GRAPHICS SYSTEM WITH CONFIGURABLE CACHES

BACKGROUND

I. Field

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

II. Background

[0002] Graphics systems 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. For example, space coordinates are typically given by horizontal and vertical coordinates (x and y) and depth (z), color values are typically given by red, green, and blue (r, g, b), and texture coordinates are typically given by horizontal and vertical coordinates (u and v).

[0003] A graphics processor in a graphics system may perform various graphics operations to render a 2-D or 3-D image. 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 component values of each pixel within the triangle. The graphics operations may include rasterization, stencil and depth tests, texture mapping, shading, etc. Since the graphics processor performs various graphics operations on pixel data, which may be any data associated with pixels, the ability to quickly access the pixel data has a large impact on performance.

SUMMARY

[0004] Graphics systems with configurable caches and having good performance are described herein. A graphics system includes a graphics processor and a cache memory system. The graphics processor includes processing units that perform various graphics operations to render graphics images. The cache memory system may include fully configurable caches, partially configurable caches, or a combination of configurable and dedicated caches. A cache is fully configurable if it can be assigned to any one of the processing units that can be assigned with caches. A cache is partially configurable if it can be assigned to any one of a subset of the processing units. A cache is dedicated if it is assigned to a specific processing unit. The caches are fast memories that store data (e.g., pixel data and/or instructions) for the processing units.

[0005] The cache memory system may further include a control unit, a crossbar, and an arbiter. The control unit may determine memory utilization by the processing units and assign the configurable caches to the processing units based on memory utilization. The configurable caches may be assigned to achieve good utilization of these caches and to avoid memory access bottleneck at any point within the graphics processor. The crossbar couples the processing units to their assigned caches. The arbiter facilitates data exchanges between the caches and a main memory.

[0006] Various aspects and embodiments of the invention are described in further detail below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] 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.

[0008] FIG. 1 shows a graphics system with dedicated caches.

[0009] FIG. 2 shows a graphics system with fully configurable caches.

[0010] FIG. 3 shows a crossbar in the graphics system in FIG. 2.

[0011] FIG. 4 shows a cache and a state machine for a processing unit.

[0012] FIG. 5 shows a graphics system with configurable and dedicated caches.

[0013] FIG. 6 shows a process to operate a graphics system with configurable caches.

[0014] FIG. 7 shows a wireless device in a wireless communication system.

DETAILED DESCRIPTION

[0015] 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.

[0016] **FIG. 1** shows a block diagram of a graphics system 100 with dedicated caches. Graphics system 100 includes a graphics processor 110 and a cache memory system 130. Graphics processor 110 includes various processing units arranged in a pipeline and used to render 2-D and 3D images. A command engine 120a receives and decodes incoming commands that specify the graphics operations to be performed. A triangle position and *z* setup engine 120b computes necessary parameters for the subsequent rasterization process. For example, engine 120b may compute the coefficients of linear equations for the three edges of each triangle, the coefficients for depth (*z*) gradient, etc. A rasterization engine 120c decomposes each triangle or line into pixels and generates a screen coordinate for each pixel.

[0017] A stencil and depth test engine 120d performs a stencil test on each pixel, if enabled, to determine whether the pixel should be displayed or discarded. A stencil buffer (not shown in FIG. 1) stores the current stencil value for each pixel location in the image being rendered. Engine 120d compares the stored stencil value for each pixel against a reference value and retains or discards the pixel based on the outcome of the comparison. Engine 120d also performs a depth test (which is also called a *z*-test) on each pixel, if applicable, to determine whether the pixel should be displayed or discarded. A *z*-buffer (not shown in FIG. 1) stores the current *z* value for each pixel location in the image being rendered. Engine 120d compares the *z* value of each pixel (the current *z* value) against the corresponding *z* value in the *z*-buffer (the stored *z* value), displays the pixel and updates the *z*-buffer and possibly the stencil buffer if the current *z* value is closer/nearer than the stored *z* value, and discards the pixel if the current *z* value is further back than the stored *z* value.

