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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 September 2020 (24.09.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/186631 A1

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

(21) International Application Number:
PCT/CN2019/090181

(22) International Filing Date:
05 June 2019 (05.06.2019)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/821,650 21 March 2019 (21.03.2019) US

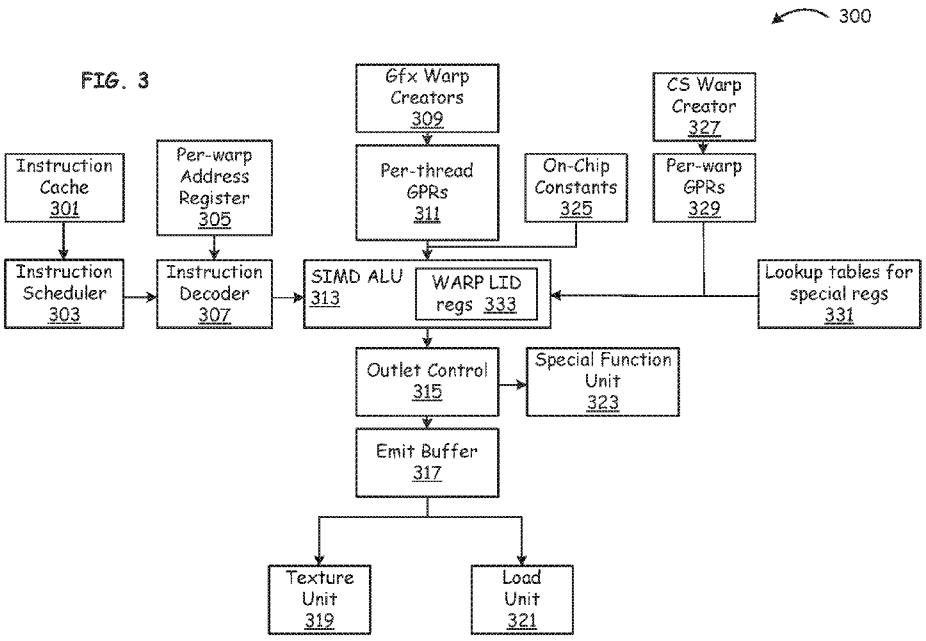
(71) Applicant: **HUAWEI TECHNOLOGIES CO., LTD.** [CN/CN]; Huawei Administration Building, Bantian, Longgang District, Shenzhen, Guangdong 518129 (CN).

(72) Inventors: **CHEN, Lin**; 13385 Winstanley Way, San Diego, California 92130 (US). **HONG, Zhou**; 11512 Country Spring Court, Cupertino, California 95014 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ,

(54) **Title:** COMPUTE SHADER WARPS WITHOUT RAMP UP



(57) **Abstract:** Data items to be processed in parallel are arranged as threads, within one or more warps. Information mapping thread identifiers to warp-specific local identifiers is stored, e.g. in a lookup table. Warp-specific local identifiers indicate positions of threads relative to warp origins. Warp origins are stored in one or more per-warp registers. The warp origins can be defined relative to a global address space or a workgroup address space. The warp-specific local identifiers are read from the table and stored in one or more warp local ID registers. The processor generates full identifiers on the fly, rather than relying on full identifiers prepopulated into per-thread registers, by combining the warp-specific local identifiers with the warp origins. The processor then uses the full identifiers to access the data items to be processed.

WO 2020/186631 A1

[Continued on next page]

TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

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

Published:

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

COMPUTE SHADER WARPS WITHOUT RAMP UP

BACKGROUND

CROSS REFERENCE TO RELATED APPLICATIONS

[1] This application claims priority to U.S. provisional patent application Serial No. 62/821,650, filed on March 21, 2019 and entitled "Compute Shader Warps Without Ramp Up", which is incorporated herein by reference as if reproduced in its entirety.

TECHNICAL FIELD

[2] This disclosure relates generally to single instruction multiple data (SIMD) processing systems, and more specifically to creating and starting compute shader (CS) warps.

DESCRIPTION OF RELATED ART

[3] Single instruction multiple data (SIMD) processing systems can concurrently apply the same instruction to multiple data items, referred to as threads. These threads are often organized into groups of data, called warps. Thus, for example, a SIMD32 can perform the same operation on 32 threads concurrently.

