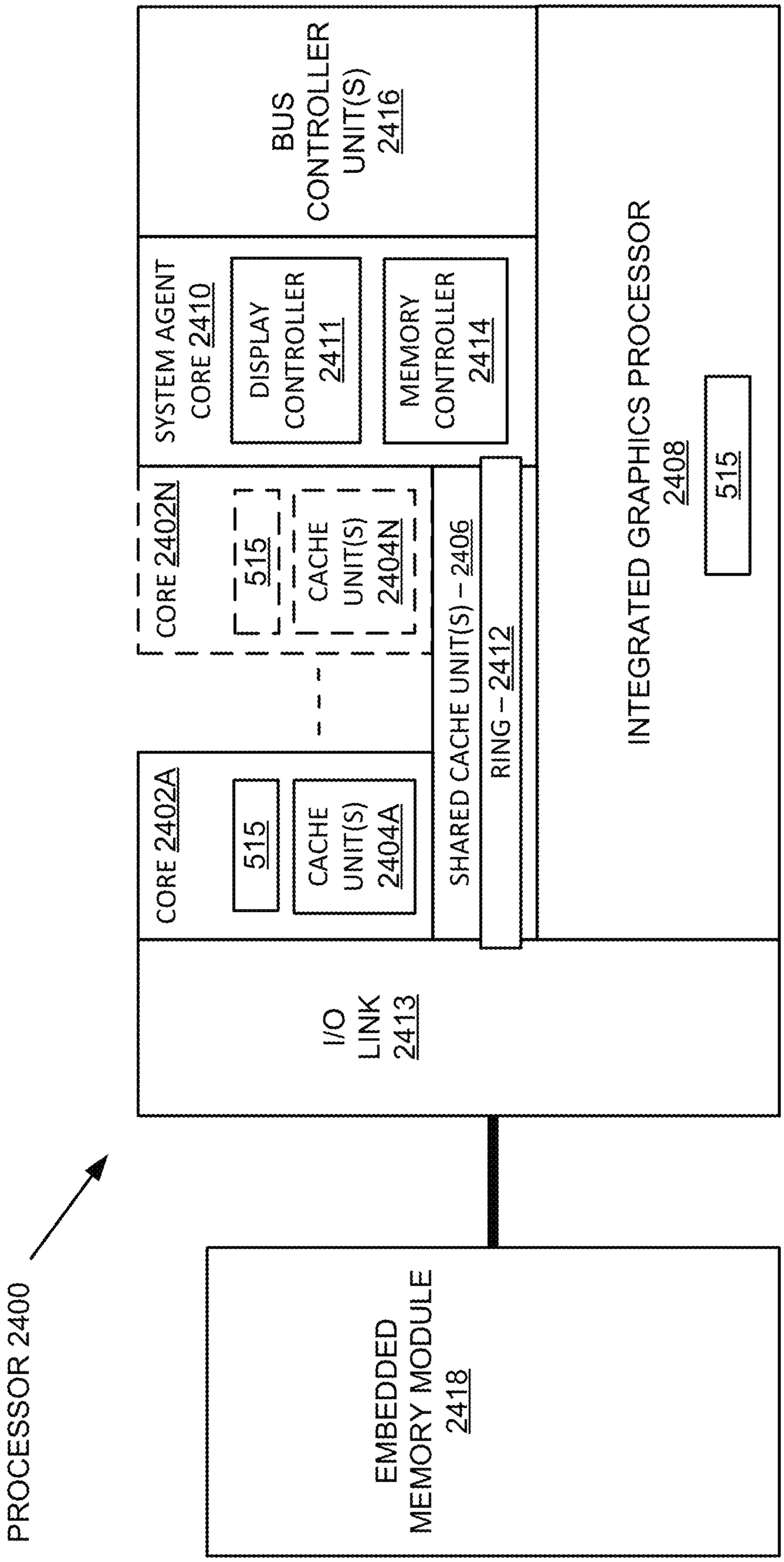

PROCESSOR 2400
I/O LINK 2413
SYSTEM AGENT CORE 2410
DISPLAY CONTROLLER 2411
MEMORY CONTROLLER 2414
BUS CONTROLLER UNIT(S) 2416
CORE 2402A
515
CACHE UNIT(S) 2404A
CORE 2402N
515
CACHE UNIT(S) 2404N
SHARED CACHE UNIT(S) – 2406
RING – 2412
INTEGRATED GRAPHICS PROCESSOR 2408
515
EMBEDDED MEMORY MODULE 2418

FIG. 24

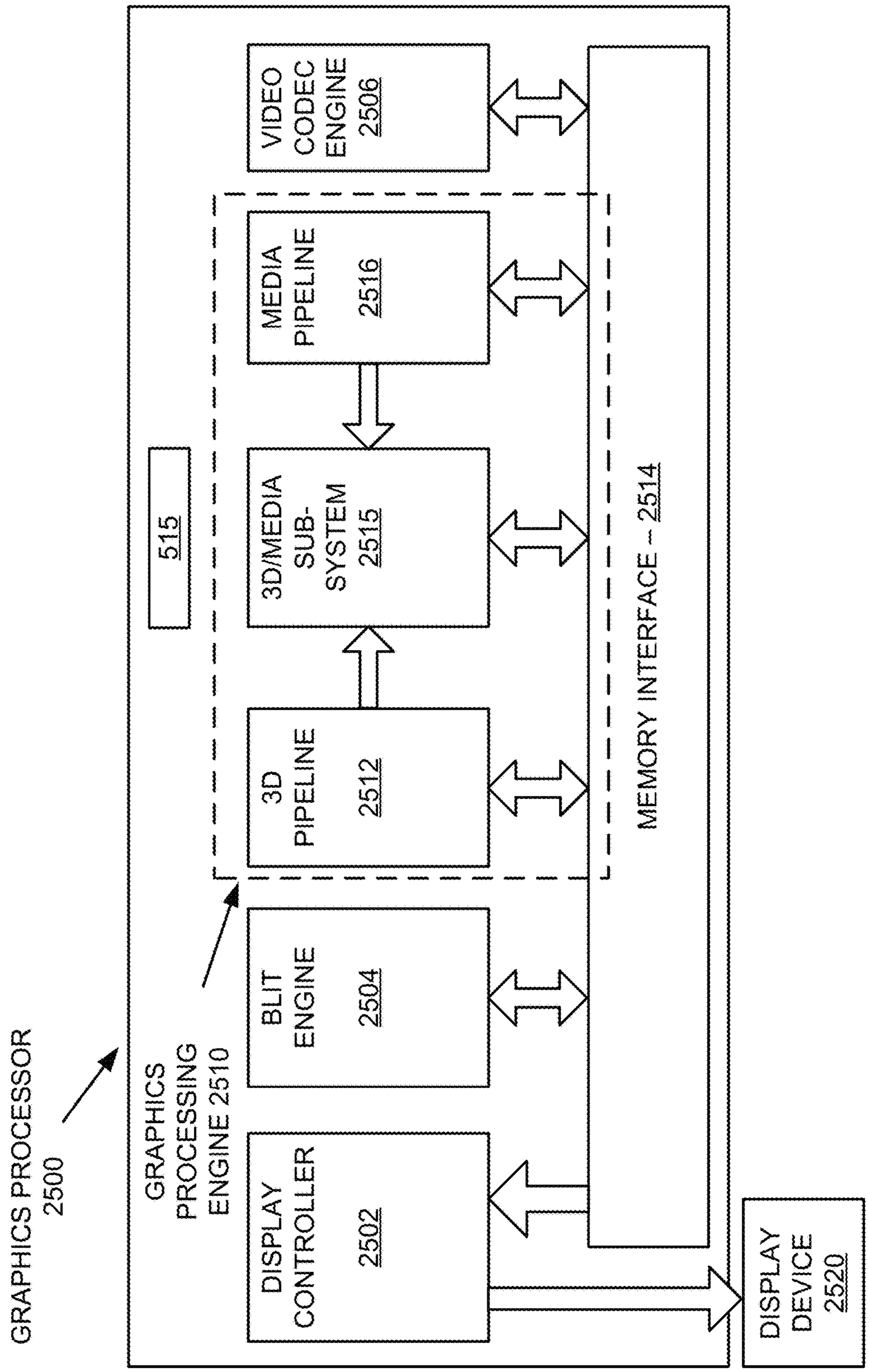

GRAPHICS PROCESSOR
2500
GRAPHICS
PROCESSING
ENGINE 2510
DISPLAY
CONTROLLER
2502
BLIT
ENGINE
2504
3D
PIPELINE
2512
3D/MEDIA
SUB-
SYSTEM
2515
MEDIA
PIPELINE
2516
515
VIDEO
CODEC
ENGINE
2506
MEMORY INTERFACE – 2514
DISPLAY
DEVICE
2520

FIG. 25

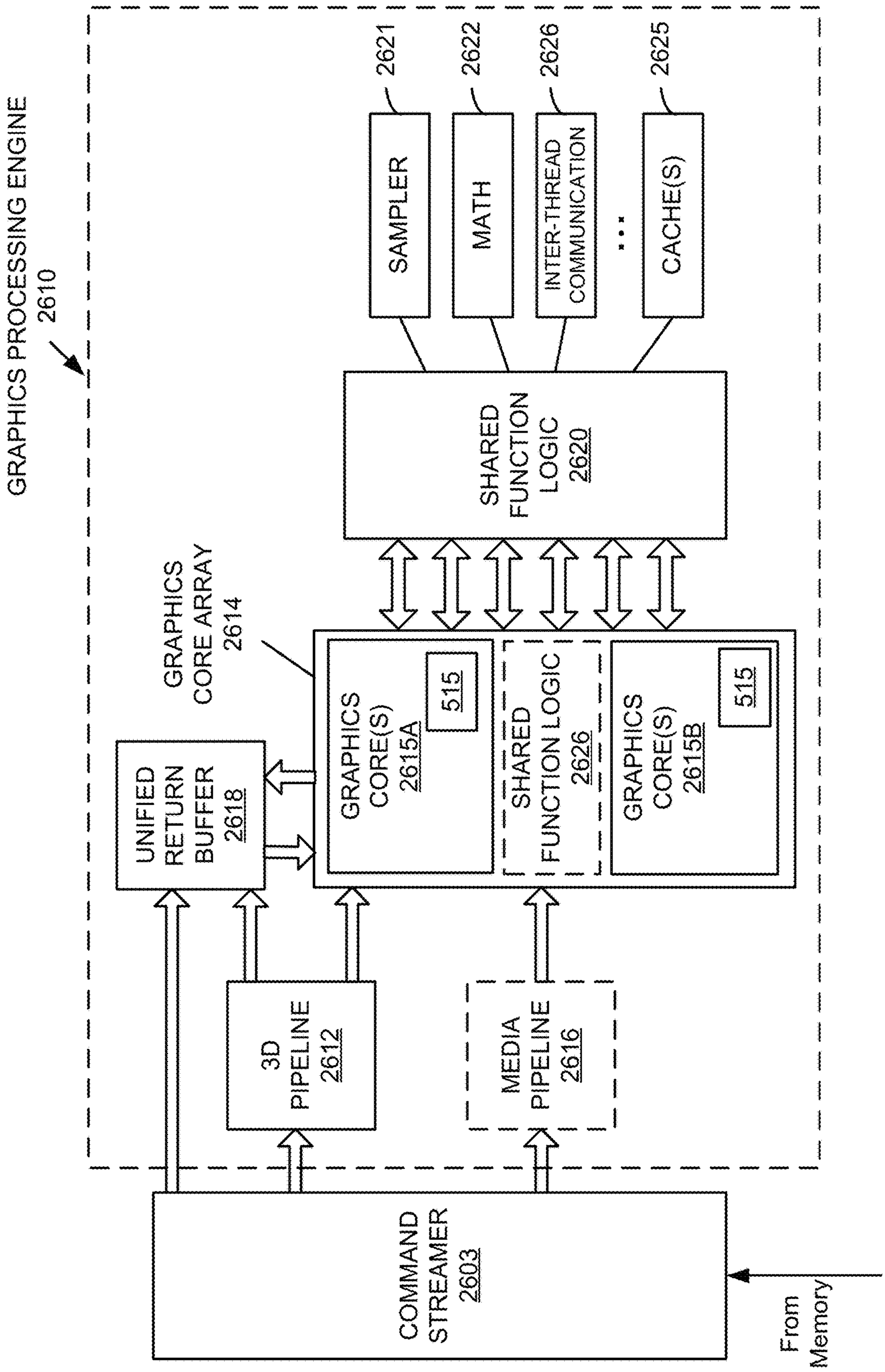

GRAPHICS PROCESSING ENGINE
2610
SAMPLER
2621
MATH
2622
INTER-THREAD COMMUNICATION
2626
...
CACHE(S)
2625
SHARED FUNCTION LOGIC
2620
GRAPHICS CORE ARRAY
2614
GRAPHICS CORE(S)
2615A
515
SHARED FUNCTION LOGIC
2626
GRAPHICS CORE(S)
2615B
515
UNIFIED RETURN BUFFER
2618
3D PIPELINE
2612
MEDIA PIPELINE
2616
COMMAND STREAMER
2603
From Memory

FIG. 26

ILLUMINATION RESAMPLING USING TEMPORAL GRADIENTS IN LIGHT TRANSPORT SIMULATION SYSTEMS AND APPLICATIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Provisional Patent Application Ser. No. 63/273,834, entitled "Computing and Using Temporal Gradients in Screen-Space Light Resampling Algorithms," filed Oct. 29, 2021, which is hereby incorporated herein in its entirety and for all purposes.

BACKGROUND

Rendering algorithms based on ray tracing and other light transport simulation techniques often produce a noisy image that needs to be denoised to be useful. Approaches such as spatiotemporal light resampling (ReSTIR) can produce less noisy images than other previous light sampling algorithms, but there is still some noise to be removed. Modern denoisers, such as those provided by the NRD library from NVIDIA Corporation, include spatiotemporal filters that can accumulate lighting information over multiple frames in order to produce a stable output signal. While this approach is very effective for noise reduction, the output signal often reacts to abrupt changes in the input relatively slowly. This can be seen as light turning on and off smoothly instead of instantly, shadows lagging behind the objects that cast them, and illumination from moving lights appearing (undesirably) as streaks or as smears. While modern denoisers may include some heuristics to mitigate these effects, these heuristics are only somewhat effective.

One approach to further reducing noise takes advantage of adaptive spatiotemporal variance-guided filtering (A-SVGF) to compute a hint, or confidence input. The A-SVGF algorithm computes "temporal gradients" as differences in shading of the same surfaces on two consecutive frames using the same random numbers. These differences are zero if the lighting environment of the surface stays the same, and are nonzero if something has changed, such as when a light has moved relative to the surface or has come in (or out) of shadow. However, it is not always immediately clear how to compute a robust confidence input from ReSTIR. A straightforward reuse of random numbers for gradient computation may not always work because there is additional persistent state, as may correspond to one or more light reservoirs, that can be carried over from the previous frame and modified. Therefore, using the previous random number sequence for a surface is likely to yield a different result because one of the reservoir reuse passes may replace or modify the light reservoir in the pixel. In A-SVGF, the image is subdivided into 3×3 squares, or strata, and one pixel is selected from each stratum. That pixel is forward-projected from the previous frame to the current frame in order to create a "temporal gradient." The reprojection makes sure that no more than one temporal gradient ends up in every stratum of the current frame, and that the temporal gradient carries some parameters of the previous surface over to the current frame. The parameters include the random number seed used to shade this surface on the previous frame, which allows the algorithm to shade the same surface in a way that is directly comparable with the previous frame. As a result, if the temporal gradient pixels produce the same shading output on the previous and current frames, the lighting in this area is considered valid; if the shading outputs are different, the lighting is invalid, and the denoiser history should be reset. While A-SVGF works in various situations, it is not always practical for more complex renderers due to the fragility that comes from reusing random number sequences. Great care must be taken to avoid computing false positive gradients based on unrelated changes in the scene. For example, a change in the total light count can cause the light sampling logic to select different lights using the same random number generator (RNG) sequence even though the surface being shaded may be unaffected by the change at all. In some instances, reusing the RNG sequence doesn't look sufficient at all because the shading results depend on a persistent state that is different from frame to frame, specifically the light reservoirs.

BRIEF DESCRIPTION OF THE DRAWINGS

Various embodiments in accordance with the present disclosure will be described with reference to the drawings, in which:

FIG. **1** illustrates components of an example image generation pipeline, in accordance with various embodiments;

FIG. **2** illustrates a process for sample evaluation and selection, in accordance with various embodiments;

FIGS. **3**A, **3**B, and **3**C illustrate images with confidence values generated using different approaches, in accordance with various embodiments;

FIG. **4** illustrates an example rendering pipeline, in accordance with various embodiments;

FIG. **5** illustrates an example process for shading an image in a sequence, in accordance with various embodiments;

FIG. **6** illustrates components of a distributed system that can be utilized to generate image content, according to at least one embodiment;

FIG. **7** illustrates an example data center system, according to at least one embodiment;

FIG. **8** is a block diagram illustrating a computer system, according to at least one embodiment;

FIG. **9** is a block diagram illustrating a computer system, according to at least one embodiment;

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

FIG. **11** illustrates a computer system, according to at least one embodiment;

FIG. **12**A illustrates a computer system, according to at least one embodiment;

FIG. **12**B illustrates a computer system, according to at least one embodiment;

FIG. **12**C illustrates a computer system, according to at least one embodiment;

FIG. **12**D illustrates a computer system, according to at least one embodiment;

FIGS. **12**E and **12**F illustrate a shared programming model, according to at least one embodiment;

FIG. **13** illustrates exemplary integrated circuits and associated graphics processors, according to at least one embodiment;

FIGS. **14**A-**14**B illustrate exemplary integrated circuits and associated graphics processors, according to at least one embodiment;

FIGS. **15**A-**15**B illustrate additional exemplary graphics processor logic according to at least one embodiment;

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

FIG. **17**A illustrates a parallel processor, according to at least one embodiment;

FIG. **17**B illustrates a partition unit, according to at least one embodiment;

FIG. **17**C illustrates a processing cluster, according to at least one embodiment;

FIG. **17**D illustrates a graphics multiprocessor, according to at least one embodiment;

FIG. **18** illustrates a multi-graphics processing unit (GPU) system, according to at least one embodiment;

FIG. **19** illustrates a graphics processor, according to at least one embodiment;

FIG. **20** is a block diagram illustrating a processor microarchitecture for a processor, according to at least one embodiment;

FIG. **21** illustrates a deep learning application processor, according to at least one embodiment;

FIG. **22** is a block diagram illustrating an example neuromorphic processor, according to at least one embodiment;

FIG. **23** illustrates at least portions of a graphics processor, according to one or more embodiments;

FIG. **24** illustrates at least portions of a graphics processor, according to one or more embodiments;

FIG. **25** illustrates at least portions of a graphics processor, according to one or more embodiments; and

FIG. **26** is a block diagram of a graphics processing engine of a graphics processor in accordance with at least one embodiment.

DETAILED DESCRIPTION

In the following description, various embodiments will be described. For purposes of explanation, specific configurations and details are set forth in order to provide a thorough understanding of the embodiments. However, it will also be apparent to one skilled in the art that the embodiments may be practiced without the specific details. Furthermore, well-known features may be omitted or simplified in order not to obscure the embodiment being described.

The systems and methods described herein may be used by, without limitation, non-autonomous vehicles, semi-autonomous vehicles (e.g., in one or more advanced driver assistance systems (ADAS)), piloted and un-piloted robots or robotic platforms, warehouse vehicles, off-road vehicles, vehicles coupled to one or more trailers, flying vessels, boats, shuttles, emergency response vehicles, motorcycles, electric or motorized bicycles, aircraft, construction vehicles, trains, underwater craft, remotely operated vehicles such as drones, and/or other vehicle types. Further, the systems and methods described herein may be used for a variety of purposes, by way of example and without limitation, for machine control, machine locomotion, machine driving, synthetic data generation, model training, perception, augmented reality, virtual reality, mixed reality, robotics, security and surveillance, simulation and digital twinning, autonomous or semi-autonomous machine applications, deep learning, environment simulation, object or actor simulation and/or digital twinning, data center processing, conversational AI, light transport simulation (e.g., ray-tracing, path tracing, etc.), collaborative content creation for 3D assets, cloud computing and/or any other suitable applications.

Disclosed embodiments may be comprised in a variety of different systems such as automotive systems (e.g., a control system for an autonomous or semi-autonomous machine, a perception system for an autonomous or semi-autonomous machine), systems implemented using a robot, aerial systems, medial systems, boating systems, smart area monitoring systems, systems for performing deep learning operations, systems for performing simulation operations, systems for performing digital twin operations, systems implemented using an edge device, systems incorporating one or more virtual machines (VMs), systems for performing synthetic data generation operations, systems implemented at least partially in a data center, systems for performing conversational AI operations, systems for performing light transport simulation, systems for performing collaborative content creation for 3D assets, systems implemented at least partially using cloud computing resources, and/or other types of systems.

Approaches in accordance with various illustrative embodiments allow for rendering (or other generation) of images, using approaches such as ray tracing, that have reduced noise with respect to other image generation approaches. In order to provide for smoothing between sequential frames of video, but avoid introducing lag into lighting effects, light information can be compared for regions (e.g., ReSTIR reservoirs) between consecutive rendered frames. Shading can be performed for the consecutive frames, and the results of the shading for corresponding reservoirs (or pixel grids/tiles) can be compared to compute gradient values, such as by using a single light sample for each tile. A filtering pass can be performed with respect to these gradients, such as to perform a bilateral blur process. This filtered-lower resolution grid version can be upscaled into a full resolution, screen-sized image and the gradients transformed into confidence values. These confidence values can be fed into a denoiser, or other post-process, to determine an extent to which to keep lighting data from the previous frame with respect to the current frame. For example, less lighting information can be used from the prior frame for a given pixel location if the confidence for that location is lower.

In at least one embodiment, these temporal gradients can be computed by reusing ReSTIR reservoirs (or other sampled light information) instead of random numbers. These reservoirs can correspond to statistical aggregates of lighting data that are accumulated over multiple frames and pixels. Temporal gradients can be computed using, for example, a separate ray tracing pass that runs after final shading. A gradient pass can compute the difference between shading results for the same surface using the same light sample on two consecutive frames. The light sample may refer to an illuminant but may not store any parameters of the illuminant, such that differences in shading due to a motion of the illuminant or change in its brightness (for example and without limitation) can still be captured.

In at least one embodiment, the computed gradients (or the confidence derived from them) may be used for at least two purposes. For example, these gradients can be used to control history rejection in the denoiser. Further, these gradients can be fed back into an algorithm, such as a ReSTIR algorithm, to control the history rejection in the temporal reservoir reuse pass. Rejecting the reservoir history can be desirable in cases when some of the history samples became invalid due to significant changes in the lighting environment that cannot be easily tracked, which results in local bias.

Gradients can be fed back into a ReSTIR algorithm in multiple ways. For example, a reservoir can contain statistical aggregates representing long histories of light samples. When sudden lighting changes occur, these statistical aggregates can become less valuable, and in some situations even

potentially detrimental. In one or more non-limiting embodiments, pixels with strong gradients may need to: have their reservoirs invalidated (similar to the process at disocclusions), have temporally reused reservoirs down-weighted (to emphasize current frame samples), rely more on spatial reuse, take more candidate samples from the current frame lighting, and/or adjust the sampling probability distribution function (PDF) for candidates to preferentially sample from any lights that recently changed.