[0018] An attribute setup engine 120e computes necessary parameters for the subsequent interpolation of pixel attributes. For example, engine 120e may compute the coefficients of linear equations for attribute interpolation. A pixel interpolation engine 120f computes attribute component values for each pixel within each triangle based on the pixel's screen coordinate and using the information from engine 120e. A texture

mapping engine 120g performs texture mapping, if enabled, to apply texture to each triangle. A texture image may be stored in a texture buffer (not shown in FIG. 1). The three vertices of each triangle may be associated with three (u, v) coordinates in the texture image, and each pixel of the triangle may then be associated with specific texture coordinates in the texture image. Texturing is achieved by modifying the color of each pixel with the color of the texture image at the location indicated by that pixel's texture coordinates.

[0019] Each pixel is associated with information such as color, depth, texture, etc. A fragment is a pixel and its associated information. A fragment shader 120h applies software program, which may be composed of a sequence of instructions, to each fragment. Fragment shader 120h may also send texture requests to texture mapping engine 120g. A fragment engine 120i performs functions such as alpha test, fog blending, alpha blending, logic operation, and dithering operation on each fragment and provides results to a color buffer.

[0020] The graphics operations shown in FIG. 1 are known in the art. A graphics processor 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] In general, a graphics processor may include any number of processing units for any set of graphics operations. Each processing unit may be an engine that is implemented with dedicated hardware, a processor, or a combination of both. For example, the engines shown in FIG. 1 may be implemented with dedicated hardware whereas the fragment shader may be implemented with a programmable central processing unit (CPU). The processing units may be arranged in various orders depending on the desired optimization. For example, to conserve power, it may be desirable to perform stencil and depth tests early in the pipeline so that pixels that are not visible are discarded early, as shown in FIG. 1. The processing units may also be arranged in other orders, e.g., stencil and depth test engine 120d may be located after texture mapping engine 120g.

[0022] Cache memory system 130 includes dedicated caches 150a through 150f for some of the processing units 120 within graphics processor 110. In the design shown in FIG. 1, stencil cache 150a stores stencil values for pixels in a region of the image being

rendered, and depth cache 150b stores depth values for pixels in the same or different region. Stencil and depth test engine 120d accesses the stencil values stored in cache 150a and performs stencil tests using these stencil values. Engine 120d also accesses *z* values stored in cache 150b and performs depth tests using these *z* values. Texture cache 150c stores a texture mapping table that is used by texture mapping engine 120g to map texture to triangles. Shader instruction cache 150d stores instructions for fragment shader 120h, and shader data cache 150e stores data for fragment shader 120h. Color cache 150f stores color (e.g., red, green, and blue) values for pixels.

[0023] An arbiter 160 couples to and facilitates exchanges of data between caches 150a through 150f and a main memory 170. A cache miss is declared whenever a processing unit 120 accesses an associated cache 150 for data and the requested data is not stored in the cache. The cache miss results in the current content of the cache being stored back to main memory 170, if needed, and a new region of data being fetched from main memory 170 and loaded into the cache. The requested data may then be obtained from the cache and provided to the processing unit. Arbiter 160 allows one cache at a time to exchange data with main memory 170.

[0024] Cache memory system 130 in FIG. 1 improves memory access performance for graphics system 100. Caches 150 are small, fast memories located close to graphics processor 110. The fast cache memories improve processing speed because processing units 120 can retrieve data and store results faster. Main memory 170 is a large, slower memory located further away (e.g., off-chip) from graphics processor 110. Main memory 170 stores data and instructions that may be loaded into caches 150. Caches 150 reduce the number of fetches from main memory 170 and generally improve the performance of graphics system 100.

[0025] FIG. 1 shows the use of dedicated caches. Each cache 150 stores data or instructions for a specific processing unit 120. The utilization of each cache 150 is dependent on the graphics application being executed. For example, a 3-D graphics application may only perform *z* pass, which may turn on only the depth and/or stencil test. For this application, only stencil cache 150a and/or depth cache 150b may be utilized, and the remaining caches 150c through 150f may be idle. As another example, a simple shader application may not utilize shader instruction cache 150d or shader data cache 150e and these caches may be idle. As yet another example, a graphics application may disable texture mapping, in which case texture cache 150c may not be

needed. As seen by these examples, the use of dedicated caches may result in poor utilization of the caches by some graphics applications. The use of dedicated caches may also result in memory access bottleneck occurring in different places for different graphics applications. The bottleneck limits the performance of graphics system 100.