[4] In order to create or start a compute shader (CS) warp, conventional techniques initialize per-thread general purpose registers (GPRs) with local IDs associated with those threads. Prior art Figure 1, for example, illustrates a system 100 in which each thread 0-*n* of a warp 110 is associated with its own GPRs. For example Thread 0 is associated with GPRs 113, Thread1 is associated with GPRs 115, and Thread *n* is associated with GPRs 117. Each of these local IDs is multi-dimensional, including LocalID.x 105, LocalID.y 107, and LocalID.z 109. In this example, a Local ID is a 3D input variable inside a warp, a Global ID is a 3D input variable inside the dispatch, and a local index is the 1D representation of the local ID.

[5] The local ID can be used in a compute shader (CS) as follows:

int thread_id = int(gl_LocalInvocationID.x).

[6] The global ID and local index can be derived from the local ID. Therefore, in conventional techniques, the per-thread GPRs are initialized with the local ID during a warp

ramp up process before starting a warp. Thus, as shown in prior art Figure 1, for each thread, CS warp creator 127 pushes LocalID.x 105, LocalID.y 107, and LocalID.z 109 into a corresponding per-thread GPRs before executing or creating a warp. This conventional warp ramp up process requires multiple cycles to complete. These cycles are undesirable overhead required by conventional techniques to ramp up a warp.

SUMMARY

[7] One problem with the prior art is that data (local ID) needs to be pushed into multiple GPRs before executing a warp. It takes multiple cycles to push the local IDs into the register file. An objective of the invention is to reduce and/or eliminate compute shader (CS) warp ramp up.

[8] In various embodiments, a processing system includes means for concurrently performing a same processing operation on a plurality of data elements. The data elements can be arranged as threads within one or more warps. The warps can be arranged into workgroups, and the workgroups into a global address space such a domain, a dispatch, or the like. forming a workgroup. Other arrangements of warps into various different types of address spaces can be used consistent with the teachings set forth herein, as long as a warp origin can be established. The processing operation can be performed using a parallel computing processor such as a single instruction multiple data (SIMD) processor, for example a SIMD32 or SIMD64, processor, which supports data level parallelism, various types of array processors, including systolic arrays, multiple instruction single data (MISD) systems that can benefit from dynamic determination of addresses associated with a thread of instructions.

[9] One or more embodiments of the above processing system can include means for mapping thread identifiers to warp-specific local identifiers, where the warp-specific local identifiers indicating positions of the threads within the one or more warps relative to warp origins. These warp origins can be defined in a workgroup space, or in a global space. A table, list, or other suitable data construct can be used to store the information mapping the thread identifiers to the warp-specific local identifiers. These tables can be read into special registers, sometimes referred to herein as Local ID (LID or LID*n*) registers accessible to the processor performing the operations on the threads.

[10] Various embodiments also include means for storing warp origins in one or more per-warp registers. In some implementations using a workgroup address space, a warp origin value indicating a starting position of the warp within a workgroup is stored in one or more per-warp registers. In some implementations employing a global address space, global coordinates of the warp's origin is stored in multiple per-warp registers.

[11] In various embodiments, a computer shader (CS) warp creator determines the warp origins at the time a workgroup is instantiated, and stores the warp origins in the per-warp general purpose registers (wGPRs). A warp origin refers to a starting location of a warp within a workgroup, or within a global address space, depending on the particular implementation. The workgroup is made up of one or more warps, and each warp is made up of one or more threads. Note that storing warp origins in wGPRs uses fewer clock cycles than storing full thread identifiers for each thread in multiple per-thread GPRs associated the individual threads.

[12] The processor obtains the warp local ID from special warp LID registers, retrieves the warp origin from the per-warp general purpose registers, combines the warp-specific local identifiers with the warp origins using, for example, the arithmetic logical unit (ALU), to generate final, sometimes referred to as "full," identifiers that indicating locations of the individual threads within an address space. As use herein, the term "local identifier" is used to refer to a location of a thread within the workgroup, or workgroup address space, in contrast to the term "global identifier," which is used herein to refer to a location of a thread in a global address space. The ALU then uses the final/full, local/global IDs to retrieve the data to be operated on from the location specified by the final/full IDs, which can be either final/full local IDs or final/full global IDs.

