Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2650539 A1 2007/12/06

(21) 2 650 539

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(86) Date de dépôt PCT/PCT Filing Date: 2007/05/25
(87) Date publication PCT/PCT Publication Date: 2007/12/06
(85) Entrée phase nationale/National Entry: 2008/10/27
(86) N° demande PCT/PCT Application No.: US 2007/069803
(87) N° publication PCT/PCT Publication No.: 2007/140338
(30) Priorité/Priority: 2006/05/25 (US11/441,696)

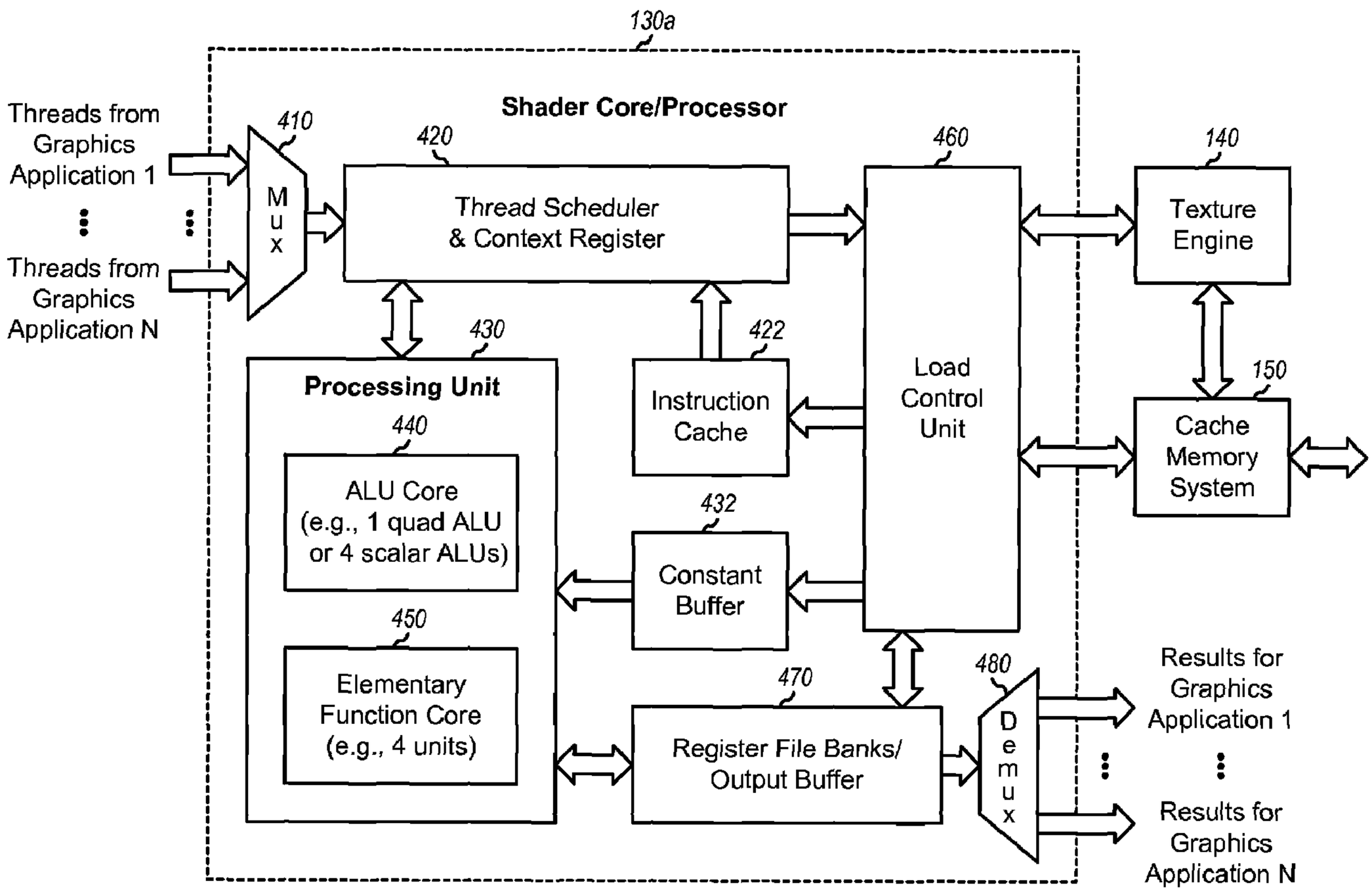
(51) Cl.Int./Int.Cl. *G06F 9/38* (2006.01),
G06T 1/20 (2006.01)

(71) Demandeur/Applicant:
QUALCOMM INCORPORATED, US

(72) Inventeurs/Inventors:
BOURD, ALEXEI V., US;
DU, YUN, US;
YU, CHUN, US;
JIAO, GUOFANG, US

(74) Agent: SMART & BIGGAR

(54) Titre : PROCESSEUR GRAPHIQUE AVEC DES UNITES DE FONCTIONS ARITHMETIQUES ET ELEMENTAIRES
(54) Title: GRAPHICS PROCESSOR WITH ARITHMETIC AND ELEMENTARY FUNCTION UNITS



(57) Abrégé/Abstract:
A graphics processor capable of efficiently performing arithmetic operations and computing elementary functions is described. The graphics processor has at least one arithmetic logic unit (ALU) that can perform arithmetic operations and at least one elementary function unit that can compute elementary functions. The ALU(s) and elementary function unit(s) may be arranged such that they can operate in parallel to improve throughput. The graphics processor may also include fewer elementary function units than ALUs, e.g., four ALUs and a single elementary function unit. The four ALUs may perform an arithmetic operation on (1) four components

(21) 2 650 539

(13) A1

(57) Abrégé(suite)/Abstract(continued):
of an attribute for one pixel or (2) one component of an attribute for four pixels. The single elementary function unit may operate on one component of one pixel at a time. The use of a single elementary function unit may reduce cost while still providing good performance.

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 December 2007 (06.12.2007)

PCT

(10) International Publication Number
WO 2007/140338 A3

(51) International Patent Classification:
G06F 9/38 (2006.01) *G06T 1/20* (2006.01)

(21) International Application Number:
PCT/US2007/069803

(22) International Filing Date: 25 May 2007 (25.05.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/441,696 25 May 2006 (25.05.2006) US

(71) Applicant *(for all designated States except US)*: **QUALCOMM Incorporated** [US/US]; Attn: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BOURD, Alexei, V.** [RU/US]; 10310 Caminito Agadir, San Diego, California 92131 (US). **DU, Yun** [CN/US]; 12341 Katydid Circle, San Diego, California 92129 (US). **YU, Chun** [CN/US]; 11496 Cypress Woods Dr., San Diego, California 92131 (US). **JIAO, Guofang** [CN/US]; 10680 Hunters Glen Dr., San Diego, California 92130 (US).

(74) Agent: **OGROD, Gregory, D.**; Attn: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

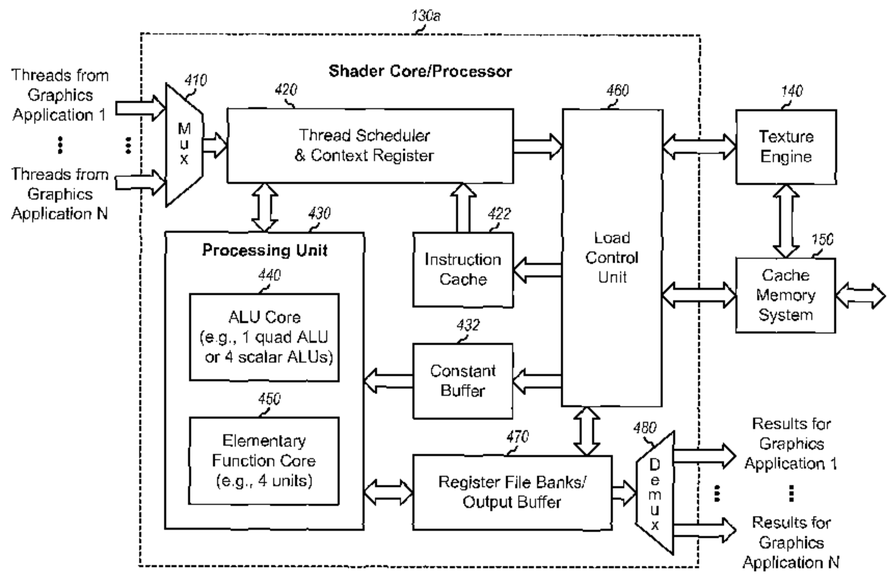
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report*

(88) Date of publication of the international search report:
6 March 2008

(54) Title: GRAPHICS PROCESSOR WITH ARITHMETIC AND ELEMENTARY FUNCTION UNITS



WO 2007/140338 A3

(57) Abstract: A graphics processor capable of efficiently performing arithmetic operations and computing elementary functions is described. The graphics processor has at least one arithmetic logic unit (ALU) that can perform arithmetic operations and at least one elementary function unit that can compute elementary functions. The ALU(s) and elementary function unit(s) may be arranged such that they can operate in parallel to improve throughput. The graphics processor may also include fewer elementary function units than ALUs, e.g., four ALUs and a single elementary function unit. The four ALUs may perform an arithmetic operation on (1) four components of an attribute for one pixel or (2) one component of an attribute for four pixels. The single elementary function unit may operate on one component of one pixel at a time. The use of a single elementary function unit may reduce cost while still providing good performance.