[0026] **FIG. 2** shows an embodiment of a graphics system 200 with fully configurable caches. Graphics system 200 includes a graphics processor 210 and a cache memory system 230. Graphics processor 210 includes processing units 220a through 220i that may operate as described above for processing units 120a through 120i, respectively, in FIG. 1. As shown in FIG. 2, only some of the processing units may utilize caches to store data and/or instructions. In particular, caches may be used to store stencil and/or depth values for stencil and depth test engine 220d, texture data for texture mapping engine 220g, instructions and/or data for fragment shader 220h, and color values for fragment engine 220i.

[0027] Cache memory system 230 includes N fully configurable caches 250a through 250n that may be assigned to store data and/or instructions for processing units 220 within graphics processor 210, where in general $N \geq 1$. The caches are fully configurable in that each cache may be assigned to any processing unit that can be assigned with caches. In an embodiment, each cache 250 may be assigned to only one processing unit 220, but a given processing unit 220 may be assigned with zero, one, or multiple caches 250. A control unit 232 assigns the available caches 250a through 250n to stencil and depth test engine 220d, texture mapping engine 220g, fragment shader 220h, and/or fragment engine 220i based on memory utilization by these processing units. A crossbar 240 couples processing units 220d, 220g, 220h, and 220i to their assigned caches 250a through 250n. Crossbar 240 may also be referred to as a cross-switch or some other terminology. An arbiter 260 couples caches 250a through 250n to a main memory 270 and facilitates data exchanges between the caches and the main memory. Arbiter 260 may also be referred to as a selector, a memory interface, or some other terminology.

[0028] In general, a cache memory system may include any number of caches that may be of any sizes. The number of caches and their sizes may be selected based on a tradeoff between memory access performance and circuit area. Larger cache sizes result in higher cache hit rates and greater likelihood of the requested data being stored in the caches. However, larger cache sizes also require more circuit area, which may increase

cost. In one specific embodiment, cache memory system 230 includes twelve caches 250, and each cache is four kilobytes. In other embodiments, fewer or additional caches as well as other cache sizes may be used for the cache memory system.

[0029] **FIG. 3** shows a block diagram of an embodiment of crossbar 240 in FIG. 2. In this embodiment, crossbar 240 includes K interface units 310a through 310k, where in general K may be any integer value. The interface units may also be referred to as master units or some other terminology. In the embodiment shown in FIG. 2, K is equal to six, and the six interface units are for stencil, depth, texture, shader instruction, shader data, and color. A given processing unit 220 may interact with one or multiple interface units 310, depending on the data requirements of the processing unit. Crossbar 240 further includes K multiplexers (Mux) 340a through 340k for the K interface units 310a through 310k, respectively, and N multiplexers 350a through 350n for the N caches 250a through 250n, respectively.

[0030] Each interface unit 310 couples to an associated processing unit 220 via a respective set of lines 302. Each interface unit 310 includes a state machine 320 and an input/output (I/O) unit 330. For each interface unit 310, I/O unit 330 receives address and/or output data from the associated processing unit 220 via lines 302 and provides the address/data to one input of each of N multiplexers 350a through 350n. I/O unit 330 also receives input data or instructions from an associated multiplexer 340 and provides the data/instructions to the associated processing unit 220 via lines 302.

[0031] Each multiplexer 350 receives address/data from all K I/O units 330a through 330k at K inputs. Each multiplexer 350 also receives a respective control Cn from control unit 232 and provides the address/data from one input, as determined by the control Cn, to the associated cache 250. Each multiplexer 350 thus couples one interface unit 310 to the associated cache 250.

[0032] Each multiplexer 340 receives input data or instructions from all N caches 250a through 250n at N inputs. Each multiplexer 340 also receives a respective control Pk from control unit 232 and provides the data/instructions from one input, as determined by the control Pk, to the associated I/O unit 330. Each multiplexer 340 thus couples one cache 250 to the associated interface unit 310.