[13] Note that, as used herein, the term "local ID" is not synonymous with "full local ID." Further note that although many of the examples discussed herein are discussed with reference to a full local ID, which references a thread's location within a workgroup, similar principles can be used for full global IDs, which reference a thread's location within a global address space.

[14] Any or all of the implementations of the processing system discussed herein can include means for determining a warp pattern mode, and means for mapping thread identifiers to the warp-specific local identifiers based on the warp pattern mode. Each warp pattern mode can be associated with an individual table, so that, for example, the origin of a

first warp in a workgroup having given warp pattern will always have the same LID*n* coordinates. The third warp of all workgroups having the same warp pattern will likewise always have the same warp origin. The warp pattern mode can be determined by a warp creator, such as a CS warp creator, based on a type of computation being performed, based on a data size, based on a setting associated with a software shader being executed, or the like. The means for mapping thread identifiers to warp-specific local identifiers can be a list or other data structure stored in hardware or firmware, a register file created by a program of instructions being executed, or the like. Because the list is associated with a particular warp pattern, instructions provided to the processor can specify the warp pattern currently being used passing a parameter indicating the location of a particular register storing a desired warp-pattern table.

[15] In any or all of the above embodiments the final local identifiers can be used to access the plurality of data elements by treating the final local/global identifiers as an index to retrieve data from storage, where the retrieved data is stored in a per-thread general purpose register.

[16] Another aspect of any or all of the various embodiments disclosed combines the warp-specific local identifiers with the warp origins by implementing at least one computer instruction that designates a per-thread general purpose register as a destination, and one or more per warp general purpose registers as a source.

[17] Various embodiments discussed above can be used, alone or in combination, to implement a method of reducing warp ramp up associated with initializing registers with full local identifiers. The methods include storing a warp origin in a per-warp register, reading a warp-specific local identifier into a warp local ID register, generating full local/global identifiers by combining the warp-specific local identifier and the warp origin using a single-instruction-multiple-data (SIMD) arithmetic-logical-unit (ALU), and using the SIMD ALU to access thread data using the full identifiers. In some or all implementations, the full identifiers are used to access the thread data using one or more per-thread general purpose registers in response to the SIMD ALU generating the full identifiers, as opposed to using full identifiers pre-loaded into per-thread GPRs. Accessing the thread data using one or more per-thread GPRs can include retrieving the thread data from storage in a memory device, and storing the retrieved data in the per-thread GPRs

[18] Various methods include determining a warp pattern mode, and selecting a lookup table, from among a plurality of lookup tables, based on the determined warp pattern mode. In any or all of the embodiments disclosed herein, generating the full/final local/global identifier can include executing an instruction provided to the SIMD ALU via an instruction decoder. The instruction can specify at least one per-thread general purpose register as a destination, and the per-warp register associated with the warp being processed as a source.

[19] Any or all of the above embodiments can be implemented using a parallel processing system that stores a warp origin in one or more per-warp registers, obtains warp-specific local identifiers from a data structure mapping thread identifiers to warp-specific local identifiers, combining the warp-specific local identifiers with the warp origin to generate full identifiers, and accessing thread data from storage using the full local identifiers.

[20] The data structure can include a pre-defined lookup table. Regardless of the exact data structure used, various methods can determine a warp pattern mode, and select the particular data structure to be used based on the warp pattern mode. Particular implementations of any of the above described embodiments can include reading warp-specific local identifiers into full or half precision warp local ID registers. Combining the warp-specific local identifiers with the warp origin can be performed by executing an instruction on multiple data items in parallel. The instruction causing the processor to add warp-specific local identifiers associated with each thread in the warp to the same warp origin.

[21] The above embodiments can be used alone or in combination to reducing warp ramp up associated with initializing registers to include full local identifiers. Thus, instead of initializing per-thread registers to include full local or global identifiers, warp origins are stored separately from warp-specific local identifiers. The warp-specific local identifiers are combined with the warp origins in real-time to generate full identifiers in a given address space (*e.g.* workgroup or global), and the full identifiers are used to access thread data.