GRAPHICS PROCESSOR WITH ARITHMETIC AND ELEMENTARY FUNCTION UNITS

BACKGROUND

I. Field

[0001] The present disclosure relates generally to circuits, and more specifically to graphics processors.

II. Background

[0002] Graphics processors are widely used to render 2-dimensional (2-D) and 3-dimensional (3-D) images for various applications such as video games, graphics, computer-aided design (CAD), simulation and visualization tools, imaging, etc. A 3-D image may be modeled with surfaces, and each surface may be approximated with polygons (typically triangles). The number of triangles used to represent a 3-D image is dependent on the complexity of the surfaces as well as the desired resolution of the image and may be quite large, e.g., in the millions. Each triangle is defined by three vertices, and each vertex is associated with various attributes such as space coordinates, color values, and texture coordinates. Each attribute may have up to four components.

[0003] A graphics processor may perform various graphics operations to render an image. The graphics operations may include rasterization, stencil and depth tests, texture mapping, shading, etc. The image is composed of many triangles, and each triangle is composed of picture elements (pixels). The graphics processor renders each triangle by determining the values of the components of each pixel within the triangle.

[0004] A graphics processor may employ a shader core to perform certain graphics operations such as shading. Shading is a highly complex graphics operation involving lighting, shadowing, etc. The shader core may need to compute transcendental elementary functions such as sine, cosine, reciprocal, logarithm, exponential, square root, and reciprocal square root. These elementary functions may be approximated with polynomial expressions, which may be evaluated with relatively simple instructions executed by an arithmetic logic unit (ALU). However, shader performance may suffer greatly from computing the elementary functions in this manner using an ALU.

SUMMARY

[0005] Graphics processors capable of efficiently performing arithmetic operations and computing elementary functions are described herein. The terms "operation" and "function" are sometimes used interchangeably. A graphics processor comprises a shader core and possibly other units. The shader core has at least one ALU that can perform arithmetic operations and at least one elementary function unit that can compute elementary functions. In some embodiments, the ALU(s) and elementary function unit(s) are arranged and interconnected such that they can operate in parallel on instructions for the same or different threads to improve throughput. For example, the ALU(s) may execute one instruction for one thread, and the elementary function unit(s) may concurrently execute another instruction for another thread. These threads may be for the same or different graphics applications.

[0006] In other embodiments, the shader core has fewer elementary function units than ALUs, e.g., four ALUs and a single elementary function unit. The four ALUs may perform an arithmetic operation on (1) up to four components of an attribute for one pixel or (2) one component of an attribute for up to four pixels. The single elementary function unit may operate on one component of one pixel at a time. The use of a single elementary function unit may reduce cost (since elementary function units are more complex and costly than ALUs) while still providing good performance (since elementary functions have lower average usage than arithmetic operations).

[0007] Various aspects and embodiments of the invention are described in further detail below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The features and nature of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify correspondingly throughout.

[0009] FIG. 1 shows a graphics processor supporting graphics applications.

[0010] FIG. 2 illustrates attributes and components of a pixel.

[0011] FIG. 3A shows pixel-parallel processing with four scalar ALUs.

[0012] FIG. 3B shows component-parallel processing with one quad ALU.

[0013] FIG. 4 shows a shader core with a 4-unit ALU core and a 4-unit elementary function (EF) core.

[0014] FIG. 5 shows a shader core with parallel ALU core and EF core.

[0015] FIG. 6 shows a shader core with a 4-unit ALU core and a 1-unit EF core.

[0016] FIG. 7 shows a block diagram of a wireless device with a graphics processor.

DETAILED DESCRIPTION

[0017] The word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment or design described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments or designs.

[0018] **FIG. 1** shows a block diagram of a graphics system 100 that supports N graphics applications/programs 110a through 110n, where in general $N \geq 1$. Graphics system 100 may be a stand-alone system or part of a larger system such as a computing system, a wireless communication device, etc. Graphics applications 110a through 110n may be for video games, graphics, etc., and may run concurrently. Each graphics application 110 may generate threads to achieve the desired results. A thread (or thread of execution) indicates a specific task that may be performed with a sequence of one or more instructions. Threads allow a graphics application to have multiple tasks performed simultaneously by different units and further allow different graphics applications to share resources.

[0019] A graphics processor 120 receives the threads from graphics applications 110a through 110n and performs the tasks indicated by these threads. In the embodiment shown in FIG. 1, graphics processor 120 includes a shader core/processor 130, a texture engine 140, and a cache memory system 150. A core generally refers to a processing unit within an integrated circuit. The terms "core", "engine", "processor" and "processing unit" are often used interchangeably. Shader core 130 may perform certain graphics operations such as shading and may compute transcendental elementary functions. Texture engine 140 may perform other graphics operations such as texture mapping. Cache memory system 150 may include one or more caches, which are fast memories that can store data and instructions for shader core 130 and texture engine 140.

[0020] Graphics processor 120 may include other processing and control units, engines, and memories. For example, graphics processor 120 may include one or more additional engines that perform triangle setup, rasterization, stencil and depth tests,

attribute setup, pixel interpolation, etc. The various graphics operations described herein are known in the art. The additional engine(s) may be coupled between graphics applications 110 and shader core 130 or may be coupled to shader core 130. Graphics processor 120 may implement a software interface such as Open Graphics Library (OpenGL), Direct3D, etc. OpenGL is described in a document entitled "The OpenGL® Graphics System: A Specification," Version 2.0, dated October 22, 2004, which is publicly available.

[0021] A main memory 160 is a large, slower memory located further away (e.g., off-chip) from graphics processor 120. Main memory 160 stores data and instructions that may be loaded into the caches within cache memory system 150.

[0022] **FIG. 2** illustrates attributes and components of a pixel. As noted above, a 2-D or 3-D image may be composed of many triangles, and each triangle may be composed of pixels. Each pixel may have various attributes such as space coordinates, color values, texture coordinates, etc. Each attribute may have up to four components. For example, space coordinates may be given by three components for horizontal and vertical coordinates (x and y) and depth (z) or by four components x, y, z, and w, where w is a fourth term for homogeneous coordinates. Homogeneous coordinates are useful for certain graphics operations such as translation, scaling, rotation, etc. Color values are typically given by red (r), green (g), and blue (b). Texture coordinates are typically given by horizontal and vertical coordinates (u and v). A pixel may also be associated with other attributes.

[0023] In many cases, it is desirable to operate on groups of pixels in an image to be rendered. The group size may be selected based on various factors such as hardware requirements, performance, etc. A group size of 2×2 may provide a good tradeoff between the various factors. Processing on four pixels in a 2×2 grid may be performed in several manners.

[0024] **FIG. 3A** shows pixel-parallel processing on four pixels 1 through 4 with four identical scalar ALUs, ALU1 through ALU4, respectively. In this example, the four components of an attribute being operated on are denoted as $A_{p,1}$, $A_{p,2}$, $A_{p,3}$ and $A_{p,4}$, where p is a pixel index and $p \in \{1, 2, 3, 4\}$ for pixels 1 through 4. These components may be for space coordinates, color values, texture coordinates, etc. The four operands to be applied to the four components are denoted as $B_{p,1}$, $B_{p,2}$, $B_{p,3}$ and $B_{p,4}$, for $p \in \{1, 2, 3, 4\}$ and may be constants. In this example, the ALUs perform a

multiply and accumulate (MAC) operation. The four components of each pixel are thus multiplied with the four operands, and the four intermediate results are accumulated to generate a final result for that pixel.