Such a process can advantageously generate content for a variety of different applications and use cases. These can include, by way of example and without limitation, use in conversational systems to provide a view of a participant to a conversation. This would apply broadly to any situation where a computer system is interacting with a human via verbal or written communication. Such approaches can also be used to generate novel content for applications such as gaming, animation, special effects, or virtual/mixed/enhanced reality experiences. Such approaches can also be beneficial when generating environments, 3D object representations, or characters for applications or services having a visual aspect or component. Generative models can be used to synthesize other types of content as well, as may relate to speech or music. Generative models can be used as parts of systems to perform more complex tasks as well, as may relate to upsampling or super-resolution, image to image resolution, or 3D/4D complex animation or shape generation.

Variations of this and other such functionality can be used as well within the scope of the various embodiments as would be apparent to one of ordinary skill in the art in light of the teachings and suggestions contained herein.

FIG. **1** illustrates an example pipeline **100** for shading objects in a sequence of images, video frames, or other displayable content in accordance with various embodiments. In this example, frame data **102** can be received for each of a sequence of frames (or images) to be output. As part of a rendering process, for example, locations of various objects can be determined, along with the corresponding appearance data for those objects (e.g., shape, size, texture, reflectivity, and the like). One or more virtual light sources can be used to virtually illuminate these objects using one or more shaders **104**. The illumination may change between frames, due to factors such as movement or adjustment of the object(s) being illuminated, movement or adjustment of the one or more light sources, or changes in other aspects or properties of a scene being rendered. In various rendering processes, using only the illumination or appearance data can result in noisy images, such that it can be desirable to perform some amount of smoothing or blending across frames. Using too much historical appearance data from prior frames, however, can result in lag or delayed motion across a sequence of image frames, particularly for lighting effects such as shadows or reflections. Further, different light sources may be incident on a surface (directly or indirectly) for different frames.

Accordingly, a pipeline **100** such as that illustrated in FIG. **1** can attempt to dynamically determine how much historical data to use from at least one prior frame in a sequence with respect to current data for a current frame in the sequence, such as for lighting information. This determination can be made down to an individual pixel level in various embodiments. In this example the pipeline **100** is shown as a linear or sequential arrangement of components, while such a system can also be thought of as a multi-pass system that processes image data in a number of passes or phases, such as three phases of rendering. In this example, the frame data **102** for a current frame is passed to a shader **104** that can determine illumination of one or more objects in this frame based on one or more virtual light sources. The shader **104** can take shading information for each frame and write that information to a cache **106** for use in determining illumination for at least one subsequent frame in this sequence.

The shader **104** can determine luminance information for light that is reflected or incident on a surface from a given light sample. This example pipeline **100** can attempt to determine gradients that are representative of differences in illumination between a current frame and a prior frame in this sequence of frames. This can include determining, for a sample location on a light source, how that sample location affects the same surface on the current frame and the previous frame, taking into account that the light source and the surface could have different positions, orientations, or environment states, etc. Computing such a gradient for each pixel location of a high resolution image can be quite computationally expensive, such that it can be desirable to instead determine these gradients for only a portion, fraction, or subset of these pixel locations.

In this example, the pixel locations can be allocated to an array of tiles (or strata) of similar (or different) sizes, such as tiles of 3×3 pixels. A region selector **108** or other such component or process can determine tiling for a frame based on selected or provided tile criteria, such as tile size or number. Information for these selected tiles can be passed to a comparator **110** or other such component or process, which can select a single sample for each tile to be compared between a current frame and a previous frame. Various approaches can be used to select a single sample value for a given tile, such as by selecting the sample value that represents the brightest illumination or highest luminance value, among other such options. In some embodiments, a selection algorithm is used that weights sample values for pixel locations in a tile by brightness, and then selects from among those weighted pixel locations or luminance values. The comparator **110** can then, for every tile in a frame of pixels, select to use the light that was used to shade the selected pixel location for the current frame or the previous light that was used to shade this surface for a previous frame. Selecting a light from the previous frame for gradient computation can be beneficial in various cases, such as when an important light was affecting the surface on the previous frame but is no longer affecting that surface. For example, the light could have been turned off, or moved in a such way that the surface is now occluded from the light, such as where there is now a shadow. In such cases, the light is not selected for shading the surface on the current frame, but a temporal gradient can be used to ensure that the previously-accumulated lighting data is invalidated.

Once the light from the current frame or previous frame is selected for a tile, the luminance can be computed for the other frame in this sequence using this selected light. If a light for a current frame is selected then that light can be used to determine luminance values for the corresponding surface in the prior frame. The comparator **110** can compare the luminance values for the current and prior frame using the selected light and use this to calculate or determine the luminance gradient for that tile (or difference between the two corresponding luminance values). In a multi-pass implementation, these tile-specific gradients can be the output of the first pass.

In at least one embodiment, these gradients can be written into a texture, or a low-resolution image of gradient values. For a 3×3 tiling approach, this can result in a resolution of this gradient image that is $\frac{1}{9}^{th}$ of the resolution on the frame

data being rendered. This low-resolution gradient image can be provided as input to a filtering component **112** or process. A filter, such as a spatial blur filter, can be applied over this lower resolution image of gradient values. Any appropriate blur filter can be used, such as a bilateral blur filter, which effectively blurs across the same surfaces in the current and previous frames. A filtered, low-resolution gradient image can then be an output of a second pass in some embodiments.

This filtered, low-resolution gradient frame can be provided as input to an upscaler **114** to upscale the frame to a target resolution, such as an output resolution for an output frame **120** to be generated. Any appropriate upscaling process can be used, such as a spatial upscale or bilateral upscale. In such an upscale, nearby gradients for a surface can be used to interpolate gradient values for individual pixel locations corresponding to similar surfaces. As part of the upscaling process, and/or by using a separate confidence transform **116** component or process, these gradient values can be transformed into confidence values. This can be a tunable transform that can be based on a power function or similar such approach. A result of this second upscaling and transform can be a frame (or image or texture) of confidence values at a target output resolution. This frame of confidence values can then be provided as input to a denoiser **118**, which can determine whether to keep or reject current or previous frame luminance information for any given pixel location, or group of locations, or an extent to which to keep and blend these values. For example, a denoiser might decide for a given pixel location to use the confidence signal to reject the previous frame information where confidence is low and keep the previous frame information where the confidence is high. Once determined, an output frame **120** can be generated for each input frame that represents the selected luminance information. Using such a dynamic approach, these output frames should be substantially free of noise (at least noise due to illumination) and should have minimal lag due to luminance smoothing.

FIG. **2** illustrates an example approach to gradient sampling that can be used in accordance with at least one embodiment. Such an approach can attempt to determine an adaptive per-pixel weight computed from temporal gradients. In this example, a sparse subset of surface and shading samples can be reprojected—as illustrated in grids **202**, **204**, and **206**, with the reprojected surface samples being merged into a visibility buffer as illustrated in grids **210** and **212**. Combining the reprojected shading samples with the newly shaded samples yields gradient samples as illustrated in grids **208** and **214**. In this example, there is at most one gradient sample per 3×3 tile/stratum. A reconstruction step can transform these scattered and noisy samples into a dense, denoised gradient image **216**.

Temporal reuse of such information can use temporal reprojection of samples from at least the previous frame. In at least one embodiment, this reprojection can be performed in screen space such that there may be no need to maintain information about visible surface samples. In a first render pass, a g-buffer or visibility buffer can be generated. For each pixel j in frame i, this yields a surface sample $G_{i,j}$ providing access to surface attributes such as world-space position, normal and diffuse albedo. The g-buffer can store each attribute explicitly, whereas the visibility buffer can store information about the triangle intersection. A deferred full-screen pass, as may be implemented in a fragment or compute shader, for example, can then apply the shading function $f_i(G_{i,j})$ to compute a color for pixel j.

Forward or backward projection can be used in different embodiments. Forward projection can carry a surface sample $G_{i-1,j}$ from the previous frame i−1 to the current frame i. The forward projected surface sample $\vec{G}_{i-1,j}$ can provide access to all surface attributes in the current frame for the same point on the surface. Such reprojection can be simpler to perform using a visibility buffer in at least one embodiment. Having access to the new world space location through $\vec{G}_{i-1,j}$ the coordinate transforms for frame i can yield the corresponding screen space location in the current frame. In particular, the index of pixel $\vec{j}$ can be computed that covers the surface sample in the current frame, as illustrated in grid **204**. A backprojected surface sample can also be used in some embodiments. Backward projection using motion vectors may exhibit better compatibility with various rendering pipelines. In an example implementation, a compute pass can be performed that examines every pixel in a current frame and its associated motion vector, and finds a matching pixel in the previous frame. Specifically, the gradient computation pass can run one compute shader thread per 3×3 tile of pixels, for example, then identify a matching surface on the previous frame for each pixel and select a single light for the tile from the nine lights in the current frame and nine lights in the previous frame (or fewer).

In each frame, a first step can be to render a new visibility buffer and to generate new seeds as illustrated in grid **210**. Rather than affording one sample of the temporal gradient per pixel. In at least one embodiment, part of the shading budget can be repurposed to evaluate gradient samples sparsely. In another embodiment, an extra pass can be used to compute the gradients, such that a gradient can be computed for each pixel instead of for each tile. Such an approach can produce higher quality results but with additional expense, such that gradients may be computed for tiles instead of for pixels in at least some embodiments as a type of performance optimization. In at least one embodiment a stratum of 3×3 pixels can be used, although strata of other sizes can be used as well in various embodiments. In each of these strata, one pixel j can be selected from the previous frame that is to be reprojected as illustrated by grid **202**. Through this stratified sampling, aliasing can be traded for temporally-incoherent noise.

Forward or backward projection can be applied to these samples to determine their screen space locations in the current frame, as illustrated in grid **204**. The depth buffer of the current frame can be used to discard reprojected surface samples which are occluded in the current frame. The other surface samples and seeds can be merged into the new visibility buffer at the appropriate pixel $\vec{j}$ as illustrated in grid **212**. Per stratum, this example process may allow for no more than one gradient sample. However, the reprojection may map multiple samples to the same stratum. Such conflicts may be resolved efficiently using, for example, GPU atomics. The sample that finishes the reprojection computations first can be merged into the visibility buffer. The shading samples can be reprojected in the same manner as the visibility information, without interpolation, as illustrated in grid **206**. The shading function of the current frame can be applied to all surface samples in the visibility buffer. In particular, this yields shading samples for the reprojected surface samples as illustrated in grid **214**. A subtraction can then produce gradient samples, as illustrated in grid **208**. It can be noted that all shading samples for the reprojected surface samples can be valid shading samples for the new frame. They sample a visible surface within the pixel

footprint, only the sample location is not at the pixel center. Such an approach does not introduce gaps into the frame buffer that otherwise would need to be filled. Nonetheless, shading samples resulting from new surface samples may be beneficial in at least one embodiment.

In at least some instances, the gradient samples will not only be sparse and irregular but also noisy. A reconstruction can be performed to obtain a dense, denoised estimate of the temporal gradient. Such reconstruction can be efficient and edge-preserving, and can support large filter regions to obtain a sufficient number of samples per pixel. In at least one embodiment, an edge-aware wavelet transform can be applied, which can perform a cross-bilateral transformation over multiple iterations. To achieve a large filter region efficiently, taps can be spread apart further in each iteration. Such gradient reconstruction can be joint-bilateral, where the luminance is filtered and simultaneously used to derive filter weights used for reconstruction of the gradient and luminance samples. The shading and gradient samples can be stored into a regular grid at stratum resolution.

In such an approach where gradients are only determined for a sampling of pixel locations, there may be at least some amount of noise or uncertainty to the data due to not all data being used for analysis. Accordingly, some lighting data may not be captured or some lighting data may be overemphasized for a given tile. Processing with a spatial blur can help to reduce the amount of noise in the gradient data. Because this gradient data is used for confidence values, however, there is no need for sharp precision in many instances as the process will produce regions of similar confidence in many instances, which indicate whether to use luminance information from a current frame or a previous frame, where in many instances those values may not be drastically different (else there would be a low confidence value and the prior value might be discarded.

FIGS. **3**A, **3**B, and **3**C illustrate images with confidence values that can be determined in accordance with various embodiments. These images illustrate a rotating sphere **302** with an internal light and a number of openings, such that the light propagating from those openings will change between frames as a result of the sphere rotating. FIG. **3**A illustrates an image **300** with values **304** for a confidence channel computed from unfiltered gradients. It can be seen that there is a significant amount of noise in the confidence values **304** at various pixel positions. FIG. **3**B illustrates an image **330** including confidence values **332** after spatial and temporal filtering resulting from light coming the holds in the rotating sphere. In these images, the confidence values are illustrated using a reverse heat map to highlight the regions where the history is to be invalidated. FIG. **3**C illustrates an image **360** representing the importance of using ray-traced bias correction in temporal resampling when computing gradients. When ray traced bias correction is not used, the lighting signal (e.g., a ReSTIR lighting signal) in regions with dynamic shadows is significantly dimmer, which can result in smaller gradients and higher computed confidence values. It can therefore be beneficial in various situations to implement ray-traced temporal bias correction.

FIG. **4** illustrates components of an example rendering pipeline **400** that can be utilized to render images in accordance with various embodiments. In this example, an application **402** is running on a central processing unit (CPU) **402**, where that application includes instructions that can be stored in system memory **404** and executed by the CPU. This application can be, for example, a video game or animation application or process that provides data about an image to be rendered. In this example, data for rendering an image can be provided, via an application programming interface (API) runtime **406** or other such interface mechanism, to a graphics processing unit (GPU) **410**. As mentioned, for at least some types of rendering or tasks a GPU can provide improved performance relative to a CPU, particularly for a large number of small parallel tasks, such as may be utilized for rendering of an image, particularly where hardware acceleration can be applied to at least some of those tasks. Instructions can be stored in GPU memory **412** until they are selected or scheduled for execution. In this example, the data and instructions can be passed to one or more shaders **414**, which may include one or more vertex shading components **416** for adding effects to objects in a scene or environment, often a 3D environment, by determining the vertex data for one or more objects in a scene and then performing various mathematical operations on that object vertex data. In this example, the vertex data is passed to one or more geometry components **418**, which can perform various tasks such as at least some of those described herein. In this example, this can include tasks such as performing model and view transformations, performing vertex shading and illumination, performing data projection, performing clipping or culling of data based on geometry, and determining an appropriate scene map, among other such tasks. For shading or illumination tasks described herein that can be based at least in part upon cumulative distribution functions, these tasks can be performed within the shaders **414** of one or more GPU on a single computing device or distributed across multiple devices. After these various geometry-based tasks are performed, the resulting data can be passed to a shading component **420** which can perform tasks such as individual pixel shading in order to generate output image data for various pixels. This data can then be cached in one or more buffers **424** in (or external to) GPU memory **422** (which can be the same as, or separate from, GPU memory **412**) until it is time to transmit that information for presentation via at least one display **430** or other such mechanism, as may be attached to, or contained within, at least one computing device or system, which may be a same computing device or system as includes the CPU **402** and GPU **410**. This process can be performed for each image to be generated, as may make up a sequence of video frames to be presented via display **430**. As discussed elsewhere herein, display **430** is not limited to a conventional video display device, such as a television, monitor, or touch screen, but can also include a projector, VR/AR/MR headset, wearable display, holographic display, and the like. As will be discussed in more detail with respect to FIG. **6**, such components may be contained in a client device for which the video is to be displayed, a server to transmit the content to a client device, or a third party system that is to generate image data on behalf of a client or server device, among other such options. In this example, tasks such as gradient computation and light contribution determination can be performed in one or more shaders **414**. In at least one embodiment, shading component **420** can perform such tasks relating to individual pixel shading in order to generate output image data for various pixels.

In at least one embodiment, a temporal gradient can be defined as the difference between the shading results of the same surface using the same light sample on two consecutive frames. Such gradients can capture changes in the lighting environment for a surface, as may relate to lights moving relative to the surface, lights changing their intensity, or lights becoming shadowed or un-shadowed. These gradients may capture the difference in shading results due to the view vector changing because of camera motion, or

not capture that, depending on the implementation; when using modern denoisers (e.g., NRD denoisers), capturing the view vector changes would not be necessary. In at least one embodiment, gradient computation should not capture any changes due to the subpixel camera jitter that results in slight material variation in the same pixel when the camera is static.

Temporal gradients can be computed using a separate compute or ray tracing (or other light transport simulation) pass that runs after final shading. In at least one embodiment, a temporal resampling pass or the fused kernel can save the screen-space position of the pixel whose reservoir was used as the temporal light sample. The luminance of the final shading results from the current and the previous frames can be used as input. Luminance can be computed from the diffuse and specular lighting textures if those textures store unmodified colors. Alternatively, light transport simulation results from implementing a bidirectional reflectance distribution function (BRDF) may also be stored into the same textures, so the sampled lighting luminance values may be stored separately in the textures.

In at least one embodiment, a gradient pass may be implemented with two phases of execution. In a first phase, at least one pixel is selected for gradient computation from the stratum. One approach is to select the pixel that has a valid history and the highest luminance out of all pixels in the stratum on either the current or the previous frame. Other heuristics can be used as well, such as that stochastically select a pixel using the luminance values as probability mass function values. For the selected pixel(s), either the current or previous frame light sample may be used, such as may correspond to whichever is the brightest and thus likely to produce the highest gradient value. During a second phase, for the selected pixel, shading in the "other frame" can be performed. This means that, if a selected light sample from the current frame is available as reference, the current surface is used, its position in the previous frame is reconstructed, and that reconstructed surface is shaded using the light information and the bounding volume hierarchy (BVH) of the previous scene. If a light sample from the previous frame is selected, the previous surface can be used as reference, its position in the current frame is reconstructed or, as may be more reliable in at least some situations, the actual position of the surface in the selected pixel on the current frame can be used. When selecting a pixel for gradient computation, exact surface positions on the current frame can be used. The surface can be shaded using the light information and BVH of the scene in the current frame. The difference between the current and previous shading results can then be used as the gradient value.

In at least one embodiment, gradients can be converted into, or otherwise used to determine, confidence values. Executing the gradients pass can produce a sparse and low-resolution signal, such as at around one-ninth ($1/9^{th}$) of a target screen resolution, that stores luminance differences and absolute values. The gradients can be filtered spatially using a wide blur, where the blur size and exact parameters may vary. In one example, a direct illumination sample application uses a 4-pass Atrous filter with a 3×3 kernel, which results in a 31 pixel kernel radius (in gradient space). The blur could be bilateral and take surface normals and positions into account, if desired. It can be noted that the luminance differences and absolute values can be filtered independently, such that a small, local change in a bright region may be unlikely to result in history invalidation. The filtered gradients can then be normalized, such as where luminance differences are divided by the absolute luminance values, and then converted into (0-1) confidence using a simple function. This signal can already be fed into the confidence input of the denoiser in many instances with success.

These gradients may often be noisy even after the spatial filtering, which may result in patchy history invalidation. Further, singular events such as a light turning on or off may only create non-zero gradients for a single frame. The spatiotemporal nature of ReSTIR can lead to noisy and locally-biased lighting on that first frame after a significant change. If the denoiser history is reset momentarily and then accumulation starts from scratch, that local bias has a significant weight in the history, resulting for example in a "black dip" effect around a light that has turned off.

In at least one embodiment, an approach to overcoming both noisy confidence and local bias can involve applying a filter, such as a short-history temporal filter, to the confidence input of the denoiser. When the temporal filter is tuned correctly, denoiser history invalidations can occur smoothly over a few frames and not abruptly. In order to reduce the GPU workload resulting from such a correction, approaches in accordance with at least one embodiment can leverage the observation that temporal resampling can frequently select the sample from the previous frame. The correction can trace a visibility ray between the previous frame surface and the selected light sample on the previous frame. This visibility has already been computed on the previous frame, such that if invisible samples are discarded in final shading, the samples would not be selected in the temporal resampling part because they no longer exist. With this assumption, temporal resampling can skip tracing the visibility ray if the selected sample comes from the previous frame. In typical scenarios, this can reduce the number of rays traced by over 90%.

FIG. **5** illustrates an example process **500** for rendering an image in a sequence of images that can be performed in accordance with at least one embodiment. It should be understood for this and other processes presented herein that there may be additional, fewer, or alternative steps performed in similar or alternative orders, or at least partially in parallel, within the scope of the various embodiments unless otherwise explicitly stated. In this example process, a current image in a sequence of images is rendered **502**. This can include determining shading information for this image from a light source of this current image (or frame). The image data (or image space) can be divided **504** into a plurality of tiles, such as groups of 3×3 pixels, and a single lighting sample selected for each tile to be compared to a corresponding sample of a prior image in this sequence. Sample data from the current and prior images can be analyzed in order to select **506** either the light from the current frame or the prior image to use to determine luminance values to compare for each tile. A light gradient can be generated **508** for each of the tiles (or other pixel regions) of the current image with respect to the corresponding tile of the prior image. Spatial blurring can be performed **510** on these gradients in order to produce a lower-resolution, blurred gradient image (or texture). This blurred gradient image can then be upscaled **512**, such as to a target output resolution that will include one gradient value per pixel location, such as may be determined using interpolation in the upscaling process. The gradients for these individual pixel locations can be transformed **514**, during or after this upscaling, to pixel-specific confidence values. A denoiser accepting these confidence values as input can be used to determine **516** an extent to which to use the lighting data from the previous image or the current image to determine pixel data values for

the individual pixel locations of the current image at the target image resolution. This can include, for example, adjusting a weight for one or more historical light values, including whether or not to consider a prior light sample at all for a current image or frame. For example, the weight may be a blending weight indicating how much the lighting data from the prior frame should be used versus the lighting data for a current frame for a given pixel location. In at least one embodiment, an algorithm used to determine this blending weight can also have one or more adjustment factors built in, enabling a user to balance reduced noise with reduced lighting lag. For example, an exponential power function can be used that blends values in a non-linear space such that for any change in lighting the confidence value can decrease quickly but then increase back to high confidence slower. Computed confidence values are used to help guide the denoiser to produce higher quality results, such as to reject historical lighting data for more dynamic lighting changes.

As discussed, various approaches presented herein are lightweight enough to execute on a client device, such as a personal computer or gaming console, in real time or near real time. Such processing can be performed on content that is generated on that client device or received from an external source, such as streaming content received over at least one network. The source can be any appropriate source, such as a game host, streaming media provider, third party content provider, or other client device, among other such options. In some instances, the processing and/or rendering of this content may be performed by one of these other devices, systems, or entities, then provided to the client device (or another such recipient) for presentation or another such use.

As an example, FIG. **6** illustrates an example network configuration **600** that can be used to provide, generate, modify, encode, process, and/or transmit data or other such content. In at least one embodiment, a client device **602** can generate or receive data for a session using components of a content application **604** on client device **602** and data stored locally on that client device. In at least one embodiment, a content application **624** executing on a server **620** (e.g., a cloud server or edge server) may initiate a session associated with at least client device **602**, as may utilize a session manager and user data stored in a user database **634**, and can cause content **632** to be determined by a content manager **626**. A content manager **626** may work with a content generator **628** to generate or synthesize content to be provided for presentation via the client device **602**. In at least one embodiment, this content generator **628** can work with a renderer **630**, or rendering engine, to generate specific types or instances of content. At least a portion of the generated data or content may be transmitted to the client device **602** using an appropriate transmission manager **622** to send by download, streaming, or another such transmission channel. An encoder may be used to encode and/or compress at least some of this data before transmitting to the client device **602**. In at least one embodiment, the client device **602** receiving such content can provide this content to a corresponding content application **604**, which may also or alternatively include a graphical user interface **610**, content generator **612**, and renderer **614** for use in providing content for presentation via the client device **602**. A decoder may also be used to decode data received over the network (s) **640** for presentation via client device **602**, such as image or video content through a display **606** and audio, such as sounds and music, through at least one audio playback device **608**, such as speakers or headphones. In at least one embodiment, at least some of this content may already be stored on, rendered on, or accessible to client device **602** such that transmission over network **640** is not required for at least that portion of content, such as where that content may have been previously downloaded or stored locally on a hard drive or optical disk. In at least one embodiment, a transmission mechanism such as data streaming can be used to transfer this content from server **620**, or user database **634**, to client device **602**. In at least one embodiment, at least a portion of this content can be obtained or streamed from another source, such as a third party service **660** or other client device **650**, that may also include a content application **662** for generating or providing content. In at least one embodiment, portions of this functionality can be performed using multiple computing devices, or multiple processors within one or more computing devices, such as may include a combination of CPUs and GPUs.