[22] In various embodiments, the warp origins are stored in a per-warp general purpose register (GPR), and warp-specific local identifiers are stored in a lookup table. A warp pattern mode can be determined, for example by a CS warp creator or an instruction to a processing unit that indicates a particular warp mode or lookup table, and the lookup table can be selected from among a plurality of lookup tables, each different lookup table storing identifiers based on different warp pattern modes. Note that in some embodiments

determining the warp mode determination and the lookup table determination are performed in a single operation.

[23] The full identifiers can be determined by executing an instruction in a single-instruction-multiple-data (SIMD) arithmetic-logical-unit (ALU), where the instruction specifies at least one per-thread general purpose register as a destination and one or more per-warp register as a source.

[24] Various embodiments of processing systems disclosed herein include a single-instruction-multiple-data (SIMD) arithmetic-logical-unit (ALU) including multiple registers, an instruction decoder coupled to the SIMD ALU, and at least one per-thread register coupled to the SIMD ALU. The processing system can also include at least one per-warp register coupled to the SIMD ALU, where the per-warp register includes information indicating a warp origin associated with a warp, the warp including at least one thread.

[25] Various implementations also include a memory, such as a special purpose register, that stores a lookup table including information mapping thread identifiers to warp-specific local identifiers. In various embodiments, the memory stores a plurality of lookup tables associated with different warp patterns. The memory used by some or all of the embodiments discussed herein can include one or more full or half precision warp local ID registers configured to obtain warp-specific local identifiers from a table selected according to a warp pattern mode.

[26] The SIMD ALU is configured to combine the warp-specific local identifiers with the warp origins to generate final identifiers indicating locations of threads within the workgroup or globally. Furthermore, the SIMD ALU can be configured to receive an instruction from the instruction decoder, where the instruction designates the at least one per-thread register as a destination, and designates the at least one per-warp register as a source. The SIMD ALU retrieves the warp origin of a particular warp from one or more per-warp registers associated with the warp to be processed, and retrieves warp-specific local identifiers of threads included in the warp from the lookup table.

[27] The SIMD ALU combines the warp origin and the warp-specific local identifiers to generate full/final identifiers specifying thread locations within the workgroup and uses the full identifiers to access data via particular per-thread registers.

[28] In yet another implementation a non-transitory computer readable medium tangibly embodying a program of instructions to be executed by a processor is employed. The

program of instructions includes one or more instructions that control the processor to retrieve a warp origin of a particular warp from a per-warp register associated with the particular warp, retrieve, from a lookup table, a warp-specific local identifier of a thread included in the particular warp, combine the warp origin and the warp-specific local identifiers to generate a full identifier specifying a thread location within a workgroup or globally, and use the full identifier to access data from storage.

[29] The non-transitory computer readable medium can include at least one instruction causing the processor to select a particular lookup table from multiple different lookup tables based on a warp pattern mode associated with the particular warp. In some implementations, the processor retrieves the warp-specific local identifier from the lookup table via full or half precision warp local ID registers.

[30] In various embodiments, a single instruction causes the processor to perform all of the following functions: retrieve a warp origin of a particular warp from a per-warp register associated with the particular warp; retrieve, from a lookup table, a warp-specific local identifier of a thread included in the particular warp; combine the warp origin and the warp-specific local identifiers to generate a full identifier specifying a thread location within a workgroup. In various embodiments, the single instruction specifies an operation that causes the processor to combine the warp origin and the warp-specific local identifiers to generate the full identifier, specifies the per-thread register as a destination; and specifies the per-warp register as a source to retrieve the warp origin.