[0025] For pixel-parallel processing in FIG. 3A, each scalar ALU operates on the four components of one pixel, and the four ALUs concurrently operate on the four pixels. ALU1 multiplies component $A_{1,1}$ with $B_{1,1}$ in the first clock period T_1, then multiplies component $A_{1,2}$ with $B_{1,2}$ and accumulates this result with the prior result in the second clock period T_2, then multiplies component $A_{1,3}$ with $B_{1,3}$ and accumulates this result with the prior result in the third clock period T_3, then multiplies component $A_{1,4}$ with $B_{1,4}$ and accumulates this result with the prior result in the fourth clock period T_4. ALU2 through ALU4 similarly operate on the components of pixels 2 through 4, respectively.

[0026] **FIG. 3B** shows component-parallel processing on four pixels with one quad ALU, which may also be called a vector-based ALU. For component-parallel processing, the quad ALU operates on all four components of one pixel at a time. Thus, the quad ALU multiplies components $A_{1,1}$, $A_{1,2}$, $A_{1,3}$ and $A_{1,4}$ with operands $B_{1,1}$, $B_{1,2}$, $B_{1,3}$ and $B_{1,4}$, respectively, and accumulates the four intermediate results to obtain the final result for the first pixel in the first clock period T_1. The quad ALU similarly operates on the components of the second, third and fourth pixels in clock periods T_2, T_3 and T_4, respectively.

[0027] FIGS. 3A and 3B show two schemes for performing quad processing on up to four components of an attribute for up to four pixels. Quad processing for arithmetic operations may be performed by a single quad ALU or four scalar ALUs. In the following description, ALUs are assumed to be scalar ALUs unless noted otherwise. Quad processing may substantially improve performance. Thus, shader core 130 may be designed with capability to perform quad processing.

[0028] **FIG. 4** shows a block diagram of an embodiment of a shader core/processor 130a with a 4-unit ALU core 440 and a 4-unit elementary function core 450. Shader core 130a may be used for shader core 130 in FIG. 1.

[0029] Within shader core 130a, a multiplexer (Mux) 410 receives threads from graphics applications 110a through 110n and provides these threads to a thread scheduler and context register 420. Thread scheduler 420 performs various functions to schedule and manage execution of threads. Thread scheduler 420 determines whether

to accept new threads, creates a register map table for each accepted thread, and allocates resources to the threads. The register map table indicates mapping between logical register address to physical register file address. For each thread, thread scheduler 420 determines whether resources required for that thread are ready, pushes the thread into a sleep queue if any resource (e.g., instruction, register file, or texture read) for the thread is not ready, and moves the thread from the sleep queue to an active queue when all of the resources are ready. Thread scheduler 420 interfaces with a load control unit 460 in order to synchronize the resources for the threads.

[0030] Thread scheduler 420 also manages execution of threads. Thread scheduler 420 fetches the instruction(s) for each thread from an instruction cache 422, decodes each instruction if necessary, and performs flow control for the thread. Thread scheduler 420 selects active threads for execution, checks for read/write port conflict among the selected threads and, if there is no conflict, sends instruction(s) for one thread into a processing core 430 and sends instruction(s) for another thread to load control unit 460. Thread scheduler 420 maintains a program/instruction counter for each thread and updates this counter as instructions are executed or program flow is altered. Thread scheduler 420 also issues requests to fetch for missing instructions and removes threads that are completed.

[0031] Instruction cache 422 stores instructions for the threads. These instructions indicate specific operations to be performed for each thread. Each operation may be an arithmetic operation, an elementary function, a memory access operation, etc. Instruction cache 422 may be loaded with instructions from cache memory system 150 and/or main memory 160, as needed, via load control unit 460

[0032] In the embodiment shown in FIG. 4, processing core 430 includes ALU core 440 and elementary function core 450. ALU core 440 performs arithmetic operations such as addition, subtraction, multiplication, multiply and accumulate, absolute, negation, comparison, saturation, etc. ALU core 440 may also perform logical operations such as AND, OR, XOR, etc. ALU core 440 may also perform format conversion, e.g., from integers to floating point numbers, and vice versa. In the embodiment shown in FIG. 4, ALU core 440 may be a single quad ALU or four scalar ALUs. ALU core 440 may perform pixel-parallel processing on one component of an attribute for up to four pixels, as shown in FIG. 3A. Alternatively, ALU core 440 may

perform component-parallel processing on up to four components of an attribute for a single pixel, as shown in FIG. 3B.

[0033] In the embodiment shown in FIG. 4, elementary function core 450 is composed of four elementary function units that can compute an elementary function for one component of an attribute for up to four pixels (pixel-parallel) or up to four components of an attribute for one pixel (component-parallel). Elementary function core 450 may compute transcendental elementary functions such as sine, cosine, reciprocal, logarithm, exponential, square root, reciprocal square root, etc, which are widely used in shader instructions. Elementary function core 450 may improve shader performance by computing the elementary functions in much less time than the time required to perform polynomial approximations of the elementary functions using simple instructions.

[0034] Load control unit 460 controls the flow of data and instructions for various units within shader core 130a. Load control unit 460 interfaces with cache memory system 150 and loads instruction cache 422, a constant buffer 432, and register file banks/output buffer 470 with data and instructions from cache memory system 150. Load control unit 460 also saves the data in output buffer 470 to cache memory system 150. Load control unit 460 also provides instructions to texture engine 140.

[0035] Constant buffer 432 stores constant values used by ALU core 440. Output buffer 470 stores temporary results as well as final results from ALU core 440 and elementary function core 450 for threads. A demultiplexer (Demux) 480 receives the final results for the executed threads from output buffer 470 and provides these results to the graphics applications.

[0036] In the embodiment shown in FIG. 4, processing core 430 includes both ALU core 440 and elementary function core 450. This embodiment allows ALU core 440 and elementary function core 450 to share buses that couple cores 440 and 450 to other units (e.g., thread scheduler 420 and output buffer 470) within shader core 130a.

[0037] Elementary function units are generally more complex than ALUs. Even with cost-effective implementations, elementary function units typically occupy much larger circuit area than ALUs and are thus more expensive than ALUs. To achieve high shader throughput for all shader instructions, the number of elementary function units may be selected to match the number of ALUs, which is four in the embodiment shown in FIG. 4. However, studies have shown that even though elementary functions are

widely used, the average usage of elementary functions is fairly lower than the average usage of ALU operations. The lower average usage results from elementary functions being called less often than arithmetic operations as well as fewer components being operated on by elementary functions than arithmetic operations. For example, elementary functions are typically called less often than arithmetic operations and may thus be adequately supported by fewer elementary function units. Furthermore, while it may be common to perform addition or multiplication on all four components of an attribute, which would then benefit from having four ALUs, it is less common to perform an elementary function on all four components. Hence, fewer elementary function units may be able to provide good performance in many cases in which elementary functions are performed on only a subset of the components, e.g., one or two components. Implementing fewer elementary function units may reduce cost while still providing good performance.

[0038] **FIG. 5** shows a block diagram of an embodiment of a shader core 130b with a 4-unit ALU core 540 and an L-unit elementary function core 550, where $1 \le L < 4$. Shader core 130b may also be used for shader core 130 in FIG. 1. Shader core 130b includes a multiplexer 510, a thread scheduler and context register 520, an instruction cache 522, a constant buffer 532, ALU core 540, elementary function core 550, a load control unit 560, register file banks/output buffer 570, and a demultiplexer 580 that operate in similar manner as units 410, 420, 422, 432, 440, 450, 460, 470 and 480, respectively, in FIG. 4.

[0039] ALU core 540 may be a single quad ALU or four scalar ALUs. ALU core 540 couples to thread scheduler 520, constant buffer 532, and output buffer 570 via one set of buses. Elementary function core 550 may be composed of one, two or three (L) elementary function units that can compute an elementary function for either L components of one pixel or one component of L pixels. Elementary function core 550 couples to thread scheduler 520, constant buffer 532, and output buffer 570 via another set of buses. In the embodiment shown in FIG. 5, ALU core 540 and elementary function core 550 are implemented separately from one another and are coupled to other units within shader core 130b via separate buses. ALU core 540 and elementary function core 550 may then operate on different instructions in parallel. These instructions may be for the same or different graphics applications.