[0033] Each cache 250 receives an address from the associated multiplexer 350 and determines whether the requested data/instruction at that address is stored in the cache. Each cache 250 provides a hit/miss (h/m) indicator that indicates a cache hit (e.g., logic

high) if the requested data/instruction is stored in the cache or a cache miss (e.g., logic low) if the requested data/instruction is not stored in the cache. Each cache 250 provides the requested data/instruction if there is a cache hit. Each cache 250 waits for a fetch from main memory 270 and then provides the requested data/instruction if there is a cache miss.

[0034] For each interface unit 310, state machine 320 receives the hit/miss indicators from all N caches 250a through 250n and a control Mk that indicates which caches, if any, have been assigned to that interface unit. State machine 320 determines whether the requested data/instruction is stored in any of the assigned caches based on the hit/miss indicators for the assigned caches. State machine 320 generates a fetch command if the requested data/instruction is not stored in any of the assigned caches.

[0035] Control unit 232 generates controls M1 through MK for state machines 320a through 320k, respectively, controls P1 through PK for multiplexers 340a through 340k, respectively, and controls C1 through CN for multiplexers 350a through 350n, respectively. These controls determine which caches, if any, are assigned to each interface unit 310.

[0036] FIG. 3 shows an embodiment of crossbar 240 for fully configurable caches. In general, full configurability of the caches may be achieved by using multiplexers with sufficient number of inputs, designing the state machines to evaluate any possible cache assignment, and generating the proper controls for the state machines and the multiplexers. Crossbar 240 may also be implemented with other designs that can map the processing units to the available caches.

[0037] **FIG. 4** shows a block diagram of an embodiment of cache 250n and state machine 320 for interface unit 310k in FIG. 3. Cache 250n includes a cache controller 410, a tag RAM 420, and a data RAM 430. Cache controller 410 performs address checking for each access of cache 250n and ensures that the correct data/instruction is being accessed. Tag RAM 420 keeps track of which data/instructions are currently stored in cache 250n. Data RAM 430 stores the data/instructions. Each line of data RAM 430 may store data for one or more pixels or one or more instructions depending on whether cache 250n is configured to store data or instructions.

[0038] Cache 250n may be implemented with a direct map cache structure or an M-way set associate cache structure. For the direct map cache structure, each line of a memory at the next level (e.g., main memory 270) may be mapped to only one line of

data RAM 430. Each line of data RAM 430 (or each cache line) is associated with a tag in tag RAM 420 that identifies the specific line of main memory 270 being stored in that cache line. As an example, if main memory 270 has 256k lines and data RAM 430 has 4k lines, then each cache line is associated with a specific 12-bit address, and 32 lines of main memory 270 are mapped to each cache line. The specific line of main memory 270 being stored in each cache line may then be identified by a 12-bit address for the cache line and a 5-bit tag that indicates which one of the 32 lines of main memory 270 is stored in the cache line. For the M-way set associate cache structure, each line of main memory 270 may be mapped to any one of M possible lines of data RAM 430. In general, the tag structure may be dependent on the main memory size, the cache size, the cache structure, the size of each cache line, and/or other factors.

[0039] Tag RAM 420 stores a tag for each line of data RAM 430 and is updated whenever any line in data RAM 430 is replaced. The tag for each cache line indicates the line of main memory 270 being stored in that cache line. Within control unit 410, an address parser 412 receives from multiplexer 350n an address for a memory access by a processing unit 220, parses the address to obtain a target tag and a cache line address, provides the cache line address to tag RAM 420, and provides the target tag to a compare logic 414. Tag RAM 420 provides the stored tag for the cache line address from parser 412. Compare logic 414 compares the stored tag against the target tag, indicates a cache hit if the tags are equal, and indicates a cache miss if the tags are different. Tag RAM 420 may be accessed multiple times if cache 250n implements the M-way set associate cache structure.

[0040] Memory access unit 416 handles access of data RAM 430 when there is a cache hit. If the memory access is a read, then data RAM 430 is accessed to read the data/instruction stored at the cache line address. If the memory access is a write, then data RAM 430 is written at the cache line address and a cache "dirty bit" is updated to indicate that the cache line has been written to, so that the line will be written back to main memory 270 prior to being replaced.