In this example, these client devices can include any appropriate computing devices, as may include a desktop computer, notebook computer, set-top box, streaming device, gaming console, smartphone, tablet computer, VR headset, AR goggles, wearable computer, or a smart television. Each client device can submit a request across at least one wired or wireless network, as may include the Internet, an Ethernet, a local area network (LAN), or a cellular network, among other such options. In this example, these requests can be submitted to an address associated with a cloud provider, who may operate or control one or more electronic resources in a cloud provider environment, such as may include a data center or server farm. In at least one embodiment, the request may be received or processed by at least one edge server, that sits on a network edge and is outside at least one security layer associated with the cloud provider environment. In this way, latency can be reduced by enabling the client devices to interact with servers that are in closer proximity, while also improving security of resources in the cloud provider environment.

In at least one embodiment, such a system can be used for performing graphical rendering operations. In other embodiments, such a system can be used for other purposes, such as for providing image or video content to test or validate autonomous machine applications, or for performing deep learning operations. In at least one embodiment, such a system can be implemented using an edge device, or may incorporate one or more Virtual Machines (VMs). In at least one embodiment, such a system can be implemented at least partially in a data center or at least partially using cloud computing resources.

Data Center

FIG. **7** illustrates an example data center **700**, in which at least one embodiment may be used. In at least one embodiment, data center **700** includes a data center infrastructure layer **710**, a framework layer **720**, a software layer **730** and an application layer **740**.

In at least one embodiment, as shown in FIG. **7**, data center infrastructure layer **710** may include a resource orchestrator **712**, grouped computing resources **714**, and node computing resources ("node C.R.s") **716**(**1**)-**716**(N), where "N" represents a positive integer (which may be a different integer "N" than used in other figures). In at least one embodiment, node C.R.s **716**(**1**)-**716**(N) may include, but are not limited to, any number of central processing units ("CPUs") or other processors (including accelerators, field programmable gate arrays (FPGAs), graphics processors, etc.), memory storage devices **718**(**1**)-**718**(N) (e.g., dynamic

read-only memory, solid state storage or disk drives), network input/output ("NW I/O") devices, network switches, virtual machines ("VMs"), power modules, and cooling modules, etc. In at least one embodiment, one or more node C.R.s from among node C.R.s **716**(1)-**716**(N) may be a server having one or more of above-mentioned computing resources.

In at least one embodiment, grouped computing resources **714** may include separate groupings of node C.R.s housed within one or more racks (not shown), or many racks housed in data centers at various geographical locations (also not shown). In at least one embodiment, separate groupings of node C.R.s within grouped computing resources **714** may include grouped compute, network, memory or storage resources that may be configured or allocated to support one or more workloads. In at least one embodiment, several node C.R.s including CPUs or processors may grouped within one or more racks to provide compute resources to support one or more workloads. In at least one embodiment, one or more racks may also include any number of power modules, cooling modules, and network switches, in any combination.

In at least one embodiment, resource orchestrator **712** may configure or otherwise control one or more node C.R.s **716**(1)-**716**(N) and/or grouped computing resources **714**. In at least one embodiment, resource orchestrator **712** may include a software design infrastructure ("SDI") management entity for data center **700**.

In at least one embodiment, as shown in FIG. **7**, framework layer **720** includes a job scheduler **722**, a configuration manager **724**, a resource manager **726** and a distributed file system **728**. In at least one embodiment, framework layer **720** may include a framework to support software **732** of software layer **730** and/or one or more application(s) **742** of application layer **740**. In at least one embodiment, software **732** or application(s) **742** may respectively include web-based service software or applications, such as those provided by Amazon Web Services, Google Cloud and Microsoft Azure. In at least one embodiment, framework layer **720** may be, but is not limited to, a type of free and open-source software web application framework such as Apache Spark™ (hereinafter "Spark") that may utilize distributed file system **728** for large-scale data processing (e.g., "big data"). In at least one embodiment, job scheduler **722** may include a Spark driver to facilitate scheduling of workloads supported by various layers of data center **700**. In at least one embodiment, configuration manager **724** may be capable of configuring different layers such as software layer **730** and framework layer **720** including Spark and distributed file system **728** for supporting large-scale data processing. In at least one embodiment, resource manager **726** may be capable of managing clustered or grouped computing resources mapped to or allocated for support of distributed file system **728** and job scheduler **722**. In at least one embodiment, clustered or grouped computing resources may include grouped computing resources **714** at data center infrastructure layer **710**. In at least one embodiment, resource manager **726** may coordinate with resource orchestrator **712** to manage these mapped or allocated computing resources.

In at least one embodiment, software **732** included in software layer **730** may include software used by at least portions of node C.R.s **716**(1)-**716**(N), grouped computing resources **714**, and/or distributed file system **728** of framework layer **720**. In at least one embodiment, one or more types of software may include, but are not limited to, Internet web page search software, e-mail virus scan software, database software, and streaming video content software.

In at least one embodiment, application(s) **742** included in application layer **740** may include one or more types of applications used by at least portions of node C.R.s **716**(1)-**716**(N), grouped computing resources **714**, and/or distributed file system **728** of framework layer **720**. In at least one embodiment, one or more types of applications may include, but are not limited to, any number of a genomics application, a cognitive compute, application and a machine learning application, including training or inferencing software, machine learning framework software (e.g., PyTorch, TensorFlow, Caffe, etc.) or other machine learning applications used in conjunction with one or more embodiments.

In at least one embodiment, any of configuration manager **724**, resource manager **726**, and resource orchestrator **712** may implement any number and type of self-modifying actions based on any amount and type of data acquired in any technically feasible fashion. In at least one embodiment, self-modifying actions may relieve a data center operator of data center **700** from making possibly bad configuration decisions and possibly avoiding underutilized and/or poor performing portions of a data center.

In at least one embodiment, data center **700** may include tools, services, software or other resources to train one or more machine learning models or predict or infer information using one or more machine learning models according to one or more embodiments described herein. For example, in at least one embodiment, a machine learning model may be trained by calculating weight parameters according to a neural network architecture using software and computing resources described above with respect to data center **700**. In at least one embodiment, trained machine learning models corresponding to one or more neural networks may be used to infer or predict information using resources described above with respect to data center **700** by using weight parameters calculated through one or more training techniques described herein.

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

Computer Systems

FIG. **8** is a block diagram illustrating an exemplary computer system, which may be a system with interconnected devices and components, a system-on-a-chip (SOC) or some combination thereof formed with a processor that may include execution units to execute an instruction, according to at least one embodiment. In at least one embodiment, a computer system **800** may include, without limitation, a component, such as a processor **802** to employ execution units including logic to perform algorithms for process data, in accordance with present disclosure, such as in embodiment described herein. In at least one embodiment, computer system **800** may include processors, such as PENTIUM® Processor family, Xeon™ Itanium®, XScale™ and/or StrongARM™, Intel® Core™, or Intel® Nervana™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems

(including PCs having other microprocessors, engineering workstations, set-top boxes and like) may also be used. In at least one embodiment, computer system **800** may execute a version of WINDOWS operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems (UNIX and Linux, for example), embedded software, and/or graphical user interfaces, may also be used.

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

In at least one embodiment, computer system **800** may include, without limitation, processor **802** that may include, without limitation, one or more execution units **808** to perform machine learning model training and/or inferencing according to techniques described herein. In at least one embodiment, computer system **800** is a single processor desktop or server system, but in another embodiment, computer system **800** may be a multiprocessor system. In at least one embodiment, processor **802** may include, without limitation, a complex instruction set computer ("CISC") microprocessor, a reduced instruction set computing ("RISC") microprocessor, a very long instruction word ("VLIW") microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such as a digital signal processor, for example. In at least one embodiment, processor **802** may be coupled to a processor bus **810** that may transmit data signals between processor **802** and other components in computer system **800**.

In at least one embodiment, processor **802** may include, without limitation, a Level 1 ("L1") internal cache memory ("cache") **804**. In at least one embodiment, processor **802** may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor **802**. Other embodiments may also include a combination of both internal and external caches depending on particular implementation and needs. In at least one embodiment, a register file **806** may store different types of data in various registers including, without limitation, integer registers, floating point registers, status registers, and an instruction pointer register.

In at least one embodiment, execution unit **808**, including, without limitation, logic to perform integer and floating point operations, also resides in processor **802**. In at least one embodiment, processor **802** may also include a microcode ("ucode") read only memory ("ROM") that stores microcode for certain macro instructions. In at least one embodiment, execution unit **808** may include logic to handle a packed instruction set **809**. In at least one embodiment, by including packed instruction set **809** in an instruction set of a general-purpose processor, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using packed data in processor **802**. In at least one embodiment, many multimedia applications may be accelerated and executed more efficiently by using a full width of a processor's data bus for performing operations on packed data, which may eliminate a need to transfer smaller units of data across that processor's data bus to perform one or more operations one data element at a time.

In at least one embodiment, execution unit **808** may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. In at least one embodiment, computer system **800** may include, without limitation, a memory **820**. In at least one embodiment, memory **820** may be a Dynamic Random Access Memory ("DRAM") device, a Static Random Access Memory ("SRAM") device, a flash memory device, or another memory device. In at least one embodiment, memory **820** may store instruction(s) **819** and/or data **821** represented by data signals that may be executed by processor **802**.

In at least one embodiment, a system logic chip may be coupled to processor bus **810** and memory **820**. In at least one embodiment, a system logic chip may include, without limitation, a memory controller hub ("MCH") **816**, and processor **802** may communicate with MCH **816** via processor bus **810**. In at least one embodiment, MCH **816** may provide a high bandwidth memory path **818** to memory **820** for instruction and data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH **816** may direct data signals between processor **802**, memory **820**, and other components in computer system **800** and to bridge data signals between processor bus **810**, memory **820**, and a system I/O interface **822**. In at least one embodiment, a system logic chip may provide a graphics port for coupling to a graphics controller. In at least one embodiment, MCH **816** may be coupled to memory **820** through high bandwidth memory path **818** and a graphics/video card **812** may be coupled to MCH **816** through an Accelerated Graphics Port ("AGP") interconnect **814**.

In at least one embodiment, computer system **800** may use system I/O interface **822** as a proprietary hub interface bus to couple MCH **816** to an I/O controller hub ("ICH") **830**. In at least one embodiment, ICH **830** may provide direct connections to some I/O devices via a local I/O bus. In at least one embodiment, a local I/O bus may include, without limitation, a high-speed I/O bus for connecting peripherals to memory **820**, a chipset, and processor **802**. Examples may include, without limitation, an audio controller **829**, a firmware hub ("flash BIOS") **828**, a wireless transceiver **826**, a data storage **824**, a legacy I/O controller **823** containing user input and keyboard interfaces **825**, a serial expansion port **827**, such as a Universal Serial Bus ("USB") port, and a network controller **834**. In at least one embodiment, data storage **824** may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

In at least one embodiment, FIG. **8** illustrates a system, which includes interconnected hardware devices or "chips", whereas in other embodiments, FIG. **8** may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. **8** may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of computer system **800** are interconnected using compute express link (CXL) interconnects.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **9** is a block diagram illustrating an electronic device **900** for utilizing a processor **910**, according to at least one embodiment. In at least one embodiment, electronic device **900** may be, for example and without limitation, a notebook,

a tower server, a rack server, a blade server, a laptop, a desktop, a tablet, a mobile device, a phone, an embedded computer, or any other suitable electronic device.

In at least one embodiment, electronic device **900** may include, without limitation, processor **910** communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor **910** is coupled using a bus or interface, such as a I^2C bus, a System Management Bus ("SMBus"), a Low Pin Count (LPC) bus, a Serial Peripheral Interface ("SPI"), a High Definition Audio ("HDA") bus, a Serial Advance Technology Attachment ("SATA") bus, a Universal Serial Bus ("USB") (versions 1, 2, 3, etc.), or a Universal Asynchronous Receiver/Transmitter ("UART") bus. In at least one embodiment, FIG. **9** illustrates a system, which includes interconnected hardware devices or "chips", whereas in other embodiments, FIG. **9** may illustrate an exemplary SoC. In at least one embodiment, devices illustrated in FIG. **9** may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of FIG. **9** are interconnected using compute express link (CXL) interconnects.

In at least one embodiment, FIG. **9** may include a display **924**, a touch screen **925**, a touch pad **930**, a Near Field Communications unit ("NFC") **945**, a sensor hub **940**, a thermal sensor **946**, an Express Chipset ("EC") **935**, a Trusted Platform Module ("TPM") **938**, BIOS/firmware/flash memory ("BIOS, FW Flash") **922**, a DSP **960**, a drive **920** such as a Solid State Disk ("SSD") or a Hard Disk Drive ("HDD"), a wireless local area network unit ("WLAN") **950**, a Bluetooth unit **952**, a Wireless Wide Area Network unit ("WWAN") **956**, a Global Positioning System (GPS) unit **955**, a camera ("USB 3.0 camera") **954** such as a USB 3.0 camera, and/or a Low Power Double Data Rate ("LPDDR") memory unit ("LPDDR3") **915** implemented in, for example, an LPDDR3 standard. These components may each be implemented in any suitable manner.

In at least one embodiment, other components may be communicatively coupled to processor **910** through components described herein. In at least one embodiment, an accelerometer **941**, an ambient light sensor ("ALS") **942**, a compass **943**, and a gyroscope **944** may be communicatively coupled to sensor hub **940**. In at least one embodiment, a thermal sensor **939**, a fan **937**, a keyboard **936**, and touch pad **930** may be communicatively coupled to EC **935**. In at least one embodiment, speakers **963**, headphones **964**, and a microphone ("mic") **965** may be communicatively coupled to an audio unit ("audio codec and class D amp") **962**, which may in turn be communicatively coupled to DSP **960**. In at least one embodiment, audio unit **962** may include, for example and without limitation, an audio coder/decoder ("codec") and a class D amplifier. In at least one embodiment, a SIM card ("SIM") **957** may be communicatively coupled to WWAN unit **956**. In at least one embodiment, components such as WLAN unit **950** and Bluetooth unit **952**, as well as WWAN unit **956** may be implemented in a Next Generation Form Factor ("NGFF").

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **10** illustrates a computer system **1000**, according to at least one embodiment. In at least one embodiment, computer system **1000** is configured to implement various processes and methods described throughout this disclosure.

In at least one embodiment, computer system **1000** comprises, without limitation, at least one central processing unit ("CPU") **1002** that is connected to a communication bus **1010** implemented using any suitable protocol, such as PCI ("Peripheral Component Interconnect"), peripheral component interconnect express ("PCI-Express"), AGP ("Accelerated Graphics Port"), HyperTransport, or any other bus or point-to-point communication protocol(s). In at least one embodiment, computer system **1000** includes, without limitation, a main memory **1004** and control logic (e.g., implemented as hardware, software, or a combination thereof) and data are stored in main memory **1004**, which may take form of random access memory ("RAM"). In at least one embodiment, a network interface subsystem ("network interface") **1022** provides an interface to other computing devices and networks for receiving data from and transmitting data to other systems with computer system **1000**.

In at least one embodiment, computer system **1000**, in at least one embodiment, includes, without limitation, input devices **1008**, a parallel processing system **1012**, and display devices **1006** that can be implemented using a conventional cathode ray tube ("CRT"), a liquid crystal display ("LCD"), a light emitting diode ("LED") display, a plasma display, or other suitable display technologies. In at least one embodiment, user input is received from input devices **1008** such as keyboard, mouse, touchpad, microphone, etc. In at least one embodiment, each module described herein can be situated on a single semiconductor platform to form a processing system.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **11** illustrates a computer system **1100**, according to at least one embodiment. In at least one embodiment, computer system **1100** includes, without limitation, a computer **1110** and a USB stick **1120**. In at least one embodiment, computer **1110** may include, without limitation, any number and type of processor(s) (not shown) and a memory (not shown). In at least one embodiment, computer **1110** includes, without limitation, a server, a cloud instance, a laptop, and a desktop computer.

In at least one embodiment, USB stick **1120** includes, without limitation, a processing unit **1130**, a USB interface **1140**, and USB interface logic **1150**. In at least one embodiment, processing unit **1130** may be any instruction execution system, apparatus, or device capable of executing instructions. In at least one embodiment, processing unit **1130** may include, without limitation, any number and type of processing cores (not shown). In at least one embodiment, processing unit **1130** comprises an application specific integrated circuit ("ASIC") that is optimized to perform any amount and type of operations associated with machine learning. For instance, in at least one embodiment, processing unit **1130** is a tensor processing unit ("TPC") that is optimized to perform machine learning inference operations. In at least one embodiment, processing unit **1130** is a vision processing unit ("VPU") that is optimized to perform machine vision and machine learning inference operations.

In at least one embodiment, USB interface **1140** may be any type of USB connector or USB socket. For instance, in at least one embodiment, USB interface **1140** is a USB 3.0 Type-C socket for data and power. In at least one embodiment, USB interface **1140** is a USB 3.0 Type-A connector. In at least one embodiment, USB interface logic **1150** may

include any amount and type of logic that enables processing unit **1130** to interface with devices (e.g., computer **1110**) via USB connector **1140**.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **12**A illustrates an exemplary architecture in which a plurality of GPUs **1210**(**1**)-**1210**(N) is communicatively coupled to a plurality of multi-core processors **1205**(**1**)-**1205**(M) over high-speed links **1240**(**1**)-**1240**(N) (e.g., buses, point-to-point interconnects, etc.). In at least one embodiment, high-speed links **1240**(**1**)-**1240**(N) support a communication throughput of 4 GB/s, 30 GB/s, 80 GB/s or higher. In at least one embodiment, various interconnect protocols may be used including, but not limited to, PCIe 4.0 or 5.0 and NVLink 2.0. In various figures, "N" and "M" represent positive integers, values of which may be different from figure to figure. In at least one embodiment, one or more GPUs in a plurality of GPUs **1210**(**1**)-**1210**(N) includes one or more graphics cores (also referred to simply as "cores") **1500** as disclosed in FIGS. **15**A and **15**B. In at least one embodiment, one or more graphics cores **1500** may be referred to as streaming multiprocessors ("SMs"), stream processors ("SPs"), stream processing units ("SPUs"), compute units ("CUs"), execution units ("EUs"), and/or slices, where a slice in this context can refer to a portion of processing resources in a processing unit (e.g., 16 cores, a ray tracing unit, a thread director or scheduler).

In addition, and in at least one embodiment, two or more of GPUs **1210** are interconnected over high-speed links **1229**(**1**)-**1229**(**2**), which may be implemented using similar or different protocols/links than those used for high-speed links **1240**(**1**)-**1240**(N). Similarly, two or more of multi-core processors **1205** may be connected over a high-speed link **1228** which may be symmetric multi-processor (SMP) buses operating at 20 GB/s, 30 GB/s, 120 GB/s or higher. Alternatively, all communication between various system components shown in FIG. **12**A may be accomplished using similar protocols/links (e.g., over a common interconnection fabric).

In at least one embodiment, each multi-core processor **1205** is communicatively coupled to a processor memory **1201**(**1**)-**1201**(M), via memory interconnects **1226**(**1**)-**1226**(M), respectively, and each GPU **1210**(**1**)-**1210**(N) is communicatively coupled to GPU memory **1220**(**1**)-**1220**(N) over GPU memory interconnects **1250**(**1**)-**1250**(N), respectively. In at least one embodiment, memory interconnects **1226** and **1250** may utilize similar or different memory access technologies. By way of example, and not limitation, processor memories **1201**(**1**)-**1201**(M) and GPU memories **1220** may be volatile memories such as dynamic random access memories (DRAMs) (including stacked DRAMs), Graphics DDR SDRAM (GDDR) (e.g., GDDR5, GDDR6), or High Bandwidth Memory (HBM) and/or may be non-volatile memories such as 3D XPoint or Nano-Ram. In at least one embodiment, some portion of processor memories **1201** may be volatile memory and another portion may be non-volatile memory (e.g., using a two-level memory (2LM) hierarchy).

As described herein, although various multi-core processors **1205** and GPUs **1210** may be physically coupled to a particular memory **1201**, **1220**, respectively, and/or a unified memory architecture may be implemented in which a virtual system address space (also referred to as "effective address" space) is distributed among various physical memories. For example, processor memories **1201**(**1**)-**1201**(M) may each comprise 64 GB of system memory address space and GPU memories **1220**(**1**)-**1220**(N) may each comprise 32 GB of system memory address space resulting in a total of 256 GB addressable memory when M=2 and N=4. Other values for N and M are possible.

FIG. **12**B illustrates additional details for an interconnection between a multi-core processor **1207** and a graphics acceleration module **1246** in accordance with one exemplary embodiment. In at least one embodiment, graphics acceleration module **1246** may include one or more GPU chips integrated on a line card which is coupled to processor **1207** via high-speed link **1240** (e.g., a PCIe bus, NVLink, etc.). In at least one embodiment, graphics acceleration module **1246** may alternatively be integrated on a package or chip with processor **1207**.

In at least one embodiment, processor **1207** includes a plurality of cores **1260**A-**1260**D (which may be referred to as "execution units"), each with a translation lookaside buffer ("TLB") **1261**A-**1261**D and one or more caches **1262**A-**1262**D. In at least one embodiment, cores **1260**A-**1260**D may include various other components for executing instructions and processing data that are not illustrated. In at least one embodiment, caches **1262**A-**1262**D may comprise Level 1 (L1) and Level 2 (L2) caches. In addition, one or more shared caches **1256** may be included in caches **1262**A-**1262**D and shared by sets of cores **1260**A-**1260**D. For example, one embodiment of processor **1207** includes 24 cores, each with its own L1 cache, twelve shared L2 caches, and twelve shared L3 caches. In this embodiment, one or more L2 and L3 caches are shared by two adjacent cores. In at least one embodiment, processor **1207** and graphics acceleration module **1246** connect with system memory **1214**, which may include processor memories **1201**(**1**)-**1201**(M) of FIG. **12**A.

In at least one embodiment, coherency is maintained for data and instructions stored in various caches **1262**A-**1262**D, **1256** and system memory **1214** via inter-core communication over a coherence bus **1264**. In at least one embodiment, for example, each cache may have cache coherency logic/circuitry associated therewith to communicate to over coherence bus **1264** in response to detected reads or writes to particular cache lines. In at least one embodiment, a cache snooping protocol is implemented over coherence bus **1264** to snoop cache accesses.

In at least one embodiment, a proxy circuit **1225** communicatively couples graphics acceleration module **1246** to coherence bus **1264**, allowing graphics acceleration module **1246** to participate in a cache coherence protocol as a peer of cores **1260**A-**1260**D. In particular, in at least one embodiment, an interface **1235** provides connectivity to proxy circuit **1225** over high-speed link **1240** and an interface **1237** connects graphics acceleration module **1246** to high-speed link **1240**.