[0040] In the embodiment shown in FIG. 5, the number of elementary function units is fewer than the number of ALUs and may be selected based on a tradeoff between cost and performance. In many cases, elementary function core 550 will be able to keep pace with ALU core 540 because of the lower average usage of elementary functions. Thread scheduler 520 appropriately schedules elementary function operations with knowledge that L (instead of four) elementary function units are available for use.

[0041] **FIG. 6** shows a block diagram of an embodiment of a shader core/processor 130c with a 4-unit ALU core 640 and a 1-unit elementary function core 650. Shader core 130c may also be used for shader core 130 in FIG. 1. Shader core 130c includes a multiplexer 610, a thread scheduler and context register 620, an instruction cache 622, a constant buffer 632, ALU core 640, elementary function core 650, a load control unit 660, register file banks/output buffer 670, and a demultiplexer 680 that operate in similar manner as units 410, 420, 422, 432, 440, 450, 460, 470 and 480, respectively, in FIG. 4.

[0042] ALU core 640 may be a single quad ALU or four scalar ALUs. ALU core 640 couples to thread scheduler 620, constant buffer 632, and output buffer 670 via a set of buses. Elementary function core 650 may be composed of a single elementary function unit that can compute an elementary function for one component of one pixel at a time. In the embodiment shown in FIG. 6, elementary function core 650 couples to load control unit 660 and output buffer 670. This embodiment reduces the number of buses to support separate ALU core 640 and elementary function core 650. This embodiment may also provide other benefits such as more efficient sharing of resources such as register file read/write port, instruction decode, etc.

[0043] Instructions for elementary functions (or EF instructions) may be generated in an appropriate manner given the design as well as the placement of elementary function core 650 within shader core 130c. If the number of EF units is equal to the number of ALU units (e.g., as shown in FIG. 4) and if the EF units have the same pipeline latency as the ALU units, then the EF instructions may be treated as ALU instructions with predictable pipeline delay. However, uneven implementations of ALU core 640 and elementary function core 650 may result in uneven throughput. Thus, in an embodiment, shader core 130c treats elementary function core 650 as a load resource and processes EF instructions in similar manner and with the same synchronization as, e.g., a texture load or a memory load. For example, a shader compiler may compile EF

instructions as instructions related to texture load instead of as ALU instructions, which may be the case in the embodiments shown in FIGS. 4 and 5.

[0044] The shader compiler may include synchronization (sync) bits in instructions as appropriate. A sync bit may indicate that the current instruction which contains the sync bit has data dependency with one or more previous instructions, which may have unpredictable delay or latency. The unpredictable latency may be due to several sources. First, unpredictable latency of texture load or memory load may result from unpredictable execution conditions such as cache hit/miss, memory access competence, memory access sequence, etc. Second, unpredictable latency may be caused by uneven implementations of the ALU core and the elementary function core. The shader compiler may insert sync bits in instructions that have data dependency with previous EF instructions, which may have unpredictable delay. These sync bits ensure that the instructions follow their dependent EF instructions and hence operate on the proper data.

[0045] In the embodiment shown in FIG. 6, thread scheduler 620 may generate elementary function requests that may share a bus with data load requests. This shared bus may comprise the bus from thread scheduler 620 to load control unit 660. However, elementary function core 650 may execute in parallel with load instructions in load control unit 660. In another embodiment, elementary function core 650 is coupled directly to thread scheduler 620 via a dedicated bus, e.g., as shown in FIG. 5. In this embodiment, elementary function requests and data load requests may use separate buses. In both embodiments, thread scheduler 620, ALU core 640, elementary function core 650, and load control unit 660 may operate in parallel on different threads for improved performance.

[0046] FIGS. 4 through 6 show specific embodiments of shader cores 130a, 130b and 130c. Other variations of shader cores 130a, 130b and 130c are also possible. For example, elementary function core 450 in FIG. 4 may include fewer than four elementary function units. As another example, elementary function core 650 in FIG. 6 may include more than one elementary function unit, e.g., two elementary function units.

[0047] In general, a shader core may include any number of processing, control and memory units, which may be arranged in any manner. These units may also be referred to by other names. For example, a load control unit may also be called an input/output

(I/O) interface unit. In some embodiments, a shader core may include fewer elementary function units than ALUs to reduce cost with little degradation in performance. In other embodiments, a shader core may include separate ALU core and elementary function core that can operate on different instructions for the same or different graphics applications in parallel. The ALUs and elementary function units may be implemented with various designs known in the art. A shader core may also interface with external units via synchronous and/or asynchronous interfaces.

[0048] The graphics processors and shader cores described herein may be used for wireless communication, computing, networking, personal electronics, etc. An exemplary use of a graphics processor for wireless communication is described below.

[0049] **FIG. 7** shows a block diagram of an embodiment of a wireless device 700 in a wireless communication system. Wireless device 700 may be a cellular phone, a terminal, a handset, a personal digital assistant (PDA), or some other device. The wireless communication system may be a Code Division Multiple Access (CDMA) system, a Global System for Mobile Communications (GSM) system, or some other system.

[0050] Wireless device 700 is capable of providing bi-directional communication via a receive path and a transmit path. On the receive path, signals transmitted by base stations are received by an antenna 712 and provided to a receiver (RCVR) 714. Receiver 714 conditions and digitizes the received signal and provides samples to a digital section 720 for further processing. On the transmit path, a transmitter (TMTR) 716 receives data to be transmitted from digital section 720, processes and conditions the data, and generates a modulated signal, which is transmitted via antenna 712 to the base stations.

[0051] Digital section 720 includes various processing and interface units such as, for example, a modem processor 722, a video processor 724, an application processor 726, a display processor 728, a controller/processor 730, a graphics processor 740, and an external bus interface (EBI) 760. Modem processor 722 performs processing for data transmission and reception (e.g., encoding, modulation, demodulation, and decoding). Video processor 724 performs processing on video content (e.g., still images, moving videos, and moving texts) for video applications such as camcorder, video playback, and video conferencing. Application processor 726 performs processing for various applications such as multi-way calls, web browsing, media

player, and user interface. Display processor 728 performs processing to facilitate the display of videos, graphics, and texts on a display unit 780. Controller/processor 730 may direct the operation of various processing and interface units within digital section 720.

[0052] Graphics processor 740 performs processing for graphics applications and may be implemented as described above. For example, graphics processor 740 may include shader core/processor 130 and texture engine 140 in FIG. 1. A cache memory system 750 stores data and/or instructions for graphics processor 740. Cache memory system 750 may be implemented with (1) configurable caches that may be assigned to different engines within graphics processor 740 and/or (2) dedicated caches that are assigned to specific engines. EBI 760 facilitates transfer of data between digital section 720 (e.g., the caches) and main memory 770.

[0053] Digital section 720 may be implemented with one or more digital signal processors (DSPs), micro-processors, reduced instruction set computers (RISCs), etc. Digital section 720 may also be fabricated on one or more application specific integrated circuits (ASICs) or some other type of integrated circuits (ICs).

[0054] The graphics processors and shader cores/processors described herein may be implemented in various hardware units. For example, the graphics systems and shader cores/processors may be implemented in ASICs, digital signal processors (DSPs), digital signal processing device (DSPDs), programmable logic devices (PLDs), field programmable gate array (FPGAs), processors, controllers, micro-controllers, microprocessors, and other electronic units.

[0055] Certain portions of the graphics processors may be implemented in firmware and/or software. For example, the thread scheduler and/or load control unit may be implemented with firmware and/or software modules (e.g., procedures, functions, and so on) that perform the functions described herein. The firmware and/or software codes may be stored in a memory (e.g., memory 750 or 770 in FIG. 7) and executed by a processor (e.g., processor 730). The memory may be implemented within the processor or external to the processor.