[0041] State machine 320 for interface unit 310k determines whether the requested data/instruction is stored in any cache assigned to that interface unit. Within state machine 320, a mapper 450 receives the control MK from control unit 232 and generates N enable signals for the N caches 250a through 250n. Each enable signal is set to logic high if the associated cache 250 is assigned to interface unit 310k. N AND

gates 452a through 452n receive the N enable signals and N hit/miss indicators from N caches 250a through 250n, respectively. Each AND gate 452 provides either (1) a logic low if its enable signal is at logic low, which indicates that the associated cache 250 is not assigned to interface unit 310k, or (2) the hit/miss indicator from the associated cache 250 if the enable signal is at logic high, which indicates that the cache is assigned to interface unit 310k. An OR gate 454 receives the outputs of AND gates 452a through 452n and generates a hit indicator for interface unit 310k.

[0042] A cache fill unit 460 directs a cache fill of one of the caches assigned to interface unit 310k when there is a cache miss. In an embodiment, an assigned cache that is least recently used (LRU), i.e., unused for the longest time, is selected for the cache fill. Cache fill unit 460 may fill all or a portion of the selected cache with new data/instructions from main memory 270. Interface unit 310k may be stalled while the target cache line is filled from main memory 270 and the tag RAM of the selected cache is updated to reflect the new data/instructions loaded into the data RAM.

[0043] A graphics system with fully configurable caches provides the most flexibility in terms of assigning caches to processing units. A given processing unit may be assigned with zero, one, multiple, or even all of the available caches, depending on data requirements of the processing units. The available caches may be assigned to the processing units in a manner to achieve good utilization of the caches and to avoid memory access bottleneck at any one place in the graphics processor regardless of the graphics application being executed. The available caches may be intelligently assigned to processing units as described below.

[0044] In many instances, good performance may be achieved with partially configurable caches. The caches are partially configurable in that each cache may be assigned to one of a subset of processing units. Studies have been performed on various graphics applications to ascertain the benefits of using caches for different graphics operations. These studies indicate that caches are very useful in boosting performance of stencil and depth tests and texture mapping for many graphics applications and are only somewhat beneficial for color and shader. Hence, a cache memory system may be designed with many of the available caches being configurable for stencil and depth tests and texture mapping and only few of the caches being configurable for color and shader. Employing partially configurable caches may simplify the designs of the

control unit that assigns the configurable caches as well as the crossbar that couples the processing units to their assigned caches.

[0045] **FIG. 5** shows an embodiment of a graphics system 500 with configurable and dedicated caches. Graphics system 500 includes a graphics processor 510 and a cache memory system 530. Graphics processor 510 may include processing units that operate as described above for processing units 120a through 120i in FIG. 1. In particular, graphics processor 510 may include a stencil and depth test engine, a texture mapping engine, a fragment shader, and a fragment engine that may utilize data and/or instructions stored in caches.

[0046] In the embodiment shown in FIG. 5, cache memory system 530 includes R partially configurable caches 550a through 550r and one dedicated cache 550s, where in general $R \geq 1$. Table 1 shows possible assignments of each of caches 550a through 550s, where each "X" indicates a valid cache assignment. As shown in Table 1, each of the P partially configurable caches 550a through 550p may be assigned for stencil test, depth test, or texture mapping. Partially configurable cache 550q may be assigned for shader instruction, stencil test, depth test, or texture mapping. Partially configurable cache 550r may be assigned for shader data, stencil test, depth test, or texture mapping. Dedicated cache 550s is used to store color values for the fragment engine. In an embodiment, $P = 8$, and up to 10 caches may be assigned for stencil test, depth test, and/or texture mapping, assuming that caches 550q and 550r are not used for shader instructions and data. Fewer or additional caches may also be used for cache memory system 530.

Table 1

Processing Unit	Cache 550a	…	Cache 550p	Cache 550q	Cache 550r	Cache 550s
Stencil test	X	…	X	X	X	
Depth test	X	…	X	X	X	
Texture mapping	X	…	X	X	X	
Shader instruction				X		
Shader data					X	
Color						X

[0047] A control unit 532 assigns the configurable caches 550a through 550r to the stencil and depth test engine, the texture mapping engine, and/or the fragment shader

based on memory utilization by these processing units. A crossbar 540 couples the processing units to their assigned caches 550a through 550r. An arbiter 560 couples caches 550a through 550s to a main memory 570.