BRIEF DESCRIPTION OF THE DRAWING(S)

[31] Figure 1 is a block diagram illustrating a prior art arrangement in which a compute shader (CS) warp creator pushes local ID information into per-thread general purpose registers (GPRs) prior to warp generation or execution;

[32] Figure 2 is a block diagram illustrating a processing system including a graphics processing unit (GPU) having multiple shader (processing) cores, in accordance with various embodiments of the present disclosure;

[33] Figure 3 is a block diagram illustrating a shader (processing) core, in accordance with various embodiments of the present disclosure;

[34] Figure 4 is a block diagram illustrating a system that calculates local IDs of threads dynamically, instead of pre-loading the local IDs into general purpose registers (GPRs), in accordance with various embodiments of the present disclosure;

[35] Figure 5 is a diagram illustrating the relationship of thread locations to workgroup and global address spaces, according to various embodiments of the present disclosure;

[36] Figure 6 is a diagram illustrating a warp local ID lookup table in linear mode, according to various embodiments of the present disclosure;

[37] Figure 7 is a diagram illustrating a warp local ID lookup table in tiled mode, according to various embodiments of the present disclosure; and

[38] Figure 8 is a flowchart illustrating dynamic generation of full local IDs, in accordance with various embodiments of the present disclosure.

DETAILED DESCRIPTION

[39] Referring to Figure 2, a block diagram illustrating a processing system including a graphics processing unit (GPU) including multiple shader (processing) cores, or simply processing cores, is illustrated, in accordance with various embodiments of the present disclosure. In various embodiments, processing system 200 can be implemented in, among other things, desktop computers, laptop computers, gaming appliances, smart phones, feature phones, televisions, set-top boxes, tablets, watches, virtual reality appliances, graphics processing cards or assemblies suitable for inclusion in larger or more comprehensive systems, or the like.

[40] Processing system 200 includes GPU 210, Level 3 cache 230, and other devices 240. GPU 210 includes processing cores 232, 234, and 236; and Level 2 cache 228. Each of the processing cores 232, 234, and 236 includes corresponding Level 1 cache 222, 224, and 226.

[41] As used herein, the term shader core is sometimes used interchangeably with the term processing core, although the term shader core is commonly used to refer to processing units dedicated to calculating vertex or pixel shading in a graphics environment. As used herein, a shader core is not necessarily limited to performing processing a graphics environment. Instead, the term shader core, as used herein, refers to a processing core that is specially configured to perform the type of parallel processing that is typically performed in graphics processing. Because the use of conventional shader cores is so common in GPU's, the

majority of this disclosure is discussed in the context of GPUs and shader cores, even though many, if not all, of the embodiments disclosed herein can be implemented in single instruction multiple data (SIMD) processors other than GPUs.

[42] Other devices 240 can include a general purpose microprocessor, application specific integrated circuits (ASICs), central processing unit (CPU) used by processing system 200, and discrete logic circuitry. Additionally, other devices 240 can include various types of memory circuits, chips, solid state drives, including but not limited to random access memory (RAM), static RAM (SRAM), dynamic RAM (DRAM), read only memory (ROM), electrically erasable programmable read only memory (EEPROM), user input/output devices, wireless and wired network interfaces, and like. Various portions of processing system 200 can be interconnected using communication buses or other types of electrical connections employing various protocols, as well as using any suitable type of communication protocol. Some or all of the illustrated components of processing system 200 can be implemented on a single semiconductor surface substrate (*i.e.* on the same chip), in assemblies including multiple chips, or some combination thereof.

[43] Referring next to Figure 3, a block diagram illustrating a shader (processing) core will be discussed in accordance with various embodiments of the present disclosure. As used herein, the term "shader" is used interchangeably with the term "compute shader (CS)," and refers to circuitry, modules, etc. used for computing essentially arbitrary information, as opposed to performing graphics rendering. While a compute shader can do rendering, it is generally used for tasks not directly related to drawing triangles and pixels. Processing core 300 is illustrated as a single instruction multiple data (SIMD) processing core that includes a SIMD arithmetic logic unit (ALU) 313 capable of performing the same operation on multiple data elements in parallel, *e.g.* at substantially the same time. Processing core 300 is, in at least some embodiments, includes circuitry that implements a compute shader (CS), in addition to circuitry that implements a graphics shader.