[0056] The previous description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the present invention. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without

departing from the spirit or scope of the invention. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

[0057] WHAT IS CLAIMED IS:

CLAIMS

1. An apparatus comprising:

at least one arithmetic logic unit (ALU) operative to perform arithmetic operations; and

at least one elementary function unit operative to compute elementary functions, wherein the at least one elementary function unit and the at least one ALU are operable on multiple threads in parallel.

2. The apparatus of claim 1 and comprising four ALUs.

3. The apparatus of claim 2, wherein the four ALUs are operable to perform an arithmetic operation on up to four components of an attribute for a pixel.

4. The apparatus of claim 2, wherein the four ALUs are operable to perform an arithmetic operation on a component of an attribute for up to four pixels.

5. The apparatus of claim 1 and comprising fewer elementary function units than ALUs.

6. The apparatus of claim 1 and comprising a single elementary function unit.

7. The apparatus of claim 1, wherein instructions for the at least one elementary function unit are compiled with synchronization as load instructions.

8. The apparatus of claim 1, wherein control bits are used for synchronization of instructions for the at least one elementary function unit with other instructions dependent on the instructions for the at least one elementary function unit.

9. The apparatus of claim 1, wherein the at least one ALU is operable to execute a first instruction for a first thread and the at least one elementary function unit

is operable to execute a second instruction for a second thread in parallel with the at least one ALU.

10. The apparatus of claim 1, wherein the at least one ALU and the at least one elementary function unit have different latency.

11. The apparatus of claim 1, further comprising:

a load control unit operative to facilitate exchanges of data between the at least one ALU and a memory system and between the at least one elementary function unit and the memory system.

12. The apparatus of claim 11, wherein the at least one elementary function unit is coupled to the load control unit.

13. The apparatus of claim 12, wherein the at least one elementary function unit is operable in parallel with the load control unit.

14. The apparatus of claim 12, wherein requests for the at least one elementary function unit and load requests for the load control unit share a bus, and wherein the at least one elementary function unit and the load control unit are operable to execute different threads in parallel.

15. The apparatus of claim 1, further comprising:

a scheduler operative to receive threads from at least one graphics application and to schedule execution of the threads by the at least one ALU and the at least one elementary function unit.

16. The apparatus of claim 15, wherein the at least one elementary function unit is coupled to the scheduler.

17. The apparatus of claim 1, further comprising:

an output buffer coupled to the at least one ALU and the at least one elementary function unit and operative to store results from the at least one ALU and the at least one elementary function unit.

18. An integrated circuit comprising:

at least one arithmetic logic unit (ALU) operative to perform arithmetic operations; and

at least one elementary function unit operative to compute elementary functions, wherein the at least one elementary function unit and the at least one ALU are operable on multiple threads in parallel.

19. The integrated circuit of claim 18 and comprising four ALUs and fewer than four elementary function units.

20. A wireless device comprising:

a graphics processor comprising at least one arithmetic logic unit (ALU) operative to perform arithmetic operations and at least one elementary function unit operative to compute elementary functions, wherein the at least one elementary function unit and the at least one ALU are operable on multiple threads in parallel; and

a memory system operative to store data for the graphics processor.

21. The wireless device of claim 20, wherein the graphics processor comprises four ALUs and fewer than four elementary function units.

22. An apparatus comprising:

at least one arithmetic logic unit (ALU) operative to perform arithmetic operations; and

at least one elementary function unit operative to compute elementary functions, wherein the number of the at least one elementary function unit is fewer than the number of the at least one ALU.

23. The apparatus of claim 22, wherein the at least one ALU is operable in parallel with the at least one elementary function unit.

24. The apparatus of claim 22, further comprising:

a load control unit operative to facilitate exchanges of data between the at least one ALU and a memory system and between the at least one elementary function unit and the memory system.

25. The apparatus of claim 24, wherein the at least one elementary function unit is coupled to the load control unit.

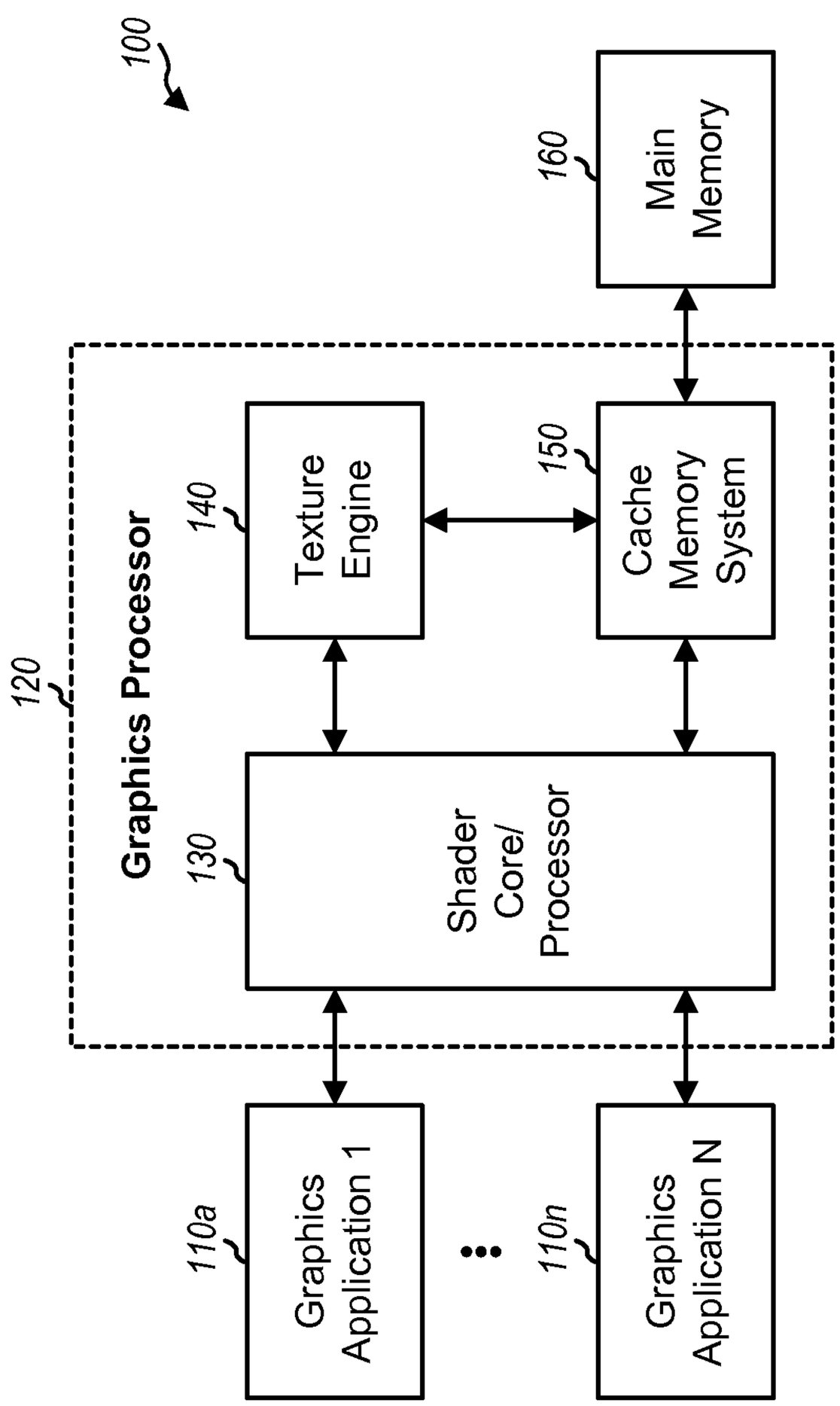

100
120
Graphics Processor
130
Shader Core/ Processor
140
Texture Engine
150
Cache Memory System
160
Main Memory
110a
Graphics Application 1
110n
Graphics Application N