[0048] FIG. 5 also shows an embodiment of crossbar 540. In this embodiment, crossbar 540 includes interface units 580a through 580e for stencil test, depth test, texture mapping, shader instruction, and shader data, respectively. Interface unit 580a couples the stencil and depth test engine to caches assigned for stencil test, if any, among caches 550a through 550r. Interface unit 580b couples the stencil and depth test engine to caches assigned for depth test, if any, among caches 550a through 550r. Interface unit 580c couples the texture mapping engine to assigned caches, if any, among caches 550a through 550r. Interface unit 580d couples the fragment shader to cache 550q, if assigned. Interface unit 580e couples the fragment shader to cache 550r, if assigned. Each interface unit 580 may include a state machine and an I/O unit, e.g., as shown in FIG. 3. For simplicity, the multiplexers for the interface units and the multiplexers for the caches are not shown in FIG. 5.

[0049] FIG. 5 shows a specific embodiment of a cache memory system with configurable and dedicated caches. In general, a cache memory system may include any number of configurable caches and any number of dedicated caches. The configurable caches may be used for any set of graphics operations, and the dedicated caches may also be used for any set of graphics operations. The use of both configurable and dedicated caches may simplify the designs of the control unit and the crossbar.

[0050] In the embodiments shown in FIGS. 2 and 5, the control unit may assign the configurable caches to the processing units in various manners. In one embodiment, the control unit assigns caches for each graphics image or frame to be rendered based on memory utilization in a prior image/frame. The control unit may ascertain memory utilization by counting the number of memory accesses made by each processing unit, the number of cache hits for each cache, the number of cache misses for each cache, etc. The control unit may then assign more caches to processing units with high memory utilization and fewer or no caches to processing units with low memory utilization.

[0051] In another embodiment, the control unit assigns the configurable caches to the processing units based on coherency of graphics images/frames. Coherency refers to the amount of changes in consecutive 2D/3D frames. Fewer caches may be assigned

for higher coherency when frames are more likely to render similar contents, and more caches may be assigned for less coherency.

[0052] In yet another embodiment, the control unit assigns the configurable caches to the processing units based on characteristics of the graphics application being executed. For example, if the graphics system is executing a 2-D graphics application, then depth test may not be needed, and no cache may be assigned for depth test. On the other extreme, if a 3-D graphics application uses only *z* pass, then all configurable caches may be assigned to depth test. As another example, if a simple shader program is being executed and all of the shader instructions can be stored within the fragment shader, then no cache may be assigned to the fragment shader.

[0053] In yet another embodiment, the control unit dynamically assigns the configurable caches. For example, the control unit may assign one or more caches to a processing unit when data requests are sent by the processing unit. The control unit may adjust the number of caches assigned to the processing unit based on the number of requests, cache hit/miss statistics, and/or other factors, which may be determined on the fly. Flush and invalidate operations may be performed on the fly for a cache that is switched or re-assigned from one processing unit to another processing unit.

[0054] In yet another embodiment, the control unit assigns caches using a combination of static and dynamic assignments. For example, the control unit may pre-assign one or more caches to a given processing unit at the beginning of rendering a frame, image, or batch, e.g., based on current statistics on memory utilization by the processing unit. The control unit may adjust the cache assignment to this processing unit during the rendering of the frame, image, or batch, e.g., periodically. The new cache assignment for each rendering period may be determined based on the statistics obtained for the prior rendering period.

[0055] The control unit may also assign the configurable caches based on other criteria. The control unit may assign caches in each image/frame, whenever changes in memory utilization are detected, when a graphics application is first executed, and/or at other times.

[0056] For simplicity, FIGS. 1 through 5 show the cache memory systems including a bank of caches. A cache may be implemented with a block of memory. A cache or a cache memory system may also be implemented with a hierarchical structure having multiple levels, e.g., level 1, level 2, level 3, etc. The caches in a lower level (e.g., level

1) tend to be faster but smaller than the caches in a higher level (e.g., level 2). The caches in each level may be filled by the caches in the next higher level whenever cache misses occur. The number of levels, the number of caches in each level, and the cache sizes for the different levels may be fixed or configurable. For example, the number of levels, the number of caches in each level, and/or the cache sizes may be selected to achieve good performance and may be configurable based on the characteristics of the graphics applications.