[44] Processing core 300 also includes instruction scheduler 303, which obtains instructions from instruction cache 301, and provides the instructions to instruction decoder 307. Instruction decoder 307, which can be, for example, a combinatorial circuit, uses information included in per-warp address register 305 to decode instructions obtained from instruction scheduler 303. Instruction decoder 307 can provide the decoded instruction to SIMD ALU 313. Decoding the instruction can include, among other things, translating

instruction codes obtained from instruction scheduler 303 into addresses from which the appropriate instruction can be obtained.

[45] SIMD ALU 313 obtains data to be processed from per-thread general purpose registers (GPRs) 311 and on-chip constants storage 325. The data stored in the per-thread GPRs is obtained from graphics (Gfx) warp creators 309. In general, Gfx warp creators 309 organize graphics processing threads into groups, or warps that can be executed in parallel by SIMD ALU 313. Gfx warp creators can function to generate thread and warp identifiers based on information obtained from various state registers (not illustrated). Thus, if a SIMD ALU is capable of processing 32 threads concurrently (SIMD32), 32 threads can be grouped into a warp by Gfx warp creators 309. Data associated with each of the 32 different threads in the warp can be stored in corresponding per-thread GPRs.

[46] In the illustrated embodiment, SIMD ALU 313 also includes warp local ID (LID) registers 333. In some embodiments, SIMD ALU 313 obtains data from per-warp GPRs 329 and from lookup tables for special registers 331 via warp LID registers 333. The lookup tables can be, in various embodiments, stored in a non-volatile memory, *e.g.*, persistent tables stored as firmware, or loaded into a volatile memory during a system boot process or as part of a software program initialization. Compute shader (CS) warp creator 327, like graphics (Gfx) warp creators, organizes threads into warps for concurrent processing, which can include generating identifiers. In various embodiments, the data organized by CS warp creator 327 differs from the data organized into threads by GFx warp creators 309 primarily, although not necessarily exclusively, by the type of data handled by each. In general, CS warp creator 327 generates warps, and stores warp data in per-warp GPRs 329. In various embodiments, data to be processed for individual threads within a warp can be loaded into per-thread GPRs.

[47] The results generated by SIMD ALU 313 by processing the warps can be sent to outlet control 315, which directs the results to special function unit 323 or emit buffer 317. Special function unit 323 can be used, for example, to implement transcendental instructions or functions, such as sin, cosine, reciprocal, and square root functions. Emit buffer 317 can be used to emit the results to texture unit 319 or load unit 321.

[48] Texture unit 319 applies texture operations to pixels. Load unit 321 can be a load–store unit responsible for executing load and/or store instructions, generating virtual

addresses of load and store operations, loading data from a memory, or storing data back to memory from registers.

[49] Referring next to Figure 4, a system 400, which calculates full local IDs of threads dynamically, instead of pre-loading the local IDs into general purpose registers (GPRs), will be discussed in accordance with various embodiments of the present disclosure. Note that similar techniques and systems can be used to calculate global IDs of threads dynamically. In general use of system 400 can reduce, and even fully eliminate, a warp ramp up process in which local IDs are pre-loaded into per-thread GPRs. By doing so, performance of a compute shader (CS) can be improved by reducing warp ramp up time to zero.

[50] Local IDs within a warp 410 are controlled by warp pattern modes, which can be determined by a CS warp creator. In various embodiments, a lookup table is stored in lookup tables for special registers 331 (Figure 3) for each supported warp pattern mode. A mode is associated with a particular combination of warp size and tiling. Warp LID registers 333 (Figure 3), sometimes referred to herein as "special registers," can be used to read data from one or more lookup tables for special registers, such as Warp LID registers 333. Each thread of a warp will get a unique ID from the table. In at least some embodiments, the ID is multi-dimensional.

[51] The warp origin of each warp within the workgroup is stored in per-warp GPRs. All threads within a warp share the same warp origin. The warp origin is combined with the warp local ID of a thread, for example by adding, concatenating, appending, prepending, or the like, to get a final local ID of that thread within a WG. That is, the final local ID of a thread, sometimes referred to as the full local ID, specifies the location of the thread within the workgroup. Combining a warp origin based on a workgroup address space can be used to generate a workgroup based full identifier, while using a warp origin based on a global address space can be used to create a full global ID. In various embodiments, a full global ID, which identifies a location of a thread within a global address space, and a full local ID, which identifies the location of a thread within a workgroup, employ the same techniques described herein, with the primary difference being the use of a global warp origin for the full global ID, and a workgroup warp origin for the full local ID.