FIG. 1

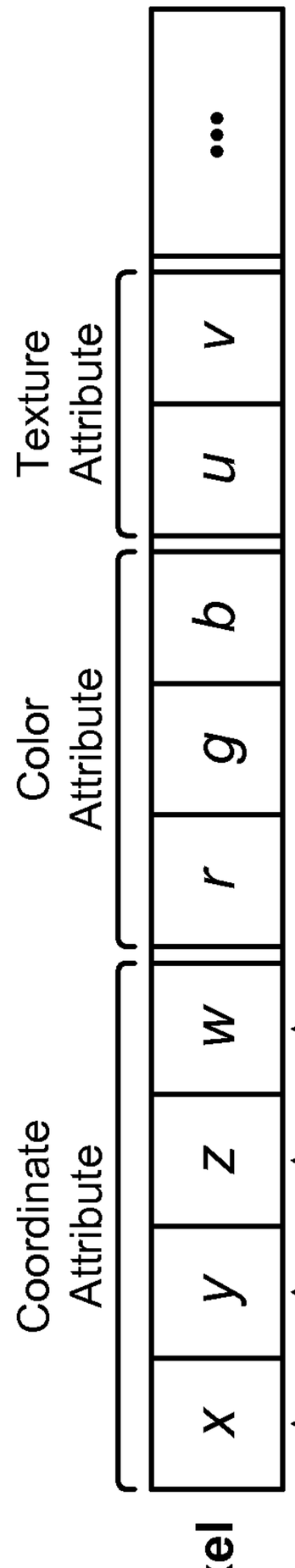

Texture Attribute
Color Attribute
u
v
b
g
r
Coordinate Attribute
x
y
z
w
Pixel
Components

FIG. 2

Clock Period	**Scalar ALU1** (Pixel 1)	**Scalar ALU2** (Pixel 2)	**Scalar ALU3** (Pixel 3)	**Scalar ALU4** (Pixel 4)
T_1	$A_{1,1} \bullet B_{1,1}$	$A_{2,1} \bullet B_{2,1}$	$A_{3,1} \bullet B_{3,1}$	$A_{4,1} \bullet B_{4,1}$
T_2	$+ A_{1,2} \bullet B_{1,2}$	$+ A_{2,2} \bullet B_{2,2}$	$+ A_{3,2} \bullet B_{3,2}$	$+ A_{4,2} \bullet B_{4,2}$
T_3	$+ A_{1,3} \bullet B_{1,3}$	$+ A_{2,3} \bullet B_{2,3}$	$+ A_{3,3} \bullet B_{3,3}$	$+ A_{4,3} \bullet B_{4,3}$
T_4	$+ A_{1,4} \bullet B_{1,4}$	$+ A_{2,4} \bullet B_{2,4}$	$+ A_{3,4} \bullet B_{3,4}$	$+ A_{4,4} \bullet B_{4,4}$

FIG. 3A

Clock Period	**Quad ALU** (Pixels 1 - 4)
T_1	$A_{1,1} \bullet B_{1,1} + A_{1,2} \bullet B_{1,2} + A_{1,3} \bullet B_{1,3} + A_{1,4} \bullet B_{1,4}$
T_2	$A_{2,1} \bullet B_{2,1} + A_{2,2} \bullet B_{2,2} + A_{2,3} \bullet B_{2,3} + A_{2,4} \bullet B_{2,4}$
T_3	$A_{3,1} \bullet B_{3,1} + A_{3,2} \bullet B_{3,2} + A_{3,3} \bullet B_{3,3} + A_{3,4} \bullet B_{3,4}$
T_4	$A_{4,1} \bullet B_{4,1} + A_{4,2} \bullet B_{4,2} + A_{4,3} \bullet B_{4,3} + A_{4,4} \bullet B_{4,4}$

FIG. 3B

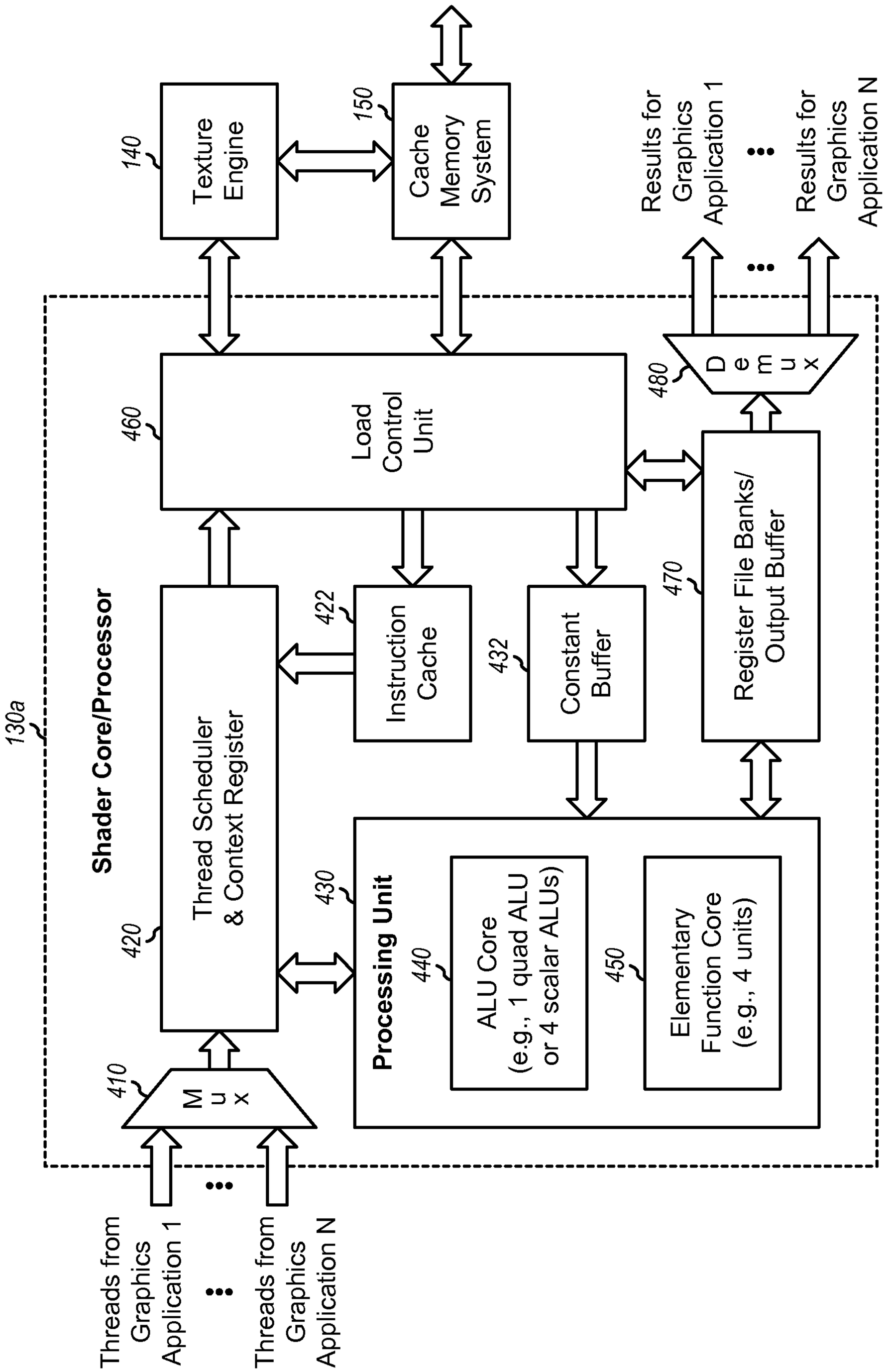

Texture Engine
140
Cache Memory System
150
Load Control Unit
460
Shader Core/Processor
130a
Thread Scheduler & Context Register
420
Instruction Cache
422
Constant Buffer
432
Register File Banks/ Output Buffer
470
Processing Unit
430
ALU Core (e.g., 1 quad ALU or 4 scalar ALUs)
440
Elementary Function Core (e.g., 4 units)
450
Mux
410
Demux
480
Threads from Graphics Application 1
Threads from Graphics Application N
Results for Graphics Application 1
Results for Graphics Application N