[0057] **FIG. 6** shows an embodiment of a process 600 for operating a graphics system with configurable caches. Memory utilization by a plurality of processing units configured to perform graphics operations to render graphics images is determined (block 612). Memory utilization may be determined by monitoring memory accesses made by the processing units, by ascertaining the characteristics of the graphics application being executed, and/or in other manners. A plurality of caches are assigned to at least one processing unit among the plurality of processing units based on memory utilization (block 614). Each processing unit may be assigned with zero, one, multiple, or all of the caches depending on (1) memory utilization by that processing unit as well as the other processing units and (2) the caches available for assignment to that processing unit. Memory utilization may be ascertained based on various statistics such as, e.g., data requests by the processing unit, cache hit/miss statistics, etc. The cache assignment may be performed periodically (e.g., every graphics image/frame), whenever a change in memory utilization is detected, etc. The caches may also be re-assign during rendering of an image/frame based on detected changes in memory utilization. Each of the at least one processing unit is coupled to a respective set of caches assigned to the processing unit, e.g., via a respective interface unit (block 616).

[0058] The graphics systems and configurable caches described herein may be used for wireless communication, computing, networking, personal electronics, etc. An exemplary use of a graphics system with configurable caches for wireless communication is described below.

[0059] **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.

[0060] 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.

[0061] 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.

[0062] Graphics processor 740 performs processing for graphics applications and may be implemented as described above. A cache memory system 750 stores data and/or instructions for graphics processor 740 and may be implemented with configurable caches and possibly dedicated caches. Cache memory system 750 may further include a crossbar that couples the configurable caches to the processing units within graphics processor 740 and an arbiter that couples the caches to a main memory 770 via a bus 732 and EBI 760. EBI 760 facilitates transfer of data between digital section 720 (e.g., the caches) and main memory 770.

[0063] 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).

[0064] The graphics systems and configurable caches described herein may be implemented in various hardware units. For example, the graphics systems and configurable caches may be implemented in ASICs, digital signal processing device (DSPDs), programmable logic devices (PLDs), field programmable gate array (FPGAs), processors, controllers, micro-controllers, microprocessors, and other electronic units.

[0065] Certain portions of the graphics systems may be implemented in firmware and/or software. For example, the 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 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.

[0066] 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.

[0067] WHAT IS CLAIMED IS:

CLAIMS

1. An apparatus comprising:

a plurality of processing units configured to perform graphics operations to render graphics images;

a plurality of caches configured to store data for at least one processing unit among the plurality of processing units; and

a crossbar configured to couple the plurality of caches to the at least one processing unit.

2. The apparatus of claim 1, wherein each of the plurality of caches is assignable to any one of the plurality of processing units.

3. The apparatus of claim 1, wherein each of the plurality of caches is assignable to any one of a respective subset of the plurality of processing units.

4. The apparatus of claim 1, wherein the plurality of caches comprise one or more dedicated caches assigned to one or more processing units and at least one configurable cache assignable to remaining processing units.

5. The apparatus of claim 4, wherein each configurable cache is assignable to any one of a respective subset of the remaining processing units.

6. The apparatus of claim 4, wherein the remaining processing units comprise a depth test engine and a texture mapping engine.

7. The apparatus of claim 1, further comprising:

a control unit configured to ascertain memory utilization by the plurality of processing units and to assign the plurality of caches to the at least one processing unit based on memory utilization.

8. The apparatus of claim 7, wherein the control unit is configured to assign the plurality of caches for each graphics image to be rendered based on memory utilization for a prior graphics image.

9. The apparatus of claim 7, wherein the control unit is configured to ascertain memory utilization based on data requests by the processing units, cache hit/miss statistics, or a combination thereof.

10. The apparatus of claim 7, wherein the control unit is configured to detect changes in memory utilization by the plurality of processing units during rendering of an image and to re-assign the plurality of caches based on the detected changes in memory utilization.

11. The apparatus of claim 1, further comprising:

a control unit configured to assign the plurality of caches to the at least one processing unit based on memory utilization by a graphics application being executed.

12. The apparatus of claim 1, wherein the crossbar comprises

a plurality of interface units, each interface unit configured to couple an associated processing unit to a set of caches assigned to the processing unit.

13. The apparatus of claim 12, wherein each interface unit comprises

a state machine configured to determine whether data requested by the associated processing unit is stored in any one of the set of caches assigned to the processing unit.

14. The apparatus of claim 13, wherein the state machine for each interface unit receives cache hit/miss indicators from the plurality of caches and a control indicating the set of caches assigned to the associated processing unit.

15. The apparatus of claim 13, wherein the state machine for each interface unit is configured to fill one of the set of caches assigned to the associated processing unit when a cache miss occurs.

16. The apparatus of claim 1, wherein the plurality of caches are arranged in a hierarchical structure with at least two levels of caches.

17. The apparatus of claim 16, wherein at least one level in the hierarchical structure has a configurable number of caches.

18. The apparatus of claim 16, wherein at least one level in the hierarchical structure has configurable cache sizes.

19. The apparatus of claim 1, wherein the plurality of caches are arranged in a configurable number of levels in a hierarchical structure.

20. The apparatus of claim 1, wherein the plurality of caches have configurable cache sizes.

21. The apparatus of claim 1, further comprising:

an arbiter coupled to the plurality of caches and configured to facilitate data exchanges between the plurality of caches and a main memory.

22. The apparatus of claim 1, wherein the plurality of processing units comprise a depth test engine and a texture mapping engine.

23. The apparatus of claim 22, wherein the plurality of processing units are arranged in a pipeline, and wherein the depth test engine is located earlier in the pipeline than the texture mapping engine.

24. An integrated circuit comprising:

a plurality of processing units configured to perform graphics operations to render graphics images;

a plurality of caches configured to store data for at least one processing unit among the plurality of processing units; and

a crossbar configured to couple the plurality of caches to the at least one processing unit.

25. The integrated circuit of claim 24, wherein each of the plurality of caches is assignable to any one of a respective subset of the plurality of processing units.

26. The integrated circuit of claim 24, further comprising:

a control unit configured to ascertain memory utilization by the plurality of processing units and to assign the plurality of caches to the at least one processing unit based on memory utilization.

27. A wireless device comprising:

a graphics processor comprising a plurality of processing units configured to perform graphics operations to render graphics images; and

a cache memory system comprising

a plurality of caches configured to store data for at least one processing unit among the plurality of processing units, and

a crossbar configured to couple the plurality of caches to the at least one processing unit.

28. The wireless device of claim 27, wherein the cache memory system further comprises

an arbiter coupled to the plurality of caches and configured to facilitate data exchanges between the plurality of caches and a main memory.

29. A method comprising:

determining memory utilization by a plurality of processing units configured to perform graphics operations to render graphics images;

assigning a plurality of caches to at least one processing unit among the plurality of processing units based on memory utilization by the plurality of processing units; and

coupling each of the at least one processing unit to a respective set of caches assigned to the processing unit.

30. The method of claim 29, further comprising:

coupling one or more caches directly to one or more processing units among the plurality of processing units.

31. The method of claim 29, wherein the assigning the plurality of caches comprises

assigning the plurality of caches to the at least one processing unit for each graphics image to be rendered based on memory utilization for a prior graphics image.

32. An apparatus comprising:

means for determining memory utilization by a plurality of processing units configured to perform graphics operations to render graphics images;

means for assigning a plurality of caches to at least one processing unit among the plurality of processing units based on memory utilization by the plurality of processing units; and

means for coupling each of the at least one processing unit to a respective set of caches assigned to the processing unit.

33. The apparatus of claim 32, further comprising:

means for coupling one or more caches directly to one or more processing units among the plurality of processing units.

34. The apparatus of claim 32, wherein the means for assigning the plurality of caches comprises

means for assigning the plurality of caches to the at least one processing unit for each graphics image to be rendered based on memory utilization for a prior graphics image.