In at least one embodiment, an accelerator integration circuit **1236** provides cache management, memory access, context management, and interrupt management services on behalf of a plurality of graphics processing engines **1231**(**1**)-**1231**(N) of graphics acceleration module **1246**. In at least one embodiment, graphics processing engines **1231**(**1**)-**1231**(N) may each comprise a separate graphics processing unit (GPU). In at least one embodiment, plurality of graphics processing engines **1231**(**1**)-**1231**(N) of graphics acceleration module **1246** include one or more graphics cores **1500** as discussed in connection with FIGS. **15**A and **15**B. In at least one embodiment, graphics processing engines **1231**(**1**)-**1231**(N) alternatively may comprise different types of graphics processing engines within a GPU, such

as graphics execution units, media processing engines (e.g., video encoders/decoders), samplers, and blit engines. In at least one embodiment, graphics acceleration module **1246** may be a GPU with a plurality of graphics processing engines **1231**(**1**)-**1231**(N) or graphics processing engines **1231**(**1**)-**1231**(N) may be individual GPUs integrated on a common package, line card, or chip.

In at least one embodiment, accelerator integration circuit **1236** includes a memory management unit (MMU) **1239** for performing various memory management functions such as virtual-to-physical memory translations (also referred to as effective-to-real memory translations) and memory access protocols for accessing system memory **1214**. In at least one embodiment, MMU **1239** may also include a translation lookaside buffer (TLB) (not shown) for caching virtual/effective to physical/real address translations. In at least one embodiment, a cache **1238** can store commands and data for efficient access by graphics processing engines **1231**(**1**)-**1231**(N). In at least one embodiment, data stored in cache **1238** and graphics memories **1233**(**1**)-**1233**(M) is kept coherent with core caches **1262**A-**1262**D, **1256** and system memory **1214**, possibly using a fetch unit **1244**. As mentioned, this may be accomplished via proxy circuit **1225** on behalf of cache **1238** and memories **1233**(**1**)-**1233**(M) (e.g., sending updates to cache **1238** related to modifications/accesses of cache lines on processor caches **1262**A-**1262**D, **1256** and receiving updates from cache **1238**).

In at least one embodiment, a set of registers **1245** store context data for threads executed by graphics processing engines **1231**(**1**)-**1231**(N) and a context management circuit **1248** manages thread contexts. For example, context management circuit **1248** may perform save and restore operations to save and restore contexts of various threads during contexts switches (e.g., where a first thread is saved and a second thread is stored so that a second thread can be execute by a graphics processing engine). For example, on a context switch, context management circuit **1248** may store current register values to a designated region in memory (e.g., identified by a context pointer). It may then restore register values when returning to a context. In at least one embodiment, an interrupt management circuit **1247** receives and processes interrupts received from system devices.

In at least one embodiment, virtual/effective addresses from a graphics processing engine **1231** are translated to real/physical addresses in system memory **1214** by MMU **1239**. In at least one embodiment, accelerator integration circuit **1236** supports multiple (e.g., 4, 8, 16) graphics accelerator modules **1246** and/or other accelerator devices. In at least one embodiment, graphics accelerator module **1246** may be dedicated to a single application executed on processor **1207** or may be shared between multiple applications. In at least one embodiment, a virtualized graphics execution environment is presented in which resources of graphics processing engines **1231**(**1**)-**1231**(N) are shared with multiple applications or virtual machines (VMs). In at least one embodiment, resources may be subdivided into "slices" which are allocated to different VMs and/or applications based on processing requirements and priorities associated with VMs and/or applications.

In at least one embodiment, accelerator integration circuit **1236** performs as a bridge to a system for graphics acceleration module **1246** and provides address translation and system memory cache services. In addition, in at least one embodiment, accelerator integration circuit **1236** may provide virtualization facilities for a host processor to manage virtualization of graphics processing engines **1231**(**1**)-**1231**(N), interrupts, and memory management.

In at least one embodiment, because hardware resources of graphics processing engines **1231**(**1**)-**1231**(N) are mapped explicitly to a real address space seen by host processor **1207**, any host processor can address these resources directly using an effective address value. In at least one embodiment, one function of accelerator integration circuit **1236** is physical separation of graphics processing engines **1231**(**1**)-**1231**(N) so that they appear to a system as independent units.

In at least one embodiment, one or more graphics memories **1233**(**1**)-**1233**(M) are coupled to each of graphics processing engines **1231**(**1**)-**1231**(N), respectively and N=M. In at least one embodiment, graphics memories **1233**(**1**)-**1233**(M) store instructions and data being processed by each of graphics processing engines **1231**(**1**)-**1231**(N). In at least one embodiment, graphics memories **1233**(**1**)-**1233**(M) may be volatile memories such as DRAMs (including stacked DRAMs), GDDR memory (e.g., GDDR5, GDDR6), or HBM, and/or may be non-volatile memories such as 3D XPoint or Nano-Ram.

In at least one embodiment, to reduce data traffic over high-speed link **1240**, biasing techniques can be used to ensure that data stored in graphics memories **1233**(**1**)-**1233**(M) is data that will be used most frequently by graphics processing engines **1231**(**1**)-**1231**(N) and preferably not used by cores **1260**A-**1260**D (at least not frequently). Similarly, in at least one embodiment, a biasing mechanism attempts to keep data needed by cores (and preferably not graphics processing engines **1231**(**1**)-**1231**(N)) within caches **1262**A-**1262**D, **1256** and system memory **1214**.

FIG. **12**C illustrates another exemplary embodiment in which accelerator integration circuit **1236** is integrated within processor **1207**. In this embodiment, graphics processing engines **1231**(**1**)-**1231**(N) communicate directly over high-speed link **1240** to accelerator integration circuit **1236** via interface **1237** and interface **1235** (which, again, may be any form of bus or interface protocol). In at least one embodiment, accelerator integration circuit **1236** may perform similar operations as those described with respect to FIG. **12**B, but potentially at a higher throughput given its close proximity to coherence bus **1264** and caches **1262**A-**1262**D, **1256**. In at least one embodiment, an accelerator integration circuit supports different programming models including a dedicated-process programming model (no graphics acceleration module virtualization) and shared programming models (with virtualization), which may include programming models which are controlled by accelerator integration circuit **1236** and programming models which are controlled by graphics acceleration module **1246**.

In at least one embodiment, graphics processing engines **1231**(**1**)-**1231**(N) are dedicated to a single application or process under a single operating system. In at least one embodiment, a single application can funnel other application requests to graphics processing engines **1231**(**1**)-**1231**(N), providing virtualization within a VM/partition.

In at least one embodiment, graphics processing engines **1231**(**1**)-**1231**(N), may be shared by multiple VM/application partitions. In at least one embodiment, shared models may use a system hypervisor to virtualize graphics processing engines **1231**(**1**)-**1231**(N) to allow access by each operating system. In at least one embodiment, for single-partition systems without a hypervisor, graphics processing engines **1231**(**1**)-**1231**(N) are owned by an operating system. In at least one embodiment, an operating system can virtualize

graphics processing engines **1231**(**1**)-**1231**(N) to provide access to each process or application.

In at least one embodiment, graphics acceleration module **1246** or an individual graphics processing engine **1231**(**1**)-**1231**(N) selects a process element using a process handle. In at least one embodiment, process elements are stored in system memory **1214** and are addressable using an effective address to real address translation technique described herein. In at least one embodiment, a process handle may be an implementation-specific value provided to a host process when registering its context with graphics processing engine **1231**(**1**)-**1231**(N) (that is, calling system software to add a process element to a process element linked list). In at least one embodiment, a lower 16-bits of a process handle may be an offset of a process element within a process element linked list.

FIG. **12**D illustrates an exemplary accelerator integration slice **1290**. In at least one embodiment, a "slice" comprises a specified portion of processing resources of accelerator integration circuit **1236**. In at least one embodiment, an application is effective address space **1282** within system memory **1214** stores process elements **1283**. In at least one embodiment, process elements **1283** are stored in response to GPU invocations **1281** from applications **1280** executed on processor **1207**. In at least one embodiment, a process element **1283** contains process state for corresponding application **1280**. In at least one embodiment, a work descriptor (WD) **1284** contained in process element **1283** can be a single job requested by an application or may contain a pointer to a queue of jobs. In at least one embodiment, WD **1284** is a pointer to a job request queue in an application's effective address space **1282**.

In at least one embodiment, graphics acceleration module **1246** and/or individual graphics processing engines **1231**(**1**)-**1231**(N) can be shared by all or a subset of processes in a system. In at least one embodiment, an infrastructure for setting up process states and sending a WD **1284** to a graphics acceleration module **1246** to start a job in a virtualized environment may be included.

In at least one embodiment, a dedicated-process programming model is implementation-specific. In at least one embodiment, in this model, a single process owns graphics acceleration module **1246** or an individual graphics processing engine **1231**. In at least one embodiment, when graphics acceleration module **1246** is owned by a single process, a hypervisor initializes accelerator integration circuit **1236** for an owning partition and an operating system initializes accelerator integration circuit **1236** for an owning process when graphics acceleration module **1246** is assigned.

In at least one embodiment, in operation, a WD fetch unit **1291** in accelerator integration slice **1290** fetches next WD **1284**, which includes an indication of work to be done by one or more graphics processing engines of graphics acceleration module **1246**. In at least one embodiment, data from WD **1284** may be stored in registers **1245** and used by MMU **1239**, interrupt management circuit **1247** and/or context management circuit **1248** as illustrated. For example, one embodiment of MMU **1239** includes segment/page walk circuitry for accessing segment/page tables **1286** within an OS virtual address space **1285**. In at least one embodiment, interrupt management circuit **1247** may process interrupt events **1292** received from graphics acceleration module **1246**. In at least one embodiment, when performing graphics operations, an effective address **1293** generated by a graphics processing engine **1231**(**1**)-**1231**(N) is translated to a real address by MMU **1239**.

In at least one embodiment, registers **1245** are duplicated for each graphics processing engine **1231**(**1**)-**1231**(N) and/or graphics acceleration module **1246** and may be initialized by a hypervisor or an operating system. In at least one embodiment, each of these duplicated registers may be included in an accelerator integration slice **1290**. Exemplary registers that may be initialized by a hypervisor are shown in Table 1.

TABLE 1

Hypervisor Initialized Registers

Register #	Description
1	Slice Control Register
2	Real Address (RA) Scheduled Processes Area Pointer
3	Authority Mask Override Register
4	Interrupt Vector Table Entry Offset
5	Interrupt Vector Table Entry Limit
6	State Register
7	Logical Partition ID
8	Real address (RA) Hypervisor Accelerator Utilization Record Pointer
9	Storage Description Register

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

TABLE 2

Operating System Initialized Registers

Register #	Description
1	Process and Thread Identification
2	Effective Address (EA) Context Save/Restore Pointer
3	Virtual Address (VA) Accelerator Utilization Record Pointer
4	Virtual Address (VA) Storage Segment Table Pointer
5	Authority Mask
6	Work descriptor

In at least one embodiment, each WD **1284** is specific to a particular graphics acceleration module **1246** and/or graphics processing engines **1231**(**1**)-**1231**(N). In at least one embodiment, it contains all information required by a graphics processing engine **1231**(**1**)-**1231**(N) to do work, or it can be a pointer to a memory location where an application has set up a command queue of work to be completed.

FIG. **12**E illustrates additional details for one exemplary embodiment of a shared model. This embodiment includes a hypervisor real address space **1298** in which a process element list **1299** is stored. In at least one embodiment, hypervisor real address space **1298** is accessible via a hypervisor **1296** which virtualizes graphics acceleration module engines for operating system **1295**.

In at least one embodiment, shared programming models allow for all or a subset of processes from all or a subset of partitions in a system to use a graphics acceleration module **1246**. In at least one embodiment, there are two programming models where graphics acceleration module **1246** is shared by multiple processes and partitions, namely time-sliced shared and graphics directed shared.

In at least one embodiment, in this model, system hypervisor **1296** owns graphics acceleration module **1246** and makes its function available to all operating systems **1295**. In at least one embodiment, for a graphics acceleration module **1246** to support virtualization by system hypervisor

1296, graphics acceleration module **1246** may adhere to certain requirements, such as (1) an application's job request must be autonomous (that is, state does not need to be maintained between jobs), or graphics acceleration module **1246** must provide a context save and restore mechanism, (2) an application's job request is guaranteed by graphics acceleration module **1246** to complete in a specified amount of time, including any translation faults, or graphics acceleration module **1246** provides an ability to preempt processing of a job, and (3) graphics acceleration module **1246** must be guaranteed fairness between processes when operating in a directed shared programming model.

In at least one embodiment, application **1280** is required to make an operating system **1295** system call with a graphics acceleration module type, a work descriptor (WD), an authority mask register (AMR) value, and a context save/restore area pointer (CSRP). In at least one embodiment, graphics acceleration module type describes a targeted acceleration function for a system call. In at least one embodiment, graphics acceleration module type may be a system-specific value. In at least one embodiment, WD is formatted specifically for graphics acceleration module **1246** and can be in a form of a graphics acceleration module **1246** command, an effective address pointer to a user-defined structure, an effective address pointer to a queue of commands, or any other data structure to describe work to be done by graphics acceleration module **1246**.

In at least one embodiment, an AMR value is an AMR state to use for a current process. In at least one embodiment, a value passed to an operating system is similar to an application setting an AMR. In at least one embodiment, if accelerator integration circuit **1236** (not shown) and graphics acceleration module **1246** implementations do not support a User Authority Mask Override Register (UAMOR), an operating system may apply a current UAMOR value to an AMR value before passing an AMR in a hypervisor call. In at least one embodiment, hypervisor **1296** may optionally apply a current Authority Mask Override Register (AMOR) value before placing an AMR into process element **1283**. In at least one embodiment, CSRP is one of registers **1245** containing an effective address of an area in an application's effective address space **1282** for graphics acceleration module **1246** to save and restore context state. In at least one embodiment, this pointer is optional if no state is required to be saved between jobs or when a job is preempted. In at least one embodiment, context save/restore area may be pinned system memory.

Upon receiving a system call, operating system **1295** may verify that application **1280** has registered and been given authority to use graphics acceleration module **1246**. In at least one embodiment, operating system **1295** then calls hypervisor **1296** with information shown in Table 3.

TABLE 3

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

TABLE 3-continued

OS to Hypervisor Call Parameters	
Parameter #	Description

In at least one embodiment, upon receiving a hypervisor call, hypervisor **1296** verifies that operating system **1295** has registered and been given authority to use graphics acceleration module **1246**. In at least one embodiment, hypervisor **1296** then puts process element **1283** into a process element linked list for a corresponding graphics acceleration module **1246** type. In at least one embodiment, a process element may include information shown in Table 4.

TABLE 4

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

In at least one embodiment, hypervisor initializes a plurality of accelerator integration slice **1290** registers **1245**.

As illustrated in FIG. **12**F, in at least one embodiment, a unified memory is used, addressable via a common virtual memory address space used to access physical processor memories **1201**(**1**)-**1201**(N) and GPU memories **1220**(**1**)-**1220**(N). In this implementation, operations executed on GPUs **1210**(**1**)-**1210**(N) utilize a same virtual/effective memory address space to access processor memories **1201**(**1**)-**1201**(M) and vice versa, thereby simplifying programmability. In at least one embodiment, a first portion of a virtual/effective address space is allocated to processor memory **1201**(**1**), a second portion to second processor memory **1201**(N), a third portion to GPU memory **1220**(**1**), and so on. In at least one embodiment, an entire virtual/effective memory space (sometimes referred to as an effective address space) is thereby distributed across each of processor memories **1201** and GPU memories **1220**, allowing any processor or GPU to access any physical memory with a virtual address mapped to that memory.

In at least one embodiment, bias/coherence management circuitry **1294**A-**1294**E within one or more of MMUs **1239**A-**1239**E ensures cache coherence between caches of one or more host processors (e.g., **1205**) and GPUs **1210** and implements biasing techniques indicating physical memories in which certain types of data should be stored. In at least one embodiment, while multiple instances of bias/coherence management circuitry **1294**A-**1294**E are illustrated in FIG. **12**F, bias/coherence circuitry may be imple-

mented within an MMU of one or more host processors **1205** and/or within accelerator integration circuit **1236**.

One embodiment allows GPU memories **1220** to be mapped as part of system memory, and accessed using shared virtual memory (SVM) technology, but without suffering performance drawbacks associated with full system cache coherence. In at least one embodiment, an ability for GPU memories **1220** to be accessed as system memory without onerous cache coherence overhead provides a beneficial operating environment for GPU offload. In at least one embodiment, this arrangement allows software of host processor **1205** to setup operands and access computation results, without overhead of tradition I/O DMA data copies. In at least one embodiment, such traditional copies involve driver calls, interrupts and memory mapped I/O (MMIO) accesses that are all inefficient relative to simple memory accesses. In at least one embodiment, an ability to access GPU memories **1220** without cache coherence overheads can be critical to execution time of an offloaded computation. In at least one embodiment, in cases with substantial streaming write memory traffic, for example, cache coherence overhead can significantly reduce an effective write bandwidth seen by a GPU **1210**. In at least one embodiment, efficiency of operand setup, efficiency of results access, and efficiency of GPU computation may play a role in determining effectiveness of a GPU offload.

In at least one embodiment, selection of GPU bias and host processor bias is driven by a bias tracker data structure. In at least one embodiment, a bias table may be used, for example, which may be a page-granular structure (e.g., controlled at a granularity of a memory page) that includes 1 or 2 bits per GPU-attached memory page. In at least one embodiment, a bias table may be implemented in a stolen memory range of one or more GPU memories **1220**, with or without a bias cache in a GPU **1210** (e.g., to cache frequently/recently used entries of a bias table). Alternatively, in at least one embodiment, an entire bias table may be maintained within a GPU.

In at least one embodiment, a bias table entry associated with each access to a GPU attached memory **1220** is accessed prior to actual access to a GPU memory, causing following operations. In at least one embodiment, local requests from a GPU **1210** that find their page in GPU bias are forwarded directly to a corresponding GPU memory **1220**. In at least one embodiment, local requests from a GPU that find their page in host bias are forwarded to processor **1205** (e.g., over a high-speed link as described herein). In at least one embodiment, requests from processor **1205** that find a requested page in host processor bias complete a request like a normal memory read. Alternatively, requests directed to a GPU-biased page may be forwarded to a GPU **1210**. In at least one embodiment, a GPU may then transition a page to a host processor bias if it is not currently using a page. In at least one embodiment, a bias state of a page can be changed either by a software-based mechanism, a hardware-assisted software-based mechanism, or, for a limited set of cases, a purely hardware-based mechanism.

In at least one embodiment, one mechanism for changing bias state employs an API call (e.g., OpenCL), which, in turn, calls a GPU's device driver which, in turn, sends a message (or enqueues a command descriptor) to a GPU directing it to change a bias state and, for some transitions, perform a cache flushing operation in a host. In at least one embodiment, a cache flushing operation is used for a transition from host processor **1205** bias to GPU bias, but is not for an opposite transition.

In at least one embodiment, cache coherency is maintained by temporarily rendering GPU-biased pages uncacheable by host processor **1205**. In at least one embodiment, to access these pages, processor **1205** may request access from GPU **1210**, which may or may not grant access right away. In at least one embodiment, thus, to reduce communication between processor **1205** and GPU **1210** it is beneficial to ensure that GPU-biased pages are those which are required by a GPU but not host processor **1205** and vice versa.

FIG. **13** illustrates exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included in at least one embodiment, including additional graphics processors/cores, peripheral interface controllers, or general-purpose processor cores.

FIG. **13** is a block diagram illustrating an exemplary system on a chip integrated circuit **1300** that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, integrated circuit **1300** includes one or more application processor(s) **1305** (e.g., CPUs), at least one graphics processor **1310**, and may additionally include an image processor **1315** and/or a video processor **1320**, any of which may be a modular IP core. In at least one embodiment, integrated circuit **1300** includes peripheral or bus logic including a USB controller **1325**, a UART controller **1330**, an SPI/SDIO controller **1335**, and an I^2S/I^2C controller **1340**. In at least one embodiment, integrated circuit **1300** can include a display device **1345** coupled to one or more of a high-definition multimedia interface (HDMI) controller **1350** and a mobile industry processor interface (MIPI) display interface **1355**. In at least one embodiment, storage may be provided by a flash memory subsystem **1360** including flash memory and a flash memory controller. In at least one embodiment, a memory interface may be provided via a memory controller **1365** for access to SDRAM or SRAM memory devices. In at least one embodiment, some integrated circuits additionally include an embedded security engine **1370**.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIGS. **14**A-**14**B illustrate exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included in at least one embodiment, including additional graphics processors/cores, peripheral interface controllers, or general-purpose processor cores.

FIGS. **14**A-**14**B are block diagrams illustrating exemplary graphics processors for use within an SoC, according to embodiments described herein. FIG. **14**A illustrates an exemplary graphics processor **1410** of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. FIG. **14**B illustrates an additional exemplary graphics processor **1440** of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to at least one embodiment. In at least one embodiment, graphics processor **1410** of FIG. **14**A is a low power graphics processor core. In at least one embodiment, graphics processor **1440** of FIG. **14**B is a higher performance graphics processor core. In at

least one embodiment, each of graphics processors **1410**, **1440** can be variants of graphics processor **1310** of FIG. **13**.

In at least one embodiment, graphics processor **1410** includes a vertex processor **1405** and one or more fragment processor(s) **1415**A-**1415**N (e.g., **1415**A, **1415**B, **1415**C, **1415**D, through **1415**N-**1**, and **1415**N). In at least one embodiment, graphics processor **1410** can execute different shader programs via separate logic, such that vertex processor **1405** is optimized to execute operations for vertex shader programs, while one or more fragment processor(s) **1415**A-**1415**N execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. In at least one embodiment, vertex processor **1405** performs a vertex processing stage of a 3D graphics pipeline and generates primitives and vertex data. In at least one embodiment, fragment processor(s) **1415**A-**1415**N use primitive and vertex data generated by vertex processor **1405** to produce a framebuffer that is displayed on a display device. In at least one embodiment, fragment processor(s) **1415**A-**1415**N are optimized to execute fragment shader programs as provided for in an OpenGL API, which may be used to perform similar operations as a pixel shader program as provided for in a Direct 3D API.

In at least one embodiment, graphics processor **1410** additionally includes one or more memory management units (MMUs) **1420**A-**1420**B, cache(s) **1425**A-**1425**B, and circuit interconnect(s) **1430**A-**1430**B. In at least one embodiment, one or more MMU(s) **1420**A-**1420**B provide for virtual to physical address mapping for graphics processor **1410**, including for vertex processor **1405** and/or fragment processor(s) **1415**A-**1415**N, which may reference vertex or image/texture data stored in memory, in addition to vertex or image/texture data stored in one or more cache(s) **1425**A-**1425**B. In at least one embodiment, one or more MMU(s) **1420**A-**1420**B may be synchronized with other MMUs within a system, including one or more MMUs associated with one or more application processor(s) **1305**, image processors **1315**, and/or video processors **1320** of FIG. **13**, such that each processor **1305**-**1320** can participate in a shared or unified virtual memory system. In at least one embodiment, one or more circuit interconnect(s) **1430**A-**1430**B enable graphics processor **1410** to interface with other IP cores within SoC, either via an internal bus of SoC or via a direct connection.

In at least one embodiment, graphics processor **1440** includes one or more shader core(s) **1455**A-**1455**N (e.g., **1455**A, **1455**B, **1455**C, **1455**D, **1455**E, **1455**F, through **1455**N-**1**, and **1455**N) as shown in FIG. **14**B, which provides for a unified shader core architecture in which a single core or type or core can execute all types of programmable shader code, including shader program code to implement vertex shaders, fragment shaders, and/or compute shaders. In at least one embodiment, a number of shader cores can vary. In at least one embodiment, graphics processor **1440** includes an inter-core task manager **1445**, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores **1455**A-**1455**N and a tiling unit **1458** to accelerate tiling operations for tile-based rendering, in which rendering operations for a scene are subdivided in image space, for example to exploit local spatial coherence within a scene or to optimize use of internal caches.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIGS. **15**A-**15**B illustrate additional exemplary graphics processor logic according to embodiments described herein. FIG. **15**A illustrates a graphics core **1500** that may be included within graphics processor **1310** of FIG. **13**, in at least one embodiment, and may be a unified shader core **1455**A-**1455**N as in FIG. **14**B in at least one embodiment. FIG. **15**B illustrates a highly-parallel general-purpose graphics processing unit ("GPGPU") **1530** suitable for deployment on a multi-chip module in at least one embodiment.