FIG. 4

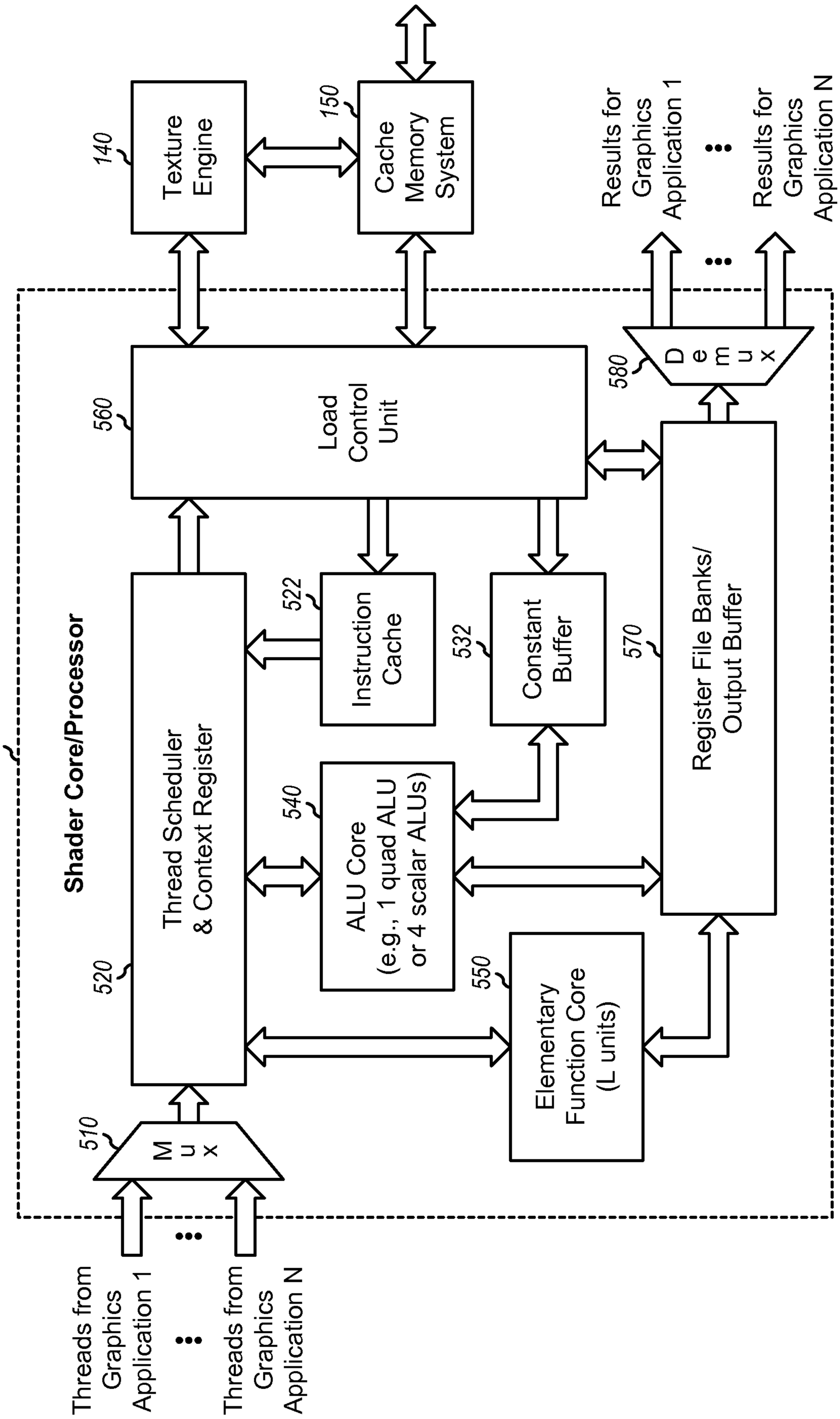

Texture Engine
140
Cache Memory System
150
Results for Graphics Application 1
Results for Graphics Application N
Load Control Unit
560
Demux
580
130b
Shader Core/Processor
Thread Scheduler & Context Register
520
Instruction Cache
522
Constant Buffer
532
Register File Banks/ Output Buffer
570
ALU Core (e.g., 1 quad ALU or 4 scalar ALUs)
540
Elementary Function Core (L units)
550
Mux
510
Threads from Graphics Application 1
Threads from Graphics Application N

FIG. 5

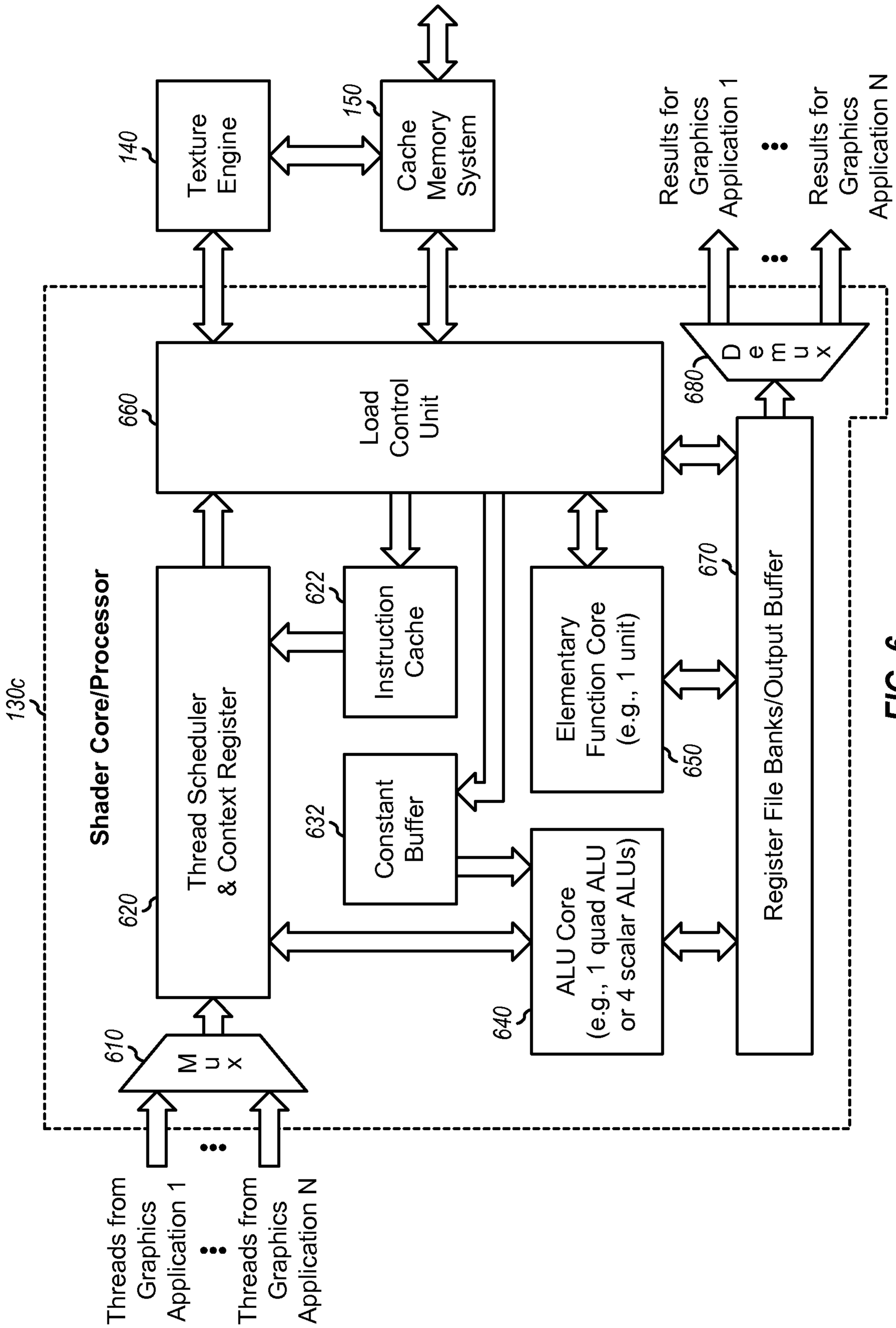

Texture Engine
140
Cache Memory System
150
Load Control Unit
660
Shader Core/Processor
130c
Thread Scheduler & Context Register
620
Instruction Cache
622
Constant Buffer
632
Mux
610
ALU Core (e.g., 1 quad ALU or 4 scalar ALUs)
640
Elementary Function Core (e.g., 1 unit)
650
Demux
680
Register File Banks/Output Buffer
670
Threads from Graphics Application 1
Threads from Graphics Application N
Results for Graphics Application 1
Results for Graphics Application N