[52] For example, a CS warp creator 327 determines a warp origin 409 and stores it in per-warp GPRs 429. A thread ID to warp Local ID mapper 427, for example lookup tables for special registers 331 (Figure 3), is used to store a LIDX 405, which represents an offset from

the warp origin of a thread in the "x" direction, and LIDY 407, which represents an offset from the warp origin of a thread in the "y" direction. For each thread, the SIMD processor retrieves the warp origin 409 from per-warp GPRs 429, and combines it with LIDX 405 and LIDY 407 obtained from thread ID to warp Local ID mapper 427 via warp LID registers 333 (Figure 3) to obtain the final local identifiers of the individual threads.

[53] The final local identifiers are used to access thread data stored within the warp being processed. So, for example, when data for each thread of a warp being processed is stored in per-thread GPRs, Thread 0 full local identifier 421 can be used to retrieve data from the corresponding location within Thread 0's per-thread GPR, likewise, Thread1 full local identifier 423 can be used to retrieve data from the corresponding location within Thread 1's per-thread GPR, Thread *n* full local identifier 425 can be used to retrieve data from the corresponding location within Thread *n*'s per-thread GPR, and so on.

[54] Referring next to Figure 5, a diagram illustrating the relationship of thread locations to workgroup and global address spaces will be discussed in accordance with various embodiments of the present disclosure. In some embodiments, the global address space can include a domain, a dispatch, or some other grouping of data arranged to include multiple warps and/or workgroups. As illustrated in Figure 5, global address space 501 is used to store threads arranged into warps, and further arranged into workgroups 517, 518, 519, and 520. Thread locations can be identified using a "workgroup" address space, specifying the thread's location within the workgroup, or using a "global" address space, specifying the thread's location within the global data address space

[55] Figure 5 illustrates workgroups WG517-WG520, each of which includes four warps, WRP0-WRP3. Workgroup WG520 will be used as the basis for discussing, the workgroup address space. The location of a thread within a workgroup address space can be specified by a set of x, y coordinates. For example, each of the warps WRP0 503, WRP1 505, WRP2 507, and WRP 3 509 included in workgroup WG520 includes thirty-two threads numbered 0-31. The address of thread 0 of WRP0 503, relative to workgroup WG520, can be expressed as $0x,0y$, while the address of thread 31 of WRP0 503 can be expressed as $7x,3y$. Similarly, the address of thread 0 of WRP3 509, relative to workgroup WG520, can be expressed as $8x,4y$, while the address of thread 31 of WRP3 509 can be expressed as $15x,7y$.

[56] In the illustrated example, the address of a particular thread within a particular workgroup will be the same, regardless of what workgroup the warp is in. So, for example,

the address of the thread 0, warp 0 of workgroup 517 (WG517,WRP0) will have the same address (within workgroup space) as thread 0, warp 0 of workgroups WG518, WG519, and WG520 – namely 0*x*,0*y*. Similarly, thread 31 of warp3 of workgroup WG517 will have the same address (within workgroup space) as the thread 31 of the warp3 of workgroups, WG518, WG519, and WG520 – namely 15*x*,7*y*.

[57] , The location of a thread within a global address space, can be specified using different values than those used to specify the location of a thread in workgroup space. For example, the location of a thread within global address space can be specified using a set of x, y, z coordinates. For example, the global address space location of thread 0 in workgroup WG520, WRP0 503 can be specified as 16*x*,0*y*,2*z*, compared to specifying the location of that same thread as 0*x*,0*y* in workgroup address space. As another example, the global address space location of thread 31 of WRP0 503 in workgroup WG520, warp 0, (WG520,WRP0), could be specified as 23*x*,3*y*,2*z*, compared to the workgroup address space location of 7*x*,0*y*,2*z*.

[58] In various embodiments, therefore, the global address of corresponding threads in particular workgroups will not be the same. For example, thread 0 of WRP0 in workgroup WG518 (WG518,WRP0), has a global address of 16*x*,0*y*,1*z*, will is different from the global ID of thread 0 of warp 0 of workgroup 519 (WG519,WRP0), which is 0*x*,0*y*,2*z*. This is in contrast to addresses in the workgroup address space, where the workgroup address of corresponding threads in particular workgroups are the same.

[59] Workgroup WG520, as illustrated, includes four warps, namely WRP0 503, WRP1 505, WRP2 507, WRP3 509. Each of the warps is divided into 8 quads, and each quad is divided into 4 threads. Numbers along the x and y axes provide row and column coordinates by which locations within a workgroup can be specified. In this example, each of the warps includes 32 threads, but depending on the processor capabilities, each warp may include more or fewer threads. Other multidimensional data arrangements, for example a different number or arrangement of workgroups, can be used consistent with the present disclosure. Additionally, warps do not necessarily need to be grouped into workgroups, and each workgroup may or may not correspond to a z axis coordinate.

[60] When using a workgroup addressing space, the warp origin of WRP0 503 is 0*x*,0*y*, the warp origin of WRP1 505 is 8*x*,0*y*, the warp origin of WRP2 507 is 0*x*,4*y*, WRP3 509 8*x*,4*y*. In various embodiments, when employing workgroup addressing space, a warp origin can be

stored using 2 per-warp GPRs (wGPRs). A first wGPR can be used to store the x value, and a second wGPR can be used to store the y value. Thus, for example, the x value for the warp origin of WRP0 503 is 0, and is stored in the first wGPR, while the y value of the warp origin of WRP0 503 is also 0, and is stored in the second wGPR. In a similar manner the warp origin of WRP1 505 is 8*x*,0*y*, and the x value of 8 can be stored in the first wGPR, while the y value of 0 is stored in the second wGPR. The warp origins of warp origin of WRP2 507 (0*x*,4*y*) and the warp origin of WRP3 509 (8*x*,4*y*) are stored in a similar manner.

[61] As illustrated, when using a global addressing space, the warp origin of WRP0 503 of WG520 is 16*x*,0*y*,2*z*. Likewise, the warp origin of WRP1 505 is 24*x*,0*y*,2*z*, the warp origin of WRP2 507 is 16*x*,4*y*,2*z*, and the warp origin of WRP3 509 is 24*x*,4*y*,2*z*. In various embodiments, when employing global addressing space, a warp origin can be stored using 3 per-warp GPRs (wGPRs). A first wGPR can be used to store the x value, a second wGPR can be used to store the y value, and a third wGPR can be used to store the z value. Thus, for example, the x value for the warp origin of WRP0 503 is "0," and is stored in the first wGPR; the y value of the warp origin of WRP0 503 is also "0," and is stored in the second wGPR, and the z value of WRP0 503 is "3," and is stored in the third wGPR.

[62] The warp local ID, sometimes referred to herein as simply the local ID, represents an *x*, *y* offset from the warp origin. In various embodiments, warp local IDs can be stored in warp LID registers 333 (Figure 3). For example, register LocalID.*n* of the second thread (thread 1) of WRP3 509 is represented by the pair of values 1*x*,0*y* stored in LocalID.x, LocalID.y, which indicates the offset of thread 1 from the warp origin. Note that the second thread (thread 1) of WRP0 503 is represented by the same pair of values 1*x*,0*y* stored in LocalID.x, LocalID.y, because both threads have the same offset from their respective warp origins. In various implementations employing linear, as opposed to tiled, warp patterns, the local ID LocalID.*n* may be expressed by a single value, LocalID.x.

[63] In some embodiments, the warp local ID registers are implemented using half-precision compute shader warp local ID registers LIDXH and LIDYH. In some implementations one or more full-precision compute shader warp local ID registers LIDX and LIDY can be used in addition to, or in place of LIDXH and LIDYH. LIDXH or LIDX holds the warp local ID in the X direction relative to the warp origin. LIDYH or LIDY holds the warp local ID in Y direction relative to the warp origin. The same values stored in LIDX (LIDXH) and LIDY (LIDYH), which represent offsets from the warp origin, can be used for either global addressing space or workgroup addressing space.

[64] In various embodiments, the local ID registers