In at least one embodiment, graphics core **1500** includes a shared instruction cache **1502**, a texture unit **1518**, and a cache/shared memory **1520** (e.g., including L1, L2, L3, last level cache, or other caches) that are common to execution resources within graphics core **1500**. In at least one embodiment, graphics core **1500** can include multiple slices **1501**A-**1501**N or a partition for each core, and a graphics processor can include multiple instances of graphics core **1500**. In at least one embodiment, each slice **1501**A-**1501**N refers to graphics core **1500**. In at least one embodiment, slices **1501**A-**1501**N have sub-slices, which are part of a slice **1501**A-**1501**N. In at least one embodiment, slices **1501**A-**1501**N are independent of other slices or dependent on other slices. In at least one embodiment, slices **1501**A-**1501**N can include support logic including a local instruction cache **1504**A-**1504**N, a thread scheduler (sequencer) **1506**A-**1506**N, a thread dispatcher **1508**A-**1508**N, and a set of registers **1510**A-**1510**N. In at least one embodiment, slices **1501**A-**1501**N can include a set of additional function units (AFUs **1512**A-**1512**N), floating-point units (FPUs **1514**A-**1514**N), integer arithmetic logic units (ALUs **1516**A-**1516**N), address computational units (ACUs **1513**A-**1513**N), double-precision floating-point units (DPFPUs **1515**A-**1515**N), and matrix processing units (MPUs **1517**A-**1517**N).

In at least one embodiment, each slice **1501**A-**1501**N includes one or more engines for floating point and integer vector operations and one or more engines to accelerate convolution and matrix operations in AI, machine learning, or large dataset workloads. In at least one embodiment, one or more slices **1501**A-**1501**N include one or more vector engines to compute a vector (e.g., compute mathematical operations for vectors). In at least one embodiment, a vector engine can compute a vector operation in 16-bit floating point (also referred to as "FP16"), 32-bit floating point (also referred to as "FP32"), or 64-bit floating point (also referred to as "FP64"). In at least one embodiment, one or more slices **1501**A-**1501**N includes 16 vector engines that are paired with 16 matrix math units to compute matrix/tensor operations, where vector engines and math units are exposed via matrix extensions. In at least one embodiment, a slice a specified portion of processing resources of a processing unit, e.g., 16 cores and a ray tracing unit or 8 cores, a thread scheduler, a thread dispatcher, and additional functional units for a processor. In at least one embodiment, graphics core **1500** includes one or more matrix engines to compute matrix operations, e.g., when computing tensor operations.

In at least one embodiment, one or more slices **1501**A-**1501**N includes one or more ray tracing units to compute ray tracing operations (e.g., 16 ray tracing units per slice slices **1501**A-**1501**N). In at least one embodiment, a ray tracing unit computes ray traversal, triangle intersection, bounding box intersect, or other ray tracing operations.

In at least one embodiment, one or more slices **1501**A-**1501**N includes a media slice that encodes, decodes, and/or transcodes data; scales and/or format converts data; and/or performs video quality operations on video data.

In at least one embodiment, one or more slices **1501**A-**1501**N are linked to L2 cache and memory fabric, link connectors, high-bandwidth memory (HBM) (e.g., HBM2e,

HDM3) stacks, and a media engine. In at least one embodiment, one or more slices **1501**A-**1501**N include multiple cores (e.g., 16 cores) and multiple ray tracing units (e.g., 16) paired to each core. In at least one embodiment, one or more slices **1501**A-**1501**N has one or more L1 caches. In at least one embodiment, one or more slices **1501**A-**1501**N include one or more vector engines; one or more instruction caches to store instructions; one or more L1 caches to cache data; one or more shared local memories (SLMs) to store data, e.g., corresponding to instructions; one or more samplers to sample data; one or more ray tracing units to perform ray tracing operations; one or more geometries to perform operations in geometry pipelines and/or apply geometric transformations to vertices or polygons; one or more rasterizers to describe an image in vector graphics format (e.g., shape) and convert it into a raster image (e.g., a series of pixels, dots, or lines, which when displayed together, create an image that is represented by shapes); one or more a Hierarchical Depth Buffer (Hiz) to buffer data; and/or one or more pixel backends. In at least one embodiment, a slice **1501**A-**1501**N includes a memory fabric, e.g., an L2 cache.

In at least one embodiment, FPUs **1514**A-**1514**N can perform single-precision (32-bit) and half-precision (16-bit) floating point operations, while DPFPUs **1515**A-**1515**N perform double precision (64-bit) floating point operations. In at least one embodiment, ALUs **1516**A-**1516**N can perform variable precision integer operations at 8-bit, 16-bit, and 32-bit precision, and can be configured for mixed precision operations. In at least one embodiment, MPUs **1517**A-**1517**N can also be configured for mixed precision matrix operations, including half-precision floating point and 8-bit integer operations. In at least one embodiment, MPUs **1517**-**1517**N can perform a variety of matrix operations to accelerate machine learning application frameworks, including enabling support for accelerated general matrix to matrix multiplication (GEMM). In at least one embodiment, AFUs **1512**A-**1512**N can perform additional logic operations not supported by floating-point or integer units, including trigonometric operations (e.g., sine, cosine). Inference and/or training logic **515** are used to perform inferencing and/or training operations associated with one or more embodiments.

In at least one embodiment, graphics core **1500** includes an interconnect and a link fabric sublayer that is attached to a switch and a GPU-GPU bridge that enables multiple graphics processors **1500** (e.g., 8) to be interlinked without glue to each other with load/store units (LSUs), data transfer units, and sync semantics across multiple graphics processors **1500**. In at least one embodiment, interconnects include standardized interconnects (e.g., PCIe) or some combination thereof.

In at least one embodiment, graphics core **1500** includes multiple tiles. In at least one embodiment, a tile is an individual die or one or more dies, where individual dies can be connected with an interconnect (e.g., embedded multi-die interconnect bridge (EMIB)). In at least one embodiment, graphics core **1500** includes a compute tile, a memory tile (e.g., where a memory tile can be exclusively accessed by different tiles or different chipsets such as a Rambo tile), substrate tile, a base tile, a HMB tile, a link tile, and EMIB tile, where all tiles are packaged together in graphics core **1500** as part of a GPU. In at least one embodiment, graphics core **1500** can include multiple tiles in a single package (also referred to as a "multi tile package"). In at least one embodiment, a compute tile can have 8 graphics cores **1500**, an L1 cache; and a base tile can have a host interface with PCIe 5.0, HBM2e, MDFI, and EMIB, a link tile with 8 links, 8 ports with an embedded switch. In at least one embodiment, tiles are connected with face-to-face (F2F) chip-on-chip bonding through fine-pitched, 36-micron, microbumps (e.g., copper pillars). In at least one embodiment, graphics core **1500** includes memory fabric, which includes memory, and is tile that is accessible by multiple tiles. In at least one embodiment, graphics core **1500** stores, accesses, or loads its own hardware contexts in memory, where a hardware context is a set of data loaded from registers before a process resumes, and where a hardware context can indicate a state of hardware (e.g., state of a GPU).

In at least one embodiment, graphics core **1500** includes serializer/deserializer (SERDES) circuitry that converts a serial data stream to a parallel data stream, or converts a parallel data stream to a serial data stream.

In at least one embodiment, graphics core **1500** includes a high speed coherent unified fabric (GPU to GPU), load/store units, bulk data transfer and sync semantics, and connected GPUs through an embedded switch, where a GPU-GPU bridge is controlled by a controller.

In at least one embodiment, graphics core **1500** performs an API, where said API abstracts hardware of graphics core **1500** and access libraries with instructions to perform math operations (e.g., math kernel library), deep neural network operations (e.g., deep neural network library), vector operations, collective communications, thread building blocks, video processing, data analytics library, and/or ray tracing operations.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **15**B illustrates a general-purpose processing unit (GPGPU) **1530** that can be configured to enable highly-parallel compute operations to be performed by an array of graphics processing units, in at least one embodiment. In at least one embodiment, GPGPU **1530** can be linked directly to other instances of GPGPU **1530** to create a multi-GPU cluster to improve training speed for deep neural networks. In at least one embodiment, GPGPU **1530** includes a host interface **1532** to enable a connection with a host processor. In at least one embodiment, host interface **1532** is a PCI Express interface. In at least one embodiment, host interface **1532** can be a vendor-specific communications interface or communications fabric. In at least one embodiment, GPGPU **1530** receives commands from a host processor and uses a global scheduler **1534** (which may be referred to as a thread sequencer and/or asynchronous compute engine) to distribute execution threads associated with those commands to a set of compute clusters **1536**A-**1536**H. In at least one embodiment, compute clusters **1536**A-**1536**H share a cache memory **1538**. In at least one embodiment, cache memory **1538** can serve as a higher-level cache for cache memories within compute clusters **1536**A-**1536**H.

In at least one embodiment, GPGPU **1530** includes memory **1544**A-**1544**B coupled with compute clusters **1536**A-**1536**H via a set of memory controllers **1542**A-**1542**B (e.g., one or more controllers for HBM2e). In at least one embodiment, memory **1544**A-**1544**B can include various types of memory devices including dynamic random access memory (DRAM) or graphics random access memory, such as synchronous graphics random access memory (SGRAM), including graphics double data rate (GDDR) memory.

In at least one embodiment, compute clusters **1536**A-**1536**H each include a set of graphics cores, such as graphics core **1500** of FIG. **15**A, which can include multiple types of

integer and floating point logic units that can perform computational operations at a range of precisions including suited for machine learning computations. For example, in at least one embodiment, at least a subset of floating point units in each of compute clusters **1536**A-**1536**H can be configured to perform 16-bit or 32-bit floating point operations, while a different subset of floating point units can be configured to perform 64-bit floating point operations.

In at least one embodiment, multiple instances of GPGPU **1530** can be configured to operate as a compute cluster. In at least one embodiment, communication used by compute clusters **1536**A-**1536**H for synchronization and data exchange varies across embodiments. In at least one embodiment, multiple instances of GPGPU **1530** communicate over host interface **1532**. In at least one embodiment, GPGPU **1530** includes an I/O hub **1539** that couples GPGPU **1530** with a GPU link **1540** that enables a direct connection to other instances of GPGPU **1530**. In at least one embodiment, GPU link **1540** is coupled to a dedicated GPU-to-GPU bridge that enables communication and synchronization between multiple instances of GPGPU **1530**. In at least one embodiment, GPU link **1540** couples with a high-speed interconnect to transmit and receive data to other GPGPUs or parallel processors. In at least one embodiment, multiple instances of GPGPU **1530** are located in separate data processing systems and communicate via a network device that is accessible via host interface **1532**. In at least one embodiment GPU link **1540** can be configured to enable a connection to a host processor in addition to or as an alternative to host interface **1532**.

In at least one embodiment, GPGPU **1530** can be configured to train neural networks. In at least one embodiment, GPGPU **1530** can be used within an inferencing platform. In at least one embodiment, in which GPGPU **1530** is used for inferencing, GPGPU **1530** may include fewer compute clusters **1536**A-**1536**H relative to when GPGPU **1530** is used for training a neural network. In at least one embodiment, memory technology associated with memory **1544**A-**1544**B may differ between inferencing and training configurations, with higher bandwidth memory technologies devoted to training configurations. In at least one embodiment, an inferencing configuration of GPGPU **1530** can support inferencing specific instructions. For example, in at least one embodiment, an inferencing configuration can provide support for one or more 8-bit integer dot product instructions, which may be used during inferencing operations for deployed neural networks.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **16** is a block diagram illustrating a computing system **1600** according to at least one embodiment. In at least one embodiment, computing system **1600** includes a processing subsystem **1601** having one or more processor(s) **1602** and a system memory **1604** communicating via an interconnection path that may include a memory hub **1605**. In at least one embodiment, memory hub **1605** may be a separate component within a chipset component or may be integrated within one or more processor(s) **1602**. In at least one embodiment, memory hub **1605** couples with an I/O subsystem **1611** via a communication link **1606**. In at least one embodiment, I/O subsystem **1611** includes an I/O hub **1607** that can enable computing system **1600** to receive input from one or more input device(s) **1608**. In at least one embodiment, I/O hub **1607** can enable a display controller, which may be included in one or more processor(s) **1602**, to provide outputs to one or more display device(s) **1610**A. In at least one embodiment, one or more display device(s) **1610**A coupled with I/O hub **1607** can include a local, internal, or embedded display device.

In at least one embodiment, processing subsystem **1601** includes one or more parallel processor(s) **1612** coupled to memory hub **1605** via a bus or other communication link **1613**. In at least one embodiment, communication link **1613** may use one of any number of standards based communication link technologies or protocols, such as, but not limited to PCI Express, or may be a vendor-specific communications interface or communications fabric. In at least one embodiment, one or more parallel processor(s) **1612** form a computationally focused parallel or vector processing system that can include a large number of processing cores and/or processing clusters, such as a many-integrated core (MIC) processor. In at least one embodiment, some or all of parallel processor(s) **1612** form a graphics processing subsystem that can output pixels to one of one or more display device(s) **1610**A coupled via I/O Hub **1607**. In at least one embodiment, parallel processor(s) **1612** can also include a display controller and display interface (not shown) to enable a direct connection to one or more display device(s) **1610**B. In at least one embodiment, parallel processor(s) **1612** include one or more cores, such as graphics cores **1500** discussed herein.

In at least one embodiment, a system storage unit **1614** can connect to I/O hub **1607** to provide a storage mechanism for computing system **1600**. In at least one embodiment, an I/O switch **1616** can be used to provide an interface mechanism to enable connections between I/O hub **1607** and other components, such as a network adapter **1618** and/or a wireless network adapter **1619** that may be integrated into platform, and various other devices that can be added via one or more add-in device(s) **1620**. In at least one embodiment, network adapter **1618** can be an Ethernet adapter or another wired network adapter. In at least one embodiment, wireless network adapter **1619** can include one or more of a Wi-Fi, Bluetooth, near field communication (NFC), or other network device that includes one or more wireless radios.

In at least one embodiment, computing system **1600** can include other components not explicitly shown, including USB or other port connections, optical storage drives, video capture devices, and like, may also be connected to I/O hub **1607**. In at least one embodiment, communication paths interconnecting various components in FIG. **16** may be implemented using any suitable protocols, such as PCI (Peripheral Component Interconnect) based protocols (e.g., PCI-Express), or other bus or point-to-point communication interfaces and/or protocol(s), such as NV-Link high-speed interconnect, or interconnect protocols.

In at least one embodiment, parallel processor(s) **1612** incorporate circuitry optimized for graphics and video processing, including, for example, video output circuitry, and constitutes a graphics processing unit (GPU), e.g., parallel processor(s) **1612** includes graphics core **1500**. In at least one embodiment, parallel processor(s) **1612** incorporate circuitry optimized for general purpose processing. In at least embodiment, components of computing system **1600** may be integrated with one or more other system elements on a single integrated circuit. For example, in at least one embodiment, parallel processor(s) **1612**, memory hub **1605**, processor(s) **1602**, and I/O hub **1607** can be integrated into a system on chip (SoC) integrated circuit. In at least one embodiment, components of computing system **1600** can be integrated into a single package to form a system in package (SIP) configuration. In at least one embodiment, at least a

portion of components of computing system **1600** can be integrated into a multi-chip module (MCM), which can be interconnected with other multi-chip modules into a modular computing system.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

Processors

FIG. **17**A illustrates a parallel processor **1700** according to at least one embodiment. In at least one embodiment, various components of parallel processor **1700** may be implemented using one or more integrated circuit devices, such as programmable processors, application specific integrated circuits (ASICs), or field programmable gate arrays (FPGA). In at least one embodiment, illustrated parallel processor **1700** is a variant of one or more parallel processor(s) **1612** shown in FIG. **16** according to an exemplary embodiment. In at least one embodiment, a parallel processor **1700** includes one or more graphics cores **1500**.

In at least one embodiment, parallel processor **1700** includes a parallel processing unit **1702**. In at least one embodiment, parallel processing unit **1702** includes an I/O unit **1704** that enables communication with other devices, including other instances of parallel processing unit **1702**. In at least one embodiment, I/O unit **1704** may be directly connected to other devices. In at least one embodiment, I/O unit **1704** connects with other devices via use of a hub or switch interface, such as a memory hub **1705**. In at least one embodiment, connections between memory hub **1705** and I/O unit **1704** form a communication link **1713**. In at least one embodiment, I/O unit **1704** connects with a host interface **1706** and a memory crossbar **1716**, where host interface **1706** receives commands directed to performing processing operations and memory crossbar **1716** receives commands directed to performing memory operations.

In at least one embodiment, when host interface **1706** receives a command buffer via I/O unit **1704**, host interface **1706** can direct work operations to perform those commands to a front end **1708**. In at least one embodiment, front end **1708** couples with a scheduler **1710** (which may be referred to as a sequencer), which is configured to distribute commands or other work items to a processing cluster array **1712**. In at least one embodiment, scheduler **1710** ensures that processing cluster array **1712** is properly configured and in a valid state before tasks are distributed to a cluster of processing cluster array **1712**. In at least one embodiment, scheduler **1710** is implemented via firmware logic executing on a microcontroller. In at least one embodiment, microcontroller implemented scheduler **1710** is configurable to perform complex scheduling and work distribution operations at coarse and fine granularity, enabling rapid preemption and context switching of threads executing on processing array **1712**. In at least one embodiment, host software can prove workloads for scheduling on processing cluster array **1712** via one of multiple graphics processing paths. In at least one embodiment, workloads can then be automatically distributed across processing array cluster **1712** by scheduler **1710** logic within a microcontroller including scheduler **1710**.

In at least one embodiment, processing cluster array **1712** can include up to "N" processing clusters (e.g., cluster **1714**A, cluster **1714**B, through cluster **1714**N), where "N" represents a positive integer (which may be a different integer "N" than used in other figures). In at least one embodiment, each cluster **1714**A-**1714**N of processing cluster array **1712** can execute a large number of concurrent threads. In at least one embodiment, scheduler **1710** can allocate work to clusters **1714**A-**1714**N of processing cluster array **1712** using various scheduling and/or work distribution algorithms, which may vary depending on workload arising for each type of program or computation. In at least one embodiment, scheduling can be handled dynamically by scheduler **1710**, or can be assisted in part by compiler logic during compilation of program logic configured for execution by processing cluster array **1712**. In at least one embodiment, different clusters **1714**A-**1714**N of processing cluster array **1712** can be allocated for processing different types of programs or for performing different types of computations.

In at least one embodiment, processing cluster array **1712** can be configured to perform various types of parallel processing operations. In at least one embodiment, processing cluster array **1712** is configured to perform general-purpose parallel compute operations. For example, in at least one embodiment, processing cluster array **1712** can include logic to execute processing tasks including filtering of video and/or audio data, performing modeling operations, including physics operations, and performing data transformations.

In at least one embodiment, processing cluster array **1712** is configured to perform parallel graphics processing operations. In at least one embodiment, processing cluster array **1712** can include additional logic to support execution of such graphics processing operations, including but not limited to, texture sampling logic to perform texture operations, as well as tessellation logic and other vertex processing logic. In at least one embodiment, processing cluster array **1712** can be configured to execute graphics processing related shader programs such as, but not limited to, vertex shaders, tessellation shaders, geometry shaders, and pixel shaders. In at least one embodiment, parallel processing unit **1702** can transfer data from system memory via I/O unit **1704** for processing. In at least one embodiment, during processing, transferred data can be stored to on-chip memory (e.g., parallel processor memory **1722**) during processing, then written back to system memory.

In at least one embodiment, when parallel processing unit **1702** is used to perform graphics processing, scheduler **1710** can be configured to divide a processing workload into approximately equal sized tasks, to better enable distribution of graphics processing operations to multiple clusters **1714**A-**1714**N of processing cluster array **1712**. In at least one embodiment, portions of processing cluster array **1712** can be configured to perform different types of processing. For example, in at least one embodiment, a first portion may be configured to perform vertex shading and topology generation, a second portion may be configured to perform tessellation and geometry shading, and a third portion may be configured to perform pixel shading or other screen space operations, to produce a rendered image for display. In at least one embodiment, intermediate data produced by one or more of clusters **1714**A-**1714**N may be stored in buffers to allow intermediate data to be transmitted between clusters **1714**A-**1714**N for further processing.

In at least one embodiment, processing cluster array **1712** can receive processing tasks to be executed via scheduler **1710**, which receives commands defining processing tasks from front end **1708**. In at least one embodiment, processing tasks can include indices of data to be processed, e.g., surface (patch) data, primitive data, vertex data, and/or pixel data, as well as state parameters and commands defining how data is to be processed (e.g., what program is to be

executed). In at least one embodiment, scheduler **1710** may be configured to fetch indices corresponding to tasks or may receive indices from front end **1708**. In at least one embodiment, front end **1708** can be configured to ensure processing cluster array **1712** is configured to a valid state before a workload specified by incoming command buffers (e.g., batch-buffers, push buffers, etc.) is initiated.

In at least one embodiment, each of one or more instances of parallel processing unit **1702** can couple with a parallel processor memory **1722**. In at least one embodiment, parallel processor memory **1722** can be accessed via memory crossbar **1716**, which can receive memory requests from processing cluster array **1712** as well as I/O unit **1704**. In at least one embodiment, memory crossbar **1716** can access parallel processor memory **1722** via a memory interface **1718**. In at least one embodiment, memory interface **1718** can include multiple partition units (e.g., partition unit **1720**A, partition unit **1720**B, through partition unit **1720**N) that can each couple to a portion (e.g., memory unit) of parallel processor memory **1722**. In at least one embodiment, a number of partition units **1720**A-**1720**N is configured to be equal to a number of memory units, such that a first partition unit **1720**A has a corresponding first memory unit **1724**A, a second partition unit **1720**B has a corresponding memory unit **1724**B, and an N-th partition unit **1720**N has a corresponding N-th memory unit **1724**N. In at least one embodiment, a number of partition units **1720**A-**1720**N may not be equal to a number of memory units.

In at least one embodiment, memory units **1724**A-**1724**N can include various types of memory devices, including dynamic random access memory (DRAM) or graphics random access memory, such as synchronous graphics random access memory (SGRAM), including graphics double data rate (GDDR) memory. In at least one embodiment, memory units **1724**A-**1724**N may also include 3D stacked memory, including but not limited to high bandwidth memory (HBM), HBM2e, or HDM3. In at least one embodiment, render targets, such as frame buffers or texture maps may be stored across memory units **1724**A-**1724**N, allowing partition units **1720**A-**1720**N to write portions of each render target in parallel to efficiently use available bandwidth of parallel processor memory **1722**. In at least one embodiment, a local instance of parallel processor memory **1722** may be excluded in favor of a unified memory design that utilizes system memory in conjunction with local cache memory.

In at least one embodiment, any one of clusters **1714**A-**1714**N of processing cluster array **1712** can process data that will be written to any of memory units **1724**A-**1724**N within parallel processor memory **1722**. In at least one embodiment, memory crossbar **1716** can be configured to transfer an output of each cluster **1714**A-**1714**N to any partition unit **1720**A-**1720**N or to another cluster **1714**A-**1714**N, which can perform additional processing operations on an output. In at least one embodiment, each cluster **1714**A-**1714**N can communicate with memory interface **1718** through memory crossbar **1716** to read from or write to various external memory devices. In at least one embodiment, memory crossbar **1716** has a connection to memory interface **1718** to communicate with I/O unit **1704**, as well as a connection to a local instance of parallel processor memory **1722**, enabling processing units within different processing clusters **1714**A-**1714**N to communicate with system memory or other memory that is not local to parallel processing unit **1702**. In at least one embodiment, memory crossbar **1716** can use virtual channels to separate traffic streams between clusters **1714**A-**1714**N and partition units **1720**A-**1720**N.

In at least one embodiment, multiple instances of parallel processing unit **1702** can be provided on a single add-in card, or multiple add-in cards can be interconnected. In at least one embodiment, different instances of parallel processing unit **1702** can be configured to interoperate even if different instances have different numbers of processing cores, different amounts of local parallel processor memory, and/or other configuration differences. For example, in at least one embodiment, some instances of parallel processing unit **1702** can include higher precision floating point units relative to other instances. In at least one embodiment, systems incorporating one or more instances of parallel processing unit **1702** or parallel processor **1700** can be implemented in a variety of configurations and form factors, including but not limited to desktop, laptop, or handheld personal computers, servers, workstations, game consoles, and/or embedded systems.

FIG. **17**B is a block diagram of a partition unit **1720** according to at least one embodiment. In at least one embodiment, partition unit **1720** is an instance of one of partition units **1720**A-**1720**N of FIG. **17**A. In at least one embodiment, partition unit **1720** includes an L2 cache **1721**, a frame buffer interface **1725**, and a ROP **1726** (raster operations unit). In at least one embodiment, L2 cache **1721** is a read/write cache that is configured to perform load and store operations received from memory crossbar **1716** and ROP **1726**. In at least one embodiment, read misses and urgent write-back requests are output by L2 cache **1721** to frame buffer interface **1725** for processing. In at least one embodiment, updates can also be sent to a frame buffer via frame buffer interface **1725** for processing. In at least one embodiment, frame buffer interface **1725** interfaces with one of memory units in parallel processor memory, such as memory units **1724**A-**1724**N of FIG. **17** (e.g., within parallel processor memory **1722**).

In at least one embodiment, ROP **1726** is a processing unit that performs raster operations such as stencil, z test, blending, etc. In at least one embodiment, ROP **1726** then outputs processed graphics data that is stored in graphics memory. In at least one embodiment, ROP **1726** includes compression logic to compress depth or color data that is written to memory and decompress depth or color data that is read from memory. In at least one embodiment, compression logic can be lossless compression logic that makes use of one or more of multiple compression algorithms. In at least one embodiment, a type of compression that is performed by ROP **1726** can vary based on statistical characteristics of data to be compressed. For example, in at least one embodiment, delta color compression is performed on depth and color data on a per-tile basis.

In at least one embodiment, ROP **1726** is included within each processing cluster (e.g., cluster **1714**A-**1714**N of FIG. **17**A) instead of within partition unit **1720**. In at least one embodiment, read and write requests for pixel data are transmitted over memory crossbar **1716** instead of pixel fragment data. In at least one embodiment, processed graphics data may be displayed on a display device, such as one of one or more display device(s) **1610** of FIG. **16**, routed for further processing by processor(s) **1602**, or routed for further processing by one of processing entities within parallel processor **1700** of FIG. **17**A.

FIG. **17**C is a block diagram of a processing cluster **1714** within a parallel processing unit according to at least one embodiment. In at least one embodiment, a processing cluster is an instance of one of processing clusters **1714**A-**1714**N of FIG. **17**A. In at least one embodiment, processing cluster **1714** can be configured to execute many threads in

parallel, where "thread" refers to an instance of a particular program executing on a particular set of input data. In at least one embodiment, single-instruction, multiple-data (SIMD) instruction issue techniques are used to support parallel execution of a large number of threads without providing multiple independent instruction units. In at least one embodiment, single-instruction, multiple-thread (SIMT) techniques are used to support parallel execution of a large number of generally synchronized threads, using a common instruction unit configured to issue instructions to a set of processing engines within each one of processing clusters.

In at least one embodiment, operation of processing cluster **1714** can be controlled via a pipeline manager **1732** that distributes processing tasks to SIMT parallel processors. In at least one embodiment, pipeline manager **1732** receives instructions from scheduler **1710** of FIG. **17**A and manages execution of those instructions via a graphics multiprocessor **1734** and/or a texture unit **1736**. In at least one embodiment, graphics multiprocessor **1734** is an exemplary instance of a SIMT parallel processor. However, in at least one embodiment, various types of SIMT parallel processors of differing architectures may be included within processing cluster **1714**. In at least one embodiment, one or more instances of graphics multiprocessor **1734** can be included within a processing cluster **1714**. In at least one embodiment, graphics multiprocessor **1734** can process data and a data crossbar **1740** can be used to distribute processed data to one of multiple possible destinations, including other shader units. In at least one embodiment, pipeline manager **1732** can facilitate distribution of processed data by specifying destinations for processed data to be distributed via data crossbar **1740**.

In at least one embodiment, each graphics multiprocessor **1734** within processing cluster **1714** can include an identical set of functional execution logic (e.g., arithmetic logic units, load-store units, etc.). In at least one embodiment, functional execution logic can be configured in a pipelined manner in which new instructions can be issued before previous instructions are complete. In at least one embodiment, functional execution logic supports a variety of operations including integer and floating point arithmetic, comparison operations, Boolean operations, bit-shifting, and computation of various algebraic functions. In at least one embodiment, same functional-unit hardware can be leveraged to perform different operations and any combination of functional units may be present.

In at least one embodiment, instructions transmitted to processing cluster **1714** constitute a thread. In at least one embodiment, a set of threads executing across a set of parallel processing engines is a thread group. In at least one embodiment, a thread group executes a common program on different input data. In at least one embodiment, each thread within a thread group can be assigned to a different processing engine within a graphics multiprocessor **1734**. In at least one embodiment, a thread group may include fewer threads than a number of processing engines within graphics multiprocessor **1734**. In at least one embodiment, when a thread group includes fewer threads than a number of processing engines, one or more of processing engines may be idle during cycles in which that thread group is being processed. In at least one embodiment, a thread group may also include more threads than a number of processing engines within graphics multiprocessor **1734**. In at least one embodiment, when a thread group includes more threads than number of processing engines within graphics multiprocessor **1734**, processing can be performed over consecutive clock cycles. In at least one embodiment, multiple thread groups can be executed concurrently on a graphics multiprocessor **1734**.

In at least one embodiment, graphics multiprocessor **1734** includes an internal cache memory to perform load and store operations. In at least one embodiment, graphics multiprocessor **1734** can forego an internal cache and use a cache memory (e.g., L1 cache **1748**) within processing cluster **1714**. In at least one embodiment, each graphics multiprocessor **1734** also has access to L2 caches within partition units (e.g., partition units **1720**A-**1720**N of FIG. **17**A) that are shared among all processing clusters **1714** and may be used to transfer data between threads. In at least one embodiment, graphics multiprocessor **1734** may also access off-chip global memory, which can include one or more of local parallel processor memory and/or system memory. In at least one embodiment, any memory external to parallel processing unit **1702** may be used as global memory. In at least one embodiment, processing cluster **1714** includes multiple instances of graphics multiprocessor **1734** and can share common instructions and data, which may be stored in L1 cache **1748**.

In at least one embodiment, each processing cluster **1714** may include an MMU **1745** (memory management unit) that is configured to map virtual addresses into physical addresses. In at least one embodiment, one or more instances of MMU **1745** may reside within memory interface **1718** of FIG. **17**A. In at least one embodiment, MMU **1745** includes a set of page table entries (PTEs) used to map a virtual address to a physical address of a tile and optionally a cache line index. In at least one embodiment, MMU **1745** may include address translation lookaside buffers (TLB) or caches that may reside within graphics multiprocessor **1734** or L1 **1748** cache or processing cluster **1714**. In at least one embodiment, a physical address is processed to distribute surface data access locally to allow for efficient request interleaving among partition units. In at least one embodiment, a cache line index may be used to determine whether a request for a cache line is a hit or miss.

In at least one embodiment, a processing cluster **1714** may be configured such that each graphics multiprocessor **1734** is coupled to a texture unit **1736** for performing texture mapping operations, e.g., determining texture sample positions, reading texture data, and filtering texture data. In at least one embodiment, texture data is read from an internal texture L1 cache (not shown) or from an L1 cache within graphics multiprocessor **1734** and is fetched from an L2 cache, local parallel processor memory, or system memory, as needed. In at least one embodiment, each graphics multiprocessor **1734** outputs processed tasks to data crossbar **1740** to provide processed task to another processing cluster **1714** for further processing or to store processed task in an L2 cache, local parallel processor memory, or system memory via memory crossbar **1716**. In at least one embodiment, a preROP **1742** (pre-raster operations unit) is configured to receive data from graphics multiprocessor **1734**, and direct data to ROP units, which may be located with partition units as described herein (e.g., partition units **1720**A-**1720**N of FIG. **17**A). In at least one embodiment, preROP **1742** unit can perform optimizations for color blending, organizing pixel color data, and performing address translations.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **17**D shows a graphics multiprocessor **1734** according to at least one embodiment. In at least one embodiment,

graphics multiprocessor **1734** couples with pipeline manager **1732** of processing cluster **1714**. In at least one embodiment, graphics multiprocessor **1734** has an execution pipeline including but not limited to an instruction cache **1752**, an instruction unit **1754**, an address mapping unit **1756**, a register file **1758**, one or more general purpose graphics processing unit (GPGPU) cores **1762**, and one or more load/store units **1766**, where one or more load/store units **1766** can perform load/store operations to load/store instructions corresponding to performing an operation. In at least one embodiment, GPGPU cores **1762** and load/store units **1766** are coupled with cache memory **1772** and shared memory **1770** via a memory and cache interconnect **1768**.

In at least one embodiment, instruction cache **1752** receives a stream of instructions to execute from pipeline manager **1732**. In at least one embodiment, instructions are cached in instruction cache **1752** and dispatched for execution by an instruction unit **1754**. In at least one embodiment, instruction unit **1754** can dispatch instructions as thread groups (e.g., warps, wavefronts, waves), with each thread of thread group assigned to a different execution unit within GPGPU cores **1762**. In at least one embodiment, an instruction can access any of a local, shared, or global address space by specifying an address within a unified address space. In at least one embodiment, address mapping unit **1756** can be used to translate addresses in a unified address space into a distinct memory address that can be accessed by load/store units **1766**.

In at least one embodiment, register file **1758** provides a set of registers for functional units of graphics multiprocessor **1734**. In at least one embodiment, register file **1758** provides temporary storage for operands connected to data paths of functional units (e.g., GPGPU cores **1762**, load/store units **1766**) of graphics multiprocessor **1734**. In at least one embodiment, register file **1758** is divided between each of functional units such that each functional unit is allocated a dedicated portion of register file **1758**. In at least one embodiment, register file **1758** is divided between different warps (which may be referred to as wavefronts and/or waves) being executed by graphics multiprocessor **1734**.

In at least one embodiment, GPGPU cores **1762** can each include floating point units (FPUs) and/or integer arithmetic logic units (ALUs) that are used to execute instructions of graphics multiprocessor **1734**. In at least one embodiment, GPGPU cores **1762** can be similar in architecture or can differ in architecture. In at least one embodiment, a first portion of GPGPU cores **1762** include a single precision FPU and an integer ALU while a second portion of GPGPU cores include a double precision FPU. In at least one embodiment, FPUs can implement IEEE 754-2008 standard floating point arithmetic or enable variable precision floating point arithmetic. In at least one embodiment, graphics multiprocessor **1734** can additionally include one or more fixed function or special function units to perform specific functions such as copy rectangle or pixel blending operations. In at least one embodiment, one or more of GPGPU cores **1762** can also include fixed or special function logic.

In at least one embodiment, GPGPU cores **1762** include SIMD logic capable of performing a single instruction on multiple sets of data. In at least one embodiment, GPGPU cores **1762** can physically execute SIMD4, SIMD8, and SIMD16 instructions and logically execute SIMD1, SIMD2, and SIMD32 instructions. In at least one embodiment, SIMD instructions for GPGPU cores can be generated at compile time by a shader compiler or automatically generated when executing programs written and compiled for single program multiple data (SPMD) or SIMT architectures. In at least one embodiment, multiple threads of a program configured for an SIMT execution model can executed via a single SIMD instruction. For example, in at least one embodiment, eight SIMT threads that perform same or similar operations can be executed in parallel via a single SIMD8 logic unit.

In at least one embodiment, memory and cache interconnect **1768** is an interconnect network that connects each functional unit of graphics multiprocessor **1734** to register file **1758** and to shared memory **1770**. In at least one embodiment, memory and cache interconnect **1768** is a crossbar interconnect that allows load/store unit **1766** to implement load and store operations between shared memory **1770** and register file **1758**. In at least one embodiment, register file **1758** can operate at a same frequency as GPGPU cores **1762**, thus data transfer between GPGPU cores **1762** and register file **1758** can have very low latency. In at least one embodiment, shared memory **1770** can be used to enable communication between threads that execute on functional units within graphics multiprocessor **1734**. In at least one embodiment, cache memory **1772** can be used as a data cache for example, to cache texture data communicated between functional units and texture unit **1736**. In at least one embodiment, shared memory **1770** can also be used as a program managed cache. In at least one embodiment, threads executing on GPGPU cores **1762** can programmatically store data within shared memory in addition to automatically cached data that is stored within cache memory **1772**.

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

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **18** illustrates a multi-GPU computing system **1800**, according to at least one embodiment. In at least one embodiment, multi-GPU computing system **1800** can include a processor **1802** coupled to multiple general purpose graphics processing units (GPGPUs) **1806**A-D via a host interface switch **1804**. In at least one embodiment, host interface switch **1804** is a PCI express switch device that couples processor **1802** to a PCI express bus over which processor **1802** can communicate with GPGPUs **1806**A-D. In at least one embodiment, GPGPUs **1806**A-D can interconnect via a set of high-speed point-to-point GPU-to-GPU links **1816**. In at least one embodiment, GPU-to-GPU links **1816** connect to each of GPGPUs **1806**A-D via a dedicated GPU link. In at least one embodiment, P2P GPU links **1816** enable direct communication between each of GPGPUs **1806**A-D without requiring communication over host inter-

face bus **1804** to which processor **1802** is connected. In at least one embodiment, with GPU-to-GPU traffic directed to P2P GPU links **1816**, host interface bus **1804** remains available for system memory access or to communicate with other instances of multi-GPU computing system **1800**, for example, via one or more network devices. While in at least one embodiment GPGPUs **1806**A-D connect to processor **1802** via host interface switch **1804**, in at least one embodiment processor **1802** includes direct support for P2P GPU links **1816** and can connect directly to GPGPUs **1806**A-D.

In at least one embodiment, multi-GPU computing system **1800** includes one or more graphics cores **1500**.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **19** is a block diagram of a graphics processor **1900**, according to at least one embodiment. In at least one embodiment, graphics processor **1900** includes a ring interconnect **1902**, a pipeline front-end **1904**, a media engine **1937**, and graphics cores **1980**A-**1980**N. In at least one embodiment, ring interconnect **1902** couples graphics processor **1900** to other processing units, including other graphics processors or one or more general-purpose processor cores. In at least one embodiment, graphics processor **1900** is one of many processors integrated within a multi-core processing system. In at least one embodiment, graphics processor **1900** includes graphics core **1500**.

In at least one embodiment, graphics processor **1900** receives batches of commands via ring interconnect **1902**. In at least one embodiment, incoming commands are interpreted by a command streamer **1903** in pipeline front-end **1904**. In at least one embodiment, graphics processor **1900** includes scalable execution logic to perform 3D geometry processing and media processing via graphics core(s) **1980**A-**1980**N. In at least one embodiment, for 3D geometry processing commands, command streamer **1903** supplies commands to geometry pipeline **1936**. In at least one embodiment, for at least some media processing commands, command streamer **1903** supplies commands to a video front end **1934**, which couples with media engine **1937**. In at least one embodiment, media engine **1937** includes a Video Quality Engine (VQE) **1930** for video and image post-processing and a multi-format encode/decode (MFX) **1933** engine to provide hardware-accelerated media data encoding and decoding. In at least one embodiment, geometry pipeline **1936** and media engine **1937** each generate execution threads for thread execution resources provided by at least one graphics core **1980**.

In at least one embodiment, graphics processor **1900** includes scalable thread execution resources featuring graphics cores **1980**A-**1980**N (which can be modular and are sometimes referred to as core slices), each having multiple sub-cores **1950**A-**50**N, **1960**A-**1960**N (sometimes referred to as core sub-slices). In at least one embodiment, graphics processor **1900** can have any number of graphics cores **1980**A. In at least one embodiment, graphics processor **1900** includes a graphics core **1980**A having at least a first sub-core **1950**A and a second sub-core **1960**A. In at least one embodiment, graphics processor **1900** is a low power processor with a single sub-core (e.g., **1950**A). In at least one embodiment, graphics processor **1900** includes multiple graphics cores **1980**A-**1980**N, each including a set of first sub-cores **1950**A-**1950**N and a set of second sub-cores **1960**A-**1960**N. In at least one embodiment, each sub-core in first sub-cores **1950**A-**1950**N includes at least a first set of execution units **1952**A-**1952**N and media/texture samplers **1954**A-**1954**N. In at least one embodiment, each sub-core in second sub-cores **1960**A-**1960**N includes at least a second set of execution units **1962**A-**1962**N and samplers **1964**A-**1964**N. In at least one embodiment, each sub-core **1950**A-**1950**N, **1960**A-**1960**N shares a set of shared resources **1970**A-**1970**N. In at least one embodiment, shared resources include shared cache memory and pixel operation logic. In at least one embodiment, graphics processor **1900** includes load/store units in pipeline front-end **1904**.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **20** is a block diagram illustrating micro-architecture for a processor **2000** that may include logic circuits to perform instructions, according to at least one embodiment. In at least one embodiment, processor **2000** may perform instructions, including x86 instructions, ARM instructions, specialized instructions for application-specific integrated circuits (ASICs), etc. In at least one embodiment, processor **2000** may include registers to store packed data, such as 64-bit wide MMX™ registers in microprocessors enabled with MMX technology from Intel Corporation of Santa Clara, Calif. In at least one embodiment, MMX registers, available in both integer and floating point forms, may operate with packed data elements that accompany single instruction, multiple data ("SIMD") and streaming SIMD extensions ("SSE") instructions. In at least one embodiment, 128-bit wide XMM registers relating to SSE2, SSE3, SSE4, AVX, or beyond (referred to generically as "SSEx") technology may hold such packed data operands. In at least one embodiment, processor **2000** may perform instructions to accelerate machine learning or deep learning algorithms, training, or inferencing.

In at least one embodiment, processor **2000** includes an in-order front end ("front end") **2001** to fetch instructions to be executed and prepare instructions to be used later in a processor pipeline. In at least one embodiment, front end **2001** may include several units. In at least one embodiment, an instruction prefetcher **2026** fetches instructions from memory and feeds instructions to an instruction decoder **2028** which in turn decodes or interprets instructions. For example, in at least one embodiment, instruction decoder **2028** decodes a received instruction into one or more operations called "micro-instructions" or "micro-operations" (also called "micro ops" or "uops" or "μ-ops") that a machine may execute. In at least one embodiment, instruction decoder **2028** parses an instruction into an opcode and corresponding data and control fields that may be used by micro-architecture to perform operations in accordance with at least one embodiment. In at least one embodiment, a trace cache **2030** may assemble decoded uops into program ordered sequences or traces in a uop queue **2034** for execution. In at least one embodiment, when trace cache **2030** encounters a complex instruction, a microcode ROM **2032** provides uops needed to complete an operation.

In at least one embodiment, some instructions may be converted into a single micro-op, whereas others need several micro-ops to complete full operation. In at least one embodiment, if more than four micro-ops are needed to complete an instruction, instruction decoder **2028** may access microcode ROM **2032** to perform that instruction. In at least one embodiment, an instruction may be decoded into a small number of micro-ops for processing at instruction decoder **2028**. In at least one embodiment, an instruction may be stored within microcode ROM **2032** should a number of micro-ops be needed to accomplish such opera-

tion. In at least one embodiment, trace cache **2030** refers to an entry point programmable logic array ("PLA") to determine a correct micro-instruction pointer for reading microcode sequences to complete one or more instructions from microcode ROM **2032** in accordance with at least one embodiment. In at least one embodiment, after microcode ROM **2032** finishes sequencing micro-ops for an instruction, front end **2001** of a machine may resume fetching micro-ops from trace cache **2030**.

In at least one embodiment, out-of-order execution engine ("out of order engine") **2003** may prepare instructions for execution. In at least one embodiment, out-of-order execution logic has a number of buffers to smooth out and re-order flow of instructions to optimize performance as they go down a pipeline and get scheduled for execution. In at least one embodiment, out-of-order execution engine **2003** includes, without limitation, an allocator/register renamer **2040**, a memory uop queue **2042**, an integer/floating point uop queue **2044**, a memory scheduler **2046**, a fast scheduler **2002**, a slow/general floating point scheduler ("slow/general FP scheduler") **2004**, and a simple floating point scheduler ("simple FP scheduler") **2006**. In at least one embodiment, fast schedule **2002**, slow/general floating point scheduler **2004**, and simple floating point scheduler **2006** are also collectively referred to herein as "uop schedulers **2002**, **2004**, **2006**." In at least one embodiment, allocator/register renamer **2040** allocates machine buffers and resources that each uop needs in order to execute. In at least one embodiment, allocator/register renamer **2040** renames logic registers onto entries in a register file. In at least one embodiment, allocator/register renamer **2040** also allocates an entry for each uop in one of two uop queues, memory uop queue **2042** for memory operations and integer/floating point uop queue **2044** for non-memory operations, in front of memory scheduler **2046** and uop schedulers **2002**, **2004**, **2006**. In at least one embodiment, uop schedulers **2002**, **2004**, **2006**, determine when a uop is ready to execute based on readiness of their dependent input register operand sources and availability of execution resources uops need to complete their operation. In at least one embodiment, fast scheduler **2002** may schedule on each half of a main clock cycle while slow/general floating point scheduler **2004** and simple floating point scheduler **2006** may schedule once per main processor clock cycle. In at least one embodiment, uop schedulers **2002**, **2004**, **2006** arbitrate for dispatch ports to schedule uops for execution.

In at least one embodiment, execution block **2011** includes, without limitation, an integer register file/bypass network **2008**, a floating point register file/bypass network ("FP register file/bypass network") **2010**, address generation units ("AGUs") **2012** and **2014**, fast Arithmetic Logic Units (ALUs) ("fast ALUs") **2016** and **2018**, a slow Arithmetic Logic Unit ("slow ALU") **2020**, a floating point ALU ("FP") **2022**, and a floating point move unit ("FP move") **2024**. In at least one embodiment, integer register file/bypass network **2008** and floating point register file/bypass network **2010** are also referred to herein as "register files **2008**, **2010**." In at least one embodiment, AGUSs **2012** and **2014**, fast ALUs **2016** and **2018**, slow ALU **2020**, floating point ALU **2022**, and floating point move unit **2024** are also referred to herein as "execution units **2012**, **2014**, **2016**, **2018**, **2020**, **2022**, and **2024**." In at least one embodiment, execution block **2011** may include, without limitation, any number (including zero) and type of register files, bypass networks, address generation units, and execution units, in any combination.

In at least one embodiment, register networks **2008**, **2010** may be arranged between uop schedulers **2002**, **2004**, **2006**, and execution units **2012**, **2014**, **2016**, **2018**, **2020**, **2022**, and **2024**. In at least one embodiment, integer register file/bypass network **2008** performs integer operations. In at least one embodiment, floating point register file/bypass network **2010** performs floating point operations. In at least one embodiment, each of register networks **2008**, **2010** may include, without limitation, a bypass network that may bypass or forward just completed results that have not yet been written into a register file to new dependent uops. In at least one embodiment, register networks **2008**, **2010** may communicate data with each other. In at least one embodiment, integer register file/bypass network **2008** may include, without limitation, two separate register files, one register file for a low-order thirty-two bits of data and a second register file for a high order thirty-two bits of data. In at least one embodiment, floating point register file/bypass network **2010** may include, without limitation, 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

In at least one embodiment, execution units **2012**, **2014**, **2016**, **2018**, **2020**, **2022**, **2024** may execute instructions. In at least one embodiment, register networks **2008**, **2010** store integer and floating point data operand values that micro-instructions need to execute. In at least one embodiment, processor **2000** may include, without limitation, any number and combination of execution units **2012**, **2014**, **2016**, **2018**, **2020**, **2022**, **2024**. In at least one embodiment, floating point ALU **2022** and floating point move unit **2024**, may execute floating point, MMX, SIMD, AVX and SSE, or other operations, including specialized machine learning instructions. In at least one embodiment, floating point ALU **2022** may include, without limitation, a 64-bit by 64-bit floating point divider to execute divide, square root, and remainder micro ops. In at least one embodiment, instructions involving a floating point value may be handled with floating point hardware. In at least one embodiment, ALU operations may be passed to fast ALUs **2016**, **2018**. In at least one embodiment, fast ALUS **2016**, **2018** may execute fast operations with an effective latency of half a clock cycle. In at least one embodiment, most complex integer operations go to slow ALU **2020** as slow ALU **2020** may include, without limitation, integer execution hardware for long-latency type of operations, such as a multiplier, shifts, flag logic, and branch processing. In at least one embodiment, memory load/store operations may be executed by AGUs **2012**, **2014**. In at least one embodiment, fast ALU **2016**, fast ALU **2018**, and slow ALU **2020** may perform integer operations on 64-bit data operands. In at least one embodiment, fast ALU **2016**, fast ALU **2018**, and slow ALU **2020** may be implemented to support a variety of data bit sizes including sixteen, thirty-two, 128, 256, etc. In at least one embodiment, floating point ALU **2022** and floating point move unit **2024** may be implemented to support a range of operands having bits of various widths, such as 128-bit wide packed data operands in conjunction with SIMD and multimedia instructions.

In at least one embodiment, uop schedulers **2002**, **2004**, **2006** dispatch dependent operations before a parent load has finished executing. In at least one embodiment, as uops may be speculatively scheduled and executed in processor **2000**, processor **2000** may also include logic to handle memory misses. In at least one embodiment, if a data load misses in a data cache, there may be dependent operations in flight in a pipeline that have left a scheduler with temporarily incorrect data. In at least one embodiment, a replay mechanism tracks and re-executes instructions that use incorrect data. In at least one embodiment, dependent operations might need to be replayed and independent ones may be allowed to

complete. In at least one embodiment, schedulers and a replay mechanism of at least one embodiment of a processor may also be designed to catch instruction sequences for text string comparison operations.

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

In at least one embodiment, processor **2000** or each core of processor **2000** includes one or more prefetchers, one or more fetchers, one or more pre-decoders, one or more decoders to decode data (e.g., instructions), one or more instruction queues to process instructions (e.g., corresponding to operations or API calls), one or more micro-operation (μOP) cache to store μOPs, one or more micro-operation (μOP) queues, an in-order execution engine, one or more load buffers, one or more store buffers, one or more reorder buffers, one or more fill buffers, an out-of-order execution engine, one or more ports, one or more shift and/or shifter units, one or more fused multiply accumulate (FMA) units, one or more load and store units ("LSUs") to perform load of store operations corresponding to loading/storing data (e.g., instructions) to perform an operation (e.g., perform an API, an API call), one or more matrix multiply accumulate (MMA) units, and/or one or more shuffle units to perform any function further described herein with respect to said processor **2000**. In at least one embodiment processor **2000** can access, use, perform, or execute instructions corresponding to calling an API.

In at least one embodiment, processor **2000** includes one or more ultra path interconnects (UPIs), e.g., that is a point-to-point processor interconnect; one or more PCIe's; one or more accelerators to accelerate computations or operations; and/or one or more memory controllers. In at least one embodiment, processor **2000** includes a shared last level cache (LLC) that is coupled to one or more memory controllers, which can enable shared memory access across processor cores.

In at least one embodiment, processor **2000** or a core of processor **2000** has a mesh architecture where processor cores, on-chip caches, memory controllers, and I/O controllers are organized in rows and columns, with wires and switches connecting them at each intersection to allow for turns. In at least one embodiment, processor **2000** has a one or more higher memory bandwidths (HMBs, e.g., HMBe) to store data or cache data, e.g., in Double Data Rate 5 Synchronous Dynamic Random-Access Memory (DDR5 SDRAM). In at least one embodiment, one or more components of processor **2000** are interconnected using compute express link (CXL) interconnects. In at least one embodiment, a memory controller uses a "least recently used" (LRU) approach to determine what gets stored in a cache. In at least one embodiment, processor **2000** includes one or more PCIe's (e.g., PCIe 5.0).

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **21** illustrates a deep learning application processor **2100**, according to at least one embodiment. In at least one embodiment, deep learning application processor **2100** uses instructions that, if executed by deep learning application processor **2100**, cause deep learning application processor **2100** to perform some or all of processes and techniques described throughout this disclosure. In at least one embodiment, deep learning application processor **2100** is an application-specific integrated circuit (ASIC). In at least one embodiment, application processor **2100** performs matrix multiply operations either "hard-wired" into hardware as a result of performing one or more instructions or both. In at least one embodiment, deep learning application processor **2100** includes, without limitation, processing clusters **2110(1)-2110(12)**, Inter-Chip Links ("ICLs") **2120(1)-2120(12)**, Inter-Chip Controllers ("ICCs") **2130(1)-2130(2)**, high-bandwidth memory second generation ("HBM2") **2140(1)-2140(4)**, memory controllers ("Mem Ctrlrs") **2142(1)-2142(4)**, high bandwidth memory physical layer ("HBM PHY") **2144(1)-2144(4)**, a management-controller central processing unit ("management-controller CPU") **2150**, a Serial Peripheral Interface, Inter-Integrated Circuit, and General Purpose Input/Output block ("SPI, I^2C, GPIO") **2160**, a peripheral component interconnect express controller and direct memory access block ("PCIe Controller and DMA") **2170**, and a sixteen-lane peripheral component interconnect express port ("PCI Express×16") **2180**.

In at least one embodiment, processing clusters **2110** may perform deep learning operations, including inference or prediction operations based on weight parameters calculated one or more training techniques, including those described herein. In at least one embodiment, each processing cluster **2110** may include, without limitation, any number and type of processors. In at least one embodiment, deep learning application processor **2100** may include any number and type of processing clusters **2100**. In at least one embodiment, Inter-Chip Links **2120** are bi-directional. In at least one embodiment, Inter-Chip Links **2120** and Inter-Chip Controllers **2130** enable multiple deep learning application processors **2100** to exchange information, including activation information resulting from performing one or more machine learning algorithms embodied in one or more neural networks. In at least one embodiment, deep learning application processor **2100** may include any number (including zero) and type of ICLs **2120** and ICCs **2130**.

In at least one embodiment, HBM2s **2140** provide a total of 32 Gigabytes (GB) of memory. In at least one embodiment, HBM2 **2140**(*i*) is associated with both memory controller **2142**(*i*) and HBM PHY **2144**(*i*) where "i" is an arbitrary integer. In at least one embodiment, any number of HBM2s **2140** may provide any type and total amount of high bandwidth memory and may be associated with any number (including zero) and type of memory controllers **2142** and HBM PHYs **2144**. In at least one embodiment, SPI, I^2C, GPIO **2160**, PCIe Controller and DMA **2170**, and/or PCIe **2180** may be replaced with any number and type of blocks that enable any number and type of communication standards in any technically feasible fashion.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **22** is a block diagram of a neuromorphic processor **2200**, according to at least one embodiment. In at least one embodiment, neuromorphic processor **2200** may receive one or more inputs from sources external to neuromorphic processor **2200**. In at least one embodiment, these inputs may be transmitted to one or more neurons **2202** within neuromorphic processor **2200**. In at least one embodiment, neurons **2202** and components thereof may be implemented using circuitry or logic, including one or more arithmetic logic units (ALUs). In at least one embodiment, neuromorphic processor **2200** may include, without limitation, thousands or millions of instances of neurons **2202**, but any suitable number of neurons **2202** may be used. In at least one embodiment, each instance of neuron **2202** may include a neuron input **2204** and a neuron output **2206**. In at least one embodiment, neurons **2202** may generate outputs that may be transmitted to inputs of other instances of neurons **2202**. For example, in at least one embodiment, neuron inputs **2204** and neuron outputs **2206** may be interconnected via synapses **2208**.

In at least one embodiment, neurons **2202** and synapses **2208** may be interconnected such that neuromorphic processor **2200** operates to process or analyze information received by neuromorphic processor **2200**. In at least one embodiment, neurons **2202** may transmit an output pulse (or "fire" or "spike") when inputs received through neuron input **2204** exceed a threshold. In at least one embodiment, neurons **2202** may sum or integrate signals received at neuron inputs **2204**. For example, in at least one embodiment, neurons **2202** may be implemented as leaky integrate-and-fire neurons, wherein if a sum (referred to as a "membrane potential") exceeds a threshold value, neuron **2202** may generate an output (or "fire") using a transfer function such as a sigmoid or threshold function. In at least one embodiment, a leaky integrate-and-fire neuron may sum signals received at neuron inputs **2204** into a membrane potential and may also apply a decay factor (or leak) to reduce a membrane potential. In at least one embodiment, a leaky integrate-and-fire neuron may fire if multiple input signals are received at neuron inputs **2204** rapidly enough to exceed a threshold value (i.e., before a membrane potential decays too low to fire). In at least one embodiment, neurons **2202** may be implemented using circuits or logic that receive inputs, integrate inputs into a membrane potential, and decay a membrane potential. In at least one embodiment, inputs may be averaged, or any other suitable transfer function may be used. Furthermore, in at least one embodiment, neurons **2202** may include, without limitation, comparator circuits or logic that generate an output spike at neuron output **2206** when result of applying a transfer function to neuron input **2204** exceeds a threshold. In at least one embodiment, once neuron **2202** fires, it may disregard previously received input information by, for example, resetting a membrane potential to 0 or another suitable default value. In at least one embodiment, once membrane potential is reset to 0, neuron **2202** may resume normal operation after a suitable period of time (or refractory period).

In at least one embodiment, neurons **2202** may be interconnected through synapses **2208**. In at least one embodiment, synapses **2208** may operate to transmit signals from an output of a first neuron **2202** to an input of a second neuron **2202**. In at least one embodiment, neurons **2202** may transmit information over more than one instance of synapse **2208**. In at least one embodiment, one or more instances of neuron output **2206** may be connected, via an instance of synapse **2208**, to an instance of neuron input **2204** in same neuron **2202**. In at least one embodiment, an instance of neuron **2202** generating an output to be transmitted over an instance of synapse **2208** may be referred to as a "pre-synaptic neuron" with respect to that instance of synapse **2208**. In at least one embodiment, an instance of neuron **2202** receiving an input transmitted over an instance of synapse **2208** may be referred to as a "post-synaptic neuron" with respect to that instance of synapse **2208**. Because an instance of neuron **2202** may receive inputs from one or more instances of synapse **2208**, and may also transmit outputs over one or more instances of synapse **2208**, a single instance of neuron **2202** may therefore be both a "pre-synaptic neuron" and "post-synaptic neuron," with respect to various instances of synapses **2208**, in at least one embodiment.

In at least one embodiment, neurons **2202** may be organized into one or more layers. In at least one embodiment, each instance of neuron **2202** may have one neuron output **2206** that may fan out through one or more synapses **2208** to one or more neuron inputs **2204**. In at least one embodiment, neuron outputs **2206** of neurons **2202** in a first layer **2210** may be connected to neuron inputs **2204** of neurons **2202** in a second layer **2212**. In at least one embodiment, layer **2210** may be referred to as a "feed-forward layer." In at least one embodiment, each instance of neuron **2202** in an instance of first layer **2210** may fan out to each instance of neuron **2202** in second layer **2212**. In at least one embodiment, first layer **2210** may be referred to as a "fully connected feed-forward layer." In at least one embodiment, each instance of neuron **2202** in an instance of second layer **2212** may fan out to fewer than all instances of neuron **2202** in a third layer **2214**. In at least one embodiment, second layer **2212** may be referred to as a "sparsely connected feed-forward layer." In at least one embodiment, neurons **2202** in second layer **2212** may fan out to neurons **2202** in multiple other layers, including to neurons **2202** also in second layer **2212**. In at least one embodiment, second layer **2212** may be referred to as a "recurrent layer." In at least one embodiment, neuromorphic processor **2200** may include, without limitation, any suitable combination of recurrent layers and feed-forward layers, including, without limitation, both sparsely connected feed-forward layers and fully connected feed-forward layers.

In at least one embodiment, neuromorphic processor **2200** may include, without limitation, a reconfigurable interconnect architecture or dedicated hard-wired interconnects to connect synapse **2208** to neurons **2202**. In at least one embodiment, neuromorphic processor **2200** may include, without limitation, circuitry or logic that allows synapses to be allocated to different neurons **2202** as needed based on neural network topology and neuron fan-in/out. For example, in at least one embodiment, synapses **2208** may be connected to neurons **2202** using an interconnect fabric, such as network-on-chip, or with dedicated connections. In at least one embodiment, synapse interconnections and components thereof may be implemented using circuitry or logic.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **23** is a block diagram of a processing system, according to at least one embodiment. In at least one embodiment, system **2300** includes one or more processors **2302** and one or more graphics processors **2308**, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors **2302** or processor cores **2307**. In at least one

embodiment, system **2300** is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices. In at least one embodiment, one or more graphics processors **2308** include one or more graphics cores **1500**.

In at least one embodiment, system **2300** can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming console, a handheld game console, or an online game console. In at least one embodiment, system **2300** is a mobile phone, a smart phone, a tablet computing device or a mobile Internet device. In at least one embodiment, processing system **2300** can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, a smart eyewear device, an augmented reality device, or a virtual reality device. In at least one embodiment, processing system **2300** is a television or set top box device having one or more processors **2302** and a graphical interface generated by one or more graphics processors **2308**.

In at least one embodiment, one or more processors **2302** each include one or more processor cores **2307** to process instructions which, when executed, perform operations for system and user software. In at least one embodiment, each of one or more processor cores **2307** is configured to process a specific instruction sequence **2309**. In at least one embodiment, instruction sequence **2309** may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). In at least one embodiment, processor cores **2307** may each process a different instruction sequence **2309**, which may include instructions to facilitate emulation of other instruction sequences. In at least one embodiment, processor core **2307** may also include other processing devices, such a Digital Signal Processor (DSP).

In at least one embodiment, processor **2302** includes a cache memory **2304**. In at least one embodiment, processor **2302** can have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory is shared among various components of processor **2302**. In at least one embodiment, processor **2302** 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 **2307** using known cache coherency techniques. In at least one embodiment, a register file **2306** is additionally included in processor **2302**, which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). In at least one embodiment, register file **2306** may include general-purpose registers or other registers.

In at least one embodiment, one or more processor(s) **2302** are coupled with one or more interface bus(es) **2310** to transmit communication signals such as address, data, or control signals between processor **2302** and other components in system **2300**. In at least one embodiment, interface bus **2310** can be a processor bus, such as a version of a Direct Media Interface (DMI) bus. In at least one embodiment, interface bus **2310** is not limited to a DMI bus, and may include one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express), memory busses, or other types of interface busses. In at least one embodiment processor(s) **2302** include an integrated memory controller **2316** and a platform controller hub **2330**. In at least one embodiment, memory controller **2316** facilitates communication between a memory device and other components of system **2300**, while platform controller hub (PCH) **2330** provides connections to I/O devices via a local I/O bus.

In at least one embodiment, a memory device **2320** can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as process memory. In at least one embodiment, memory device **2320** can operate as system memory for system **2300**, to store data **2322** and instructions **2321** for use when one or more processors **2302** executes an application or process. In at least one embodiment, memory controller **2316** also couples with an optional external graphics processor **2312**, which may communicate with one or more graphics processors **2308** in processors **2302** to perform graphics and media operations. In at least one embodiment, a display device **2311** can connect to processor(s) **2302**. In at least one embodiment, display device **2311** can include one or more of an internal display device, as in a mobile electronic device or a laptop device, or an external display device attached via a display interface (e.g., DisplayPort, etc.). In at least one embodiment, display device **2311** can include a head mounted display (HMD) such as a stereoscopic display device for use in virtual reality (VR) applications or augmented reality (AR) applications.

In at least one embodiment, platform controller hub **2330** enables peripherals to connect to memory device **2320** and processor **2302** via a high-speed I/O bus. In at least one embodiment, I/O peripherals include, but are not limited to, an audio controller **2346**, a network controller **2334**, a firmware interface **2328**, a wireless transceiver **2326**, touch sensors **2325**, a data storage device **2324** (e.g., hard disk drive, flash memory, etc.). In at least one embodiment, data storage device **2324** can connect via a storage interface (e.g., SATA) or via a peripheral bus, such as a Peripheral Component Interconnect bus (e.g., PCI, PCI Express). In at least one embodiment, touch sensors **2325** can include touch screen sensors, pressure sensors, or fingerprint sensors. In at least one embodiment, wireless transceiver **2326** can be a Wi-Fi transceiver, a Bluetooth transceiver, or a mobile network transceiver such as a 3G, $4G_{i,j}$, or Long Term Evolution (LTE) transceiver. In at least one embodiment, firmware interface **2328** enables communication with system firmware, and can be, for example, a unified extensible firmware interface (UEFI). In at least one embodiment, network controller **2334** can enable a network connection to a wired network. In at least one embodiment, a high-performance network controller (not shown) couples with interface bus **2310**. In at least one embodiment, audio controller **2346** is a multi-channel high definition audio controller. In at least one embodiment, system **2300** includes an optional legacy I/O controller **2340** for coupling legacy (e.g., Personal System 2 (PS/2)) devices to system **2300**. In at least one embodiment, platform controller hub **2330** can also connect to one or more Universal Serial Bus (USB) controllers **2342** connect input devices, such as keyboard and mouse **2343** combinations, a camera **2344**, or other USB input devices.

In at least one embodiment, an instance of memory controller **2316** and platform controller hub **2330** may be integrated into a discreet external graphics processor, such as external graphics processor **2312**. In at least one embodiment, platform controller hub **2330** and/or memory controller **2316** may be external to one or more processor(s) **2302**. For example, in at least one embodiment, system **2300** can include an external memory controller **2316** and platform controller hub **2330**, which may be configured as a memory

controller hub and peripheral controller hub within a system chipset that is in communication with processor(s) **2302**.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **24** is a block diagram of a processor **2400** having one or more processor cores **2402**A-**2402**N, an integrated memory controller **2414**, and an integrated graphics processor **2408**, according to at least one embodiment. In at least one embodiment, processor **2400** can include additional cores up to and including additional core **2402**N represented by dashed lined boxes. In at least one embodiment, each of processor cores **2402**A-**2402**N includes one or more internal cache units **2404**A-**2404**N. In at least one embodiment, each processor core also has access to one or more shared cached units **2406**. In at least one embodiment, graphics processor **2408** includes one or more graphics cores **1500**.

In at least one embodiment, internal cache units **2404**A-**2404**N and shared cache units **2406** represent a cache memory hierarchy within processor **2400**. In at least one embodiment, cache memory units **2404**A-**2404**N may include at least one level of instruction and data cache within each processor core and one or more levels of shared mid-level cache, such as a Level 2 (L2), Level 3 (L3), Level 4 (L4), or other levels of cache, where a highest level of cache before external memory is classified as an LLC. In at least one embodiment, cache coherency logic maintains coherency between various cache units **2406** and **2404**A-**2404**N.

In at least one embodiment, processor **2400** may also include a set of one or more bus controller units **2416** and a system agent core **2410**. In at least one embodiment, bus controller units **2416** manage a set of peripheral buses, such as one or more PCI or PCI express busses. In at least one embodiment, system agent core **2410** provides management functionality for various processor components. In at least one embodiment, system agent core **2410** includes one or more integrated memory controllers **2414** to manage access to various external memory devices (not shown).

In at least one embodiment, one or more of processor cores **2402**A-**2402**N include support for simultaneous multi-threading. In at least one embodiment, system agent core **2410** includes components for coordinating and operating cores **2402**A-**2402**N during multi-threaded processing. In at least one embodiment, system agent core **2410** may additionally include a power control unit (PCU), which includes logic and components to regulate one or more power states of processor cores **2402**A-**2402**N and graphics processor **2408**.

In at least one embodiment, processor **2400** additionally includes graphics processor **2408** to execute graphics processing operations. In at least one embodiment, graphics processor **2408** couples with shared cache units **2406**, and system agent core **2410**, including one or more integrated memory controllers **2414**. In at least one embodiment, system agent core **2410** also includes a display controller **2411** to drive graphics processor output to one or more coupled displays. In at least one embodiment, display controller **2411** may also be a separate module coupled with graphics processor **2408** via at least one interconnect, or may be integrated within graphics processor **2408**.

In at least one embodiment, a ring-based interconnect unit **2412** is used to couple internal components of processor **2400**. In at least one embodiment, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques. In at least one embodiment, graphics processor **2408** couples with ring interconnect **2412** via an I/O link **2413**.

In at least one embodiment, I/O link **2413** 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 **2418**, such as an eDRAM module. In at least one embodiment, each of processor cores **2402**A-**2402**N and graphics processor **2408** use embedded memory module **2418** as a shared Last Level Cache.

In at least one embodiment, processor cores **2402**A-**2402**N are homogeneous cores executing a common instruction set architecture. In at least one embodiment, processor cores **2402**A-**2402**N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores **2402**A-**2402**N execute a common instruction set, while one or more other cores of processor cores **2402**A-**2402**N executes a subset of a common instruction set or a different instruction set. In at least one embodiment, processor cores **2402**A-**2402**N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. In at least one embodiment, processor **2400** can be implemented on one or more chips or as an SoC integrated circuit.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **25** is a block diagram of a graphics processor **2500**, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. In at least one embodiment, graphics processor **2500** communicates via a memory mapped I/O interface to registers on graphics processor **2500** and with commands placed into memory. In at least one embodiment, graphics processor **2500** includes a memory interface **2514** to access memory. In at least one embodiment, memory interface **2514** is an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory. In at least one embodiment, graphics processor **2500** includes graphics core **1500**.

In at least one embodiment, graphics processor **2500** also includes a display controller **2502** to drive display output data to a display device **2520**. In at least one embodiment, display controller **2502** includes hardware for one or more overlay planes for display device **2520** and composition of multiple layers of video or user interface elements. In at least one embodiment, display device **2520** can be an internal or external display device. In at least one embodiment, display device **2520** is a head mounted display device, such as a virtual reality (VR) display device or an augmented reality (AR) display device. In at least one embodiment, graphics processor **2500** includes a video codec engine **2506** to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

In at least one embodiment, graphics processor **2500** includes a block image transfer (BLIT) engine **2504** to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in

at least one embodiment, 2D graphics operations are performed using one or more components of a graphics processing engine (GPE) **2510**. In at least one embodiment, GPE **2510** is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

In at least one embodiment, GPE **2510** includes a 3D pipeline **2512** for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). In at least one embodiment, 3D pipeline **2512** includes programmable and fixed function elements that perform various tasks and/or spawn execution threads to a 3D/Media sub-system **2515**. While 3D pipeline **2512** can be used to perform media operations, in at least one embodiment, GPE **2510** also includes a media pipeline **2516** that is used to perform media operations, such as video post-processing and image enhancement.

In at least one embodiment, media pipeline **2516** 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 **2506**. In at least one embodiment, media pipeline **2516** additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system **2515**. In at least one embodiment, spawned threads perform computations for media operations on one or more graphics execution units included in 3D/Media sub-system **2515**.

In at least one embodiment, 3D/Media subsystem **2515** includes logic for executing threads spawned by 3D pipeline **2512** and media pipeline **2516**. In at least one embodiment, 3D pipeline **2512** and media pipeline **2516** send thread execution requests to 3D/Media subsystem **2515**, which includes thread dispatch logic for arbitrating and dispatching various requests to available thread execution resources. In at least one embodiment, execution resources include an array of graphics execution units to process 3D and media threads. In at least one embodiment, 3D/Media subsystem **2515** includes one or more internal caches for thread instructions and data. In at least one embodiment, subsystem **2515** also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

FIG. **26** is a block diagram of a graphics processing engine **2610** of a graphics processor in accordance with at least one embodiment. In at least one embodiment, graphics processing engine (GPE) **2610** is a version of GPE **2510** shown in FIG. **25**. In at least one embodiment, a media pipeline **2616** is optional and may not be explicitly included within GPE **2610**. In at least one embodiment, a separate media and/or image processor is coupled to GPE **2610**.

In at least one embodiment, GPE **2610** is coupled to or includes a command streamer **2603**, which provides a command stream to a 3D pipeline **2612** and/or media pipeline **2616**. In at least one embodiment, command streamer **2603** is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In at least one embodiment, command streamer **2603** receives commands from memory and sends commands to 3D pipeline **2612** and/or media pipeline **2616**. In at least one embodiment, commands are instructions, primitives, or micro-operations fetched from a ring buffer, which stores commands for 3D pipeline **2612** and media pipeline **2616**. In at least one embodiment, a ring buffer can additionally include batch command buffers storing batches of multiple commands. In at least one embodiment, commands for 3D pipeline **2612** can also include references to data stored in memory, such as, but not limited to, vertex and geometry data for 3D pipeline **2612** and/or image data and memory objects for media pipeline **2616**. In at least one embodiment, 3D pipeline **2612** and media pipeline **2616** process commands and data by performing operations or by dispatching one or more execution threads to a graphics core array **2614**. In at least one embodiment, graphics core array **2614** includes one or more blocks of graphics cores (e.g., graphics core(s) **2615**A, graphics core(s) **2615**B), each block including one or more graphics cores. In at least one embodiment, graphics core(s) **2615**A, **2615**B may be referred to as execution units ("EUs").

In at least one embodiment, 3D pipeline **2612** includes fixed function and programmable logic to process one or more shader programs, such as vertex shaders, geometry shaders, pixel shaders, fragment shaders, compute shaders, or other shader programs, by processing instructions and dispatching execution threads to graphics core array **2614**. In at least one embodiment, graphics core array **2614** provides a unified block of execution resources for use in processing shader programs. In at least one embodiment, a multi-purpose execution logic (e.g., execution units) within graphics core(s) **2615**A-**2615**B of graphic core array **2614** includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

In at least one embodiment, graphics core array **2614** also includes execution logic to perform media functions, such as video and/or image processing. In at least one embodiment, execution units additionally include general-purpose logic that is programmable to perform parallel general-purpose computational operations, in addition to graphics processing operations.

In at least one embodiment, output data generated by threads executing on graphics core array **2614** can output data to memory in a unified return buffer (URB) **2618**. In at least one embodiment, URB **2618** can store data for multiple threads. In at least one embodiment, URB **2618** may be used to send data between different threads executing on graphics core array **2614**. In at least one embodiment, URB **2618** may additionally be used for synchronization between threads on graphics core array **2614** and fixed function logic within shared function logic **2620**.

In at least one embodiment, graphics core array **2614** is scalable, such that graphics core array **2614** includes a variable number of graphics cores, each having a variable number of execution units based on a target power and performance level of GPE **2610**. In at least one embodiment, execution resources are dynamically scalable, such that execution resources may be enabled or disabled as needed.

In at least one embodiment, graphics core array **2614** is coupled to shared function logic **2620** that includes multiple resources that are shared between graphics cores in graphics core array **2614**. In at least one embodiment, shared functions performed by shared function logic **2620** are embodied in hardware logic units that provide specialized supplemental functionality to graphics core array **2614**. In at least one embodiment, shared function logic **2620** includes but is not limited to a sampler unit **2621**, a math unit **2622**, and inter-thread communication (ITC) logic **2623**. In at least one embodiment, one or more cache(s) **2625** are included in, or coupled to, shared function logic **2620**.

In at least one embodiment, a shared function is used if demand for a specialized function is insufficient for inclusion within graphics core array **2614**. In at least one embodiment, a single instantiation of a specialized function is used in shared function logic **2620** and shared among other execution resources within graphics core array **2614**. In at least one embodiment, specific shared functions within shared function logic **2620** that are used extensively by graphics core array **2614** may be included within shared function logic **2626** within graphics core array **2614**. In at least one embodiment, shared function logic **2626** within graphics core array **2614** can include some or all logic within shared function logic **2620**. In at least one embodiment, all logic elements within shared function logic **2620** may be duplicated within shared function logic **2626** of graphics core array **2614**. In at least one embodiment, shared function logic **2620** is excluded in favor of shared function logic **2626** within graphics core array **2614**.

These and other such components can be used for generating or synthesizing content, as may involve performing shading operations as part of a ray tracing-based rendering process.

Various embodiments presented herein correspond, at least in part, to the following clauses:

1. A method, comprising:

obtaining lighting data for a plurality of pixel locations of a current frame and a previous frame in a sequence of frames;

determining gradient information for at least a subset of the pixel locations, the gradient information indicating a difference in the lighting data between the current frame and the previous frame;

determining confidence values for the plurality of pixel locations based, at least in part, upon the gradient information; and

determining a weighting of the lighting data, from the current frame and the previous frame, to be used for shading the pixel locations of the current frame based, at least in part, upon the plurality of confidence values.

2. The method of claim **1**, further comprising:

applying a spatial filter to the gradient information to generate spatially-filtered gradient information, wherein the confidence values are determined based, at least in part, upon the spatially-filtered gradient information.

3. The method of claim **2**, wherein the spatial filter comprises at least one of a wide blur kernel or a bilateral blur kernel.

4. The method of claim **2**, wherein the gradient information comprises one or more luminance differences and one or more absolute luminance values, and wherein the spatial filter filters luminance differences independently of the absolute luminance values.

5. The method of claim **2**, further comprising:

normalizing the gradient information, before or after applying the spatial filter, to generate spatially-filtered and normalized gradient information.

6. The method of claim **5**, wherein the gradient information comprises one or more luminance differences and one or more absolute luminance values, and wherein determining the confidence values includes dividing one or more luminance differences by one or more absolute luminance values, and converting an output of the dividing into a value representative of a confidence.

7. The method of claim **5**, wherein the weighting of the lighting data is determined using a denoiser that accepts the confidence values and the lighting data as input.

8. The method of claim **2**, further comprising:

performing normalization before applying the special filter to the gradient information, wherein the spatial filter is to operate on one or more difference ratios or absolute ratios.

9. The method of claim **1**, further comprising:

rendering the current frame using the weighting of the lighting data for the pixel locations of the current frame.

10. The method of claim **1**, wherein the luminance information is computed using one or more material properties of a surface represented at each pixel location.

11. A processor, comprising:

one or more circuits to cause the processor to perform operations comprising:

rendering a current image in a sequence of images;

generating a lighting gradient using lighting data for each of a plurality of pixel regions of the current image with respect to a plurality of corresponding regions of a previous image in the sequence of images;

performing spatial blurring with respect to the lighting gradients to produce a blurred gradient image;

upscaling the blurred gradient image to a target image resolution;

transforming the blurred lighting gradients for individual pixel locations of the upscaled gradient image into confidence values; and

determining, based at least in part upon the confidence values, an extent to which to use the lighting data from the previous image or the current image to render the individual pixel locations of the current image at the target image resolution.

12. The processor of claim **11**, wherein the one or more circuits are to perform operations further comprising:

determining, based at least in part upon shading data for the current image and the previous image, whether to use a current light for the current image or a previous light for the previous image to use to determine luminance values for the plurality of corresponding regions.

13. The processor of claim **12**, wherein the lighting gradients correspond to one or more differences in the determined luminance values.

14. The processor of claim **11**, wherein the spatial blurring is performed using a wide blur kernel or a bilateral blur kernel.

15. The processor of claim **11**, wherein the one or more circuits are to perform operations further comprising:

using a denoiser to determine the extent to which to use the lighting data from the previous image or the current image.

16. The processor of claim **11**, wherein the processor is comprised in at least one of:

a system for performing simulation operations;

a system for performing simulation operations to test or validate autonomous machine applications;

a system for performing digital twin operations;

a system for performing light transport simulation;

a system for rendering graphical output;

a system for performing deep learning operations;

a system implemented using an edge device;

a system for generating or presenting virtual reality (VR) content;

a system for generating or presenting augmented reality (AR) content;

a system for generating or presenting mixed reality (MR) content;

a system incorporating one or more Virtual Machines (VMs);

a system implemented at least partially in a data center;

a system for performing hardware testing using simulation;

a system for synthetic data generation;

a collaborative content creation platform for 3D assets; or

a system implemented at least partially using cloud computing resources.

17. A system, comprising:

one or more processing units to use one or more confidence values to determine an extent to which to use lighting data from a previous image or a current image, in a sequence of images, to render individual pixel locations of a current image, the one or more confidence values determined at least in part by transforming one or more lighting gradients determined using lighting data for each of a plurality of pixel regions of the current image with respect to a plurality of corresponding regions of the previous image and performing spatial blurring with respect to the lighting gradients to produce a blurred gradient image that is upscaled to a target image resolution.

18. The system of claim **17**, wherein the one or more processing units are further to:

determine, based at least in part upon shading data for the current image and the previous image, whether to use the current light for the current image or the previous light for the previous image to use to determine luminance values for the plurality of corresponding regions.

19. The system of claim **18**, wherein the lighting gradients correspond to one or more differences in the determined luminance values.

20. The system of claim **17**, wherein the system comprises at least one of:

a system for performing simulation operations;

a system for performing simulation operations to test or validate autonomous machine applications;

a system for performing digital twin operations;

a system for performing light transport simulation;

a system for rendering graphical output;

a system for performing deep learning operations;

a system implemented using an edge device;

a system for generating or presenting virtual reality (VR) content;

a system for generating or presenting augmented reality (AR) content;

a system for generating or presenting mixed reality (MR) content;

a system incorporating one or more Virtual Machines (VMs);

a system implemented at least partially in a data center;

a system for performing hardware testing using simulation;

a system for synthetic data generation;

a collaborative content creation platform for 3D assets; or

a system implemented at least partially using cloud computing resources.

Other variations are within spirit of present disclosure. Thus, while disclosed techniques are susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in drawings and have been described above in detail. It should be understood, however, that there is no intention to limit disclosure to specific form or forms disclosed, but on contrary, intention is to cover all modifications, alternative constructions, and equivalents falling within spirit and scope of disclosure, as defined in appended claims.

Use of terms "a" and "an" and "the" and similar referents in context of describing disclosed embodiments (especially in context of following claims) are to be construed to cover both singular and plural, unless otherwise indicated herein or clearly contradicted by context, and not as a definition of a term. Terms "comprising," "having," "including," and "containing" are to be construed as open-ended terms (meaning "including, but not limited to,") unless otherwise noted. Term "connected," when unmodified and referring to physical connections, is to be construed as partly or wholly contained within, attached to, or joined together, even if there is something intervening. Recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within range, unless otherwise indicated herein and each separate value is incorporated into specification as if it were individually recited herein. Use of term "set" (e.g., "a set of items") or "subset," unless otherwise noted or contradicted by context, is to be construed as a nonempty collection comprising one or more members. Further, unless otherwise noted or contradicted by context, term "subset" of a corresponding set does not necessarily denote a proper subset of corresponding set, but subset and corresponding set may be equal.

Conjunctive language, such as phrases of form "at least one of A, B, and C," or "at least one of A, B and C," unless specifically stated otherwise or otherwise clearly contradicted by context, is otherwise understood with context as used in general to present that an item, term, etc., may be either A or B or C, or any nonempty subset of set of A and B and C. For instance, in illustrative example of a set having three members, conjunctive phrases "at least one of A, B, and C" and "at least one of A, B and C" refer to any of following sets: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}. Thus, such conjunctive language is not generally intended to imply that certain embodiments require at least one of A, at least one of B, and at least one of C each to be present. In addition, unless otherwise noted or contradicted by context, term "plurality" indicates a state of being plural (e.g., "a plurality of items" indicates multiple items). A plurality is at least two items, but can be more when so indicated either explicitly or by context. Further, unless stated otherwise or otherwise clear from context, phrase "based on" means "based at least in part on" and not "based solely on."

Operations of processes described herein can be performed in any suitable order unless otherwise indicated herein or otherwise clearly contradicted by context. In at least one embodiment, a process such as those processes described herein (or variations and/or combinations thereof) is performed under control of one or more computer systems configured with executable instructions and is implemented as code (e.g., executable instructions, one or more computer programs or one or more applications) executing collectively on one or more processors, by hardware or combinations thereof. In at least one embodiment, code is stored on a computer-readable storage medium, for example, in form of a computer program comprising a plurality of instructions executable by one or more processors. In at least one embodiment, a computer-readable storage medium is a non-transitory computer-readable storage medium that excludes transitory signals (e.g., a propagating transient electric or electromagnetic transmission) but includes non-transitory data storage circuitry (e.g., buffers, cache, and

queues) within transceivers of transitory signals. In at least one embodiment, code (e.g., executable code or source code) is stored on a set of one or more non-transitory computer-readable storage media having stored thereon executable instructions (or other memory to store executable instructions) that, when executed (i.e., as a result of being executed) by one or more processors of a computer system, cause computer system to perform operations described herein. A set of non-transitory computer-readable storage media, in at least one embodiment, comprises multiple non-transitory computer-readable storage media and one or more of individual non-transitory storage media of multiple non-transitory computer-readable storage media lack all of code while multiple non-transitory computer-readable storage media collectively store all of code. In at least one embodiment, executable instructions are executed such that different instructions are executed by different processors—for example, a non-transitory computer-readable storage medium store instructions and a main central processing unit ("CPU") executes some of instructions while a graphics processing unit ("GPU") executes other instructions. In at least one embodiment, different components of a computer system have separate processors and different processors execute different subsets of instructions.

Accordingly, in at least one embodiment, computer systems are configured to implement one or more services that singly or collectively perform operations of processes described herein and such computer systems are configured with applicable hardware and/or software that enable performance of operations. Further, a computer system that implements at least one embodiment of present disclosure is a single device and, in another embodiment, is a distributed computer system comprising multiple devices that operate differently such that distributed computer system performs operations described herein and such that a single device does not perform all operations.

Use of any and all examples, or exemplary language (e.g., "such as") provided herein, is intended merely to better illuminate embodiments of disclosure and does not pose a limitation on scope of disclosure unless otherwise claimed. No language in specification should be construed as indicating any non-claimed element as essential to practice of disclosure.

All references, including publications, patent applications, and patents, cited herein are hereby incorporated by reference to same extent as if each reference were individually and specifically indicated to be incorporated by reference and were set forth in its entirety herein.

In description and claims, terms "coupled" and "connected," along with their derivatives, may be used. It should be understood that these terms may be not intended as synonyms for each other. Rather, in particular examples, "connected" or "coupled" may be used to indicate that two or more elements are in direct or indirect physical or electrical contact with each other. "Coupled" may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other.

Unless specifically stated otherwise, it may be appreciated that throughout specification terms such as "processing," "computing," "calculating," "determining," or like, refer to action and/or processes of a computer or computing system, or similar electronic computing device, that manipulate and/or transform data represented as physical, such as electronic, quantities within computing system's registers and/or memories into other data similarly represented as physical quantities within computing system's memories, registers or other such information storage, transmission or display devices.

In a similar manner, term "processor" may refer to any device or portion of a device that processes electronic data from registers and/or memory and transform that electronic data into other electronic data that may be stored in registers and/or memory. As non-limiting examples, "processor" may be a CPU or a GPU. A "computing platform" may comprise one or more processors. As used herein, "software" processes may include, for example, software and/or hardware entities that perform work over time, such as tasks, threads, and intelligent agents. Also, each process may refer to multiple processes, for carrying out instructions in sequence or in parallel, continuously or intermittently. Terms "system" and "method" are used herein interchangeably insofar as system may embody one or more methods and methods may be considered a system.

In present document, references may be made to obtaining, acquiring, receiving, or inputting analog or digital data into a subsystem, computer system, or computer-implemented machine. Obtaining, acquiring, receiving, or inputting analog and digital data can be accomplished in a variety of ways such as by receiving data as a parameter of a function call or a call to an application programming interface. In some implementations, process of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a serial or parallel interface. In another implementation, process of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a computer network from providing entity to acquiring entity. References may also be made to providing, outputting, transmitting, sending, or presenting analog or digital data. In various examples, process of providing, outputting, transmitting, sending, or presenting analog or digital data can be accomplished by transferring data as an input or output parameter of a function call, a parameter of an application programming interface or interprocess communication mechanism.

Although discussion above sets forth example implementations of described techniques, other architectures may be used to implement described functionality, and are intended to be within scope of this disclosure. Furthermore, although specific distributions of responsibilities are defined above for purposes of discussion, various functions and responsibilities might be distributed and divided in different ways, depending on circumstances.

Furthermore, although subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that subject matter claimed in appended claims is not necessarily limited to specific features or acts described. Rather, specific features and acts are disclosed as exemplary forms of implementing the claims.

What is claimed is:

1. A method, comprising:

obtaining lighting data for a plurality of pixel locations of a current frame and for the plurality of pixel locations of a previous frame, the current and previous frames included in a sequence of frames;

selecting pixel locations of the plurality of pixel locations based at least in part on the lighting data for the plurality of pixel locations of the current frame;

determining, using the selected pixel locations, gradient information for at least a subset of the plurality of pixel

locations, the gradient information indicating a difference in the lighting data between the current frame and the previous frame;

determining one or more confidence values for the plurality of pixel locations based at least in part on the gradient information; and

determining a weighting of the lighting data, to be used for shading the pixel locations of the current frame based at least in part on the one or more confidence values.

2. The method of claim **1**, further comprising:

applying a spatial filter to the gradient information to generate spatially-filtered gradient information, wherein the one or more confidence values are determined based at least in part on the spatially-filtered gradient information.

3. The method of claim **2**, wherein the spatial filter comprises at least one of a wide blur kernel or a bilateral blur kernel.

4. The method of claim **2**, wherein the gradient information comprises one or more luminance differences and one or more absolute luminance values, and wherein the spatial filter filters the one or more luminance differences independently of the one or more absolute luminance values.

5. The method of claim **2**, further comprising:

normalizing the gradient information, before or after applying the spatial filter, to generate spatially-filtered and normalized gradient information.

6. The method of claim **5**, wherein the gradient information comprises one or more luminance differences and one or more absolute luminance values, and wherein determining the one or more confidence values includes dividing one or more luminance differences by one or more absolute luminance values, and converting an output of the dividing into a value representative of a confidence.

7. The method of claim **5**, wherein the weighting of the lighting data is determined using a denoiser that accepts the confidence values and the lighting data as input.

8. The method of claim **2**, further comprising:

performing normalization before applying the spatial filter to the gradient information, wherein the spatial filter is to operate on one or more difference ratios or absolute ratios.

9. The method of claim **1**, further comprising:

rendering the current frame using the weighting of the lighting data for the pixel locations of the current frame.

10. The method of claim **1**, wherein the lighting data is computed using one or more material properties of a surface represented at each pixel location.

11. A processor, comprising:

one or more circuits to cause the processor to perform operations comprising:

rendering a current image in a sequence of images;

generating, based on selected pixel locations of the current image, a lighting gradient using lighting data for at least one pixel region of the current image with respect to a plurality of corresponding regions of a previous image in the sequence of images, the selected pixel locations being selected for use based at least in part on the individual pixel lighting data of the current image;

performing spatial blurring with respect to the lighting gradients to produce a blurred gradient image;

upscaling the blurred gradient image to a target image resolution;

transforming the lighting gradients for individual pixel locations of the upscaled blurred gradient image into confidence values; and

determining, based at least on the confidence values, an extent to use the lighting data from the previous image or the current image to render the individual pixel locations of the current image at the target image resolution.

12. The processor of claim **11**, wherein the one or more circuits are to perform operations further comprising:

determining, based at least on shading data for the current image and the previous image, whether to use a current illuminant for the current image or a previous illuminant for the previous image to determine one or more luminance values for the plurality of corresponding regions.

13. The processor of claim **12**, wherein the lighting gradients correspond to one or more differences in the one or more determined luminance values.

14. The processor of claim **11**, wherein the spatial blurring is performed using a wide blur kernel or a bilateral blur kernel.

15. The processor of claim **11**, wherein the one or more circuits are to perform operations further comprising:

using a denoiser to determine the extent to which to use the lighting data from the previous image or the current image.

16. The processor of claim **11**, wherein the processor is comprised in at least one of:

a system for performing simulation operations;

a system for performing simulation operations to test or validate autonomous machine applications;

a system for performing digital twin operations;

a system for performing light transport simulation;

a system for rendering graphical output;

a system for performing deep learning operations;

a system implemented using an edge device;

a system for generating or presenting virtual reality (VR) content;

a system for generating or presenting augmented reality (AR) content;

a system for generating or presenting mixed reality (MR) content;

a system incorporating one or more Virtual Machines (VMs);

a system implemented at least partially in a data center;

a system for performing hardware testing using simulation;

a system for synthetic data generation;

a collaborative content creation platform for 3D assets; or

a system implemented at least partially using cloud computing resources.

17. A system, comprising:

one or more processing units to use one or more confidence values to determine an extent to which lighting data from a previous image or a current image in a sequence of images is used to render individual pixel locations of a current image, wherein the one or more confidence values are determined at least in part by transforming one or more lighting gradients determined using pixel lighting data for each of a plurality of pixel regions of the current image with respect to a plurality of corresponding regions of the previous image for pixel locations selected based at least in part on the lighting data of the current image and performing

spatial blurring with respect to the lighting gradients to produce a blurred gradient image that is upscaled to a target image resolution.

18. The system of claim **17**, wherein the one or more processing units are further to:

determine, based at least on shading data for the current image and the previous image, whether to use the lighting data for the current image or the lighting data for the previous image to determine the one or more lighting gradients for the plurality of corresponding regions.

19. The system of claim **18**, wherein the lighting gradients correspond to one or more differences in the determined luminance values.

20. The system of claim **17**, wherein the system comprises at least one of:

a system for performing simulation operations;

a system for performing simulation operations to test or validate autonomous machine applications;

a system for performing digital twin operations;

a system for performing light transport simulation;

a system for rendering graphical output;

a system for performing deep learning operations;

a system implemented using an edge device;

a system for generating or presenting virtual reality (VR) content;

a system for generating or presenting augmented reality (AR) content;

a system for generating or presenting mixed reality (MR) content;

a system incorporating one or more Virtual Machines (VMs);

a system implemented at least partially in a data center;

a system for performing hardware testing using simulation;

a system for synthetic data generation;

a collaborative content creation platform for 3D assets; or

a system implemented at least partially using cloud computing resources.

* * * * *