FIG. 6

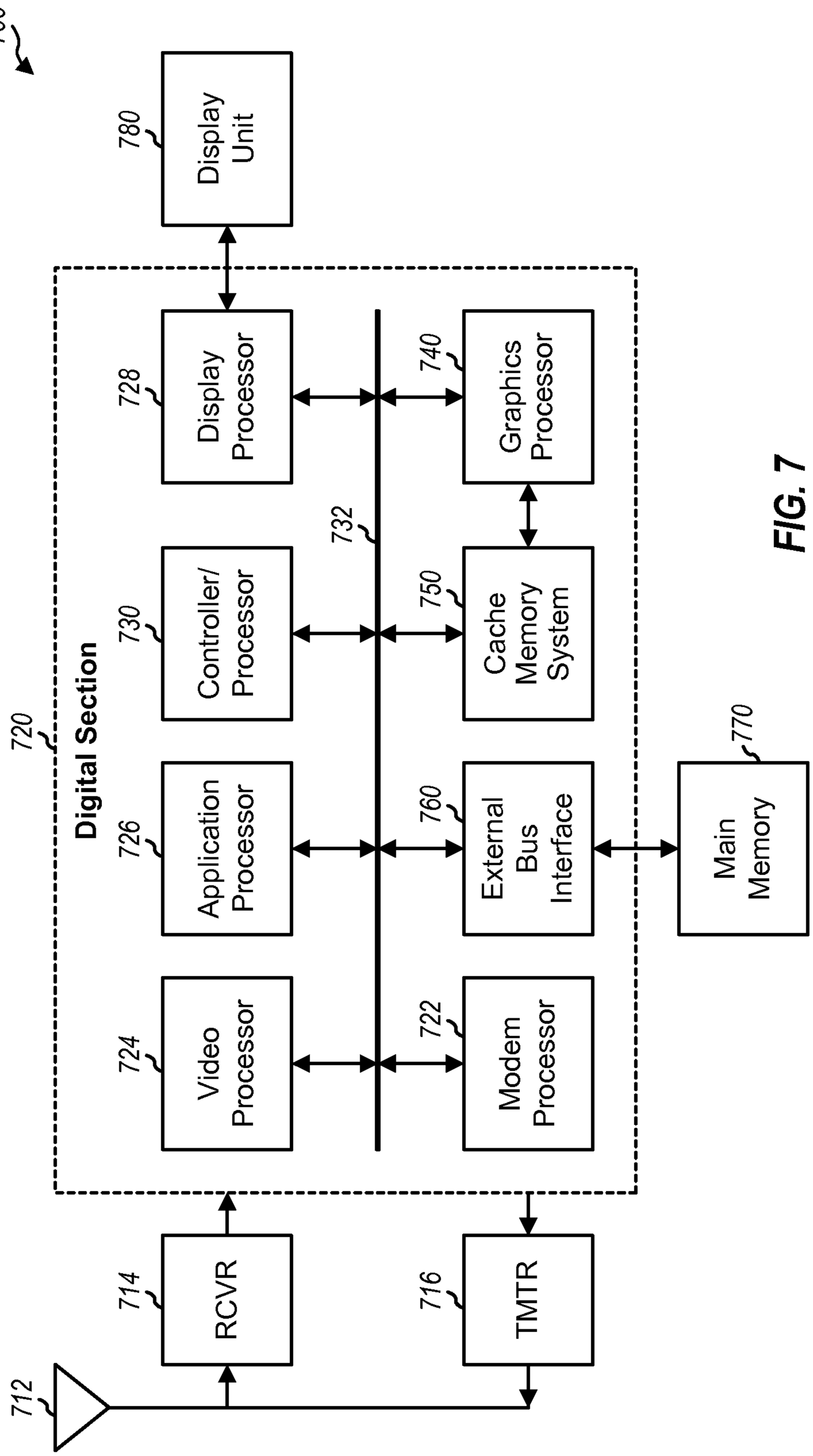
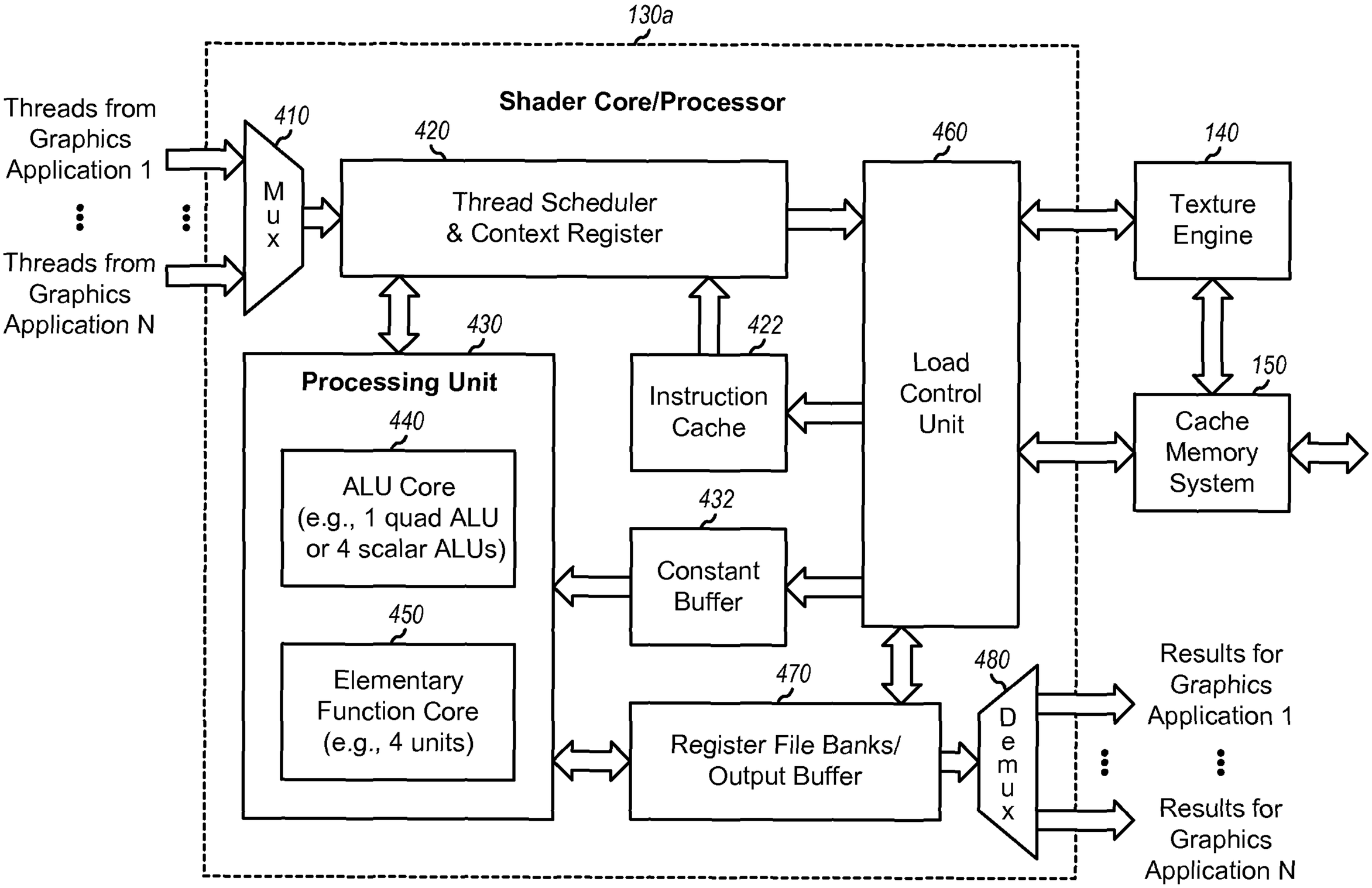

700
780
Display Unit
Digital Section
720
728
Display Processor
740
Graphics Processor
732
730
Controller/ Processor
750
Cache Memory System
726
Application Processor
760
External Bus Interface
770
Main Memory
724
Video Processor
722
Modem Processor
714
RCVR
716
TMTR
712

FIG. 7


130a
Shader Core/Processor
Threads from Graphics Application 1
Threads from Graphics Application N
410
Mux
420
Thread Scheduler & Context Register
460
140
Texture Engine
430
Processing Unit
440
ALU Core (e.g., 1 quad ALU or 4 scalar ALUs)
450
Elementary Function Core (e.g., 4 units)
422
Instruction Cache
Load Control Unit
150
Cache Memory System
432
Constant Buffer
470
Register File Banks/ Output Buffer
480
Demux
Results for Graphics Application 1
Results for Graphics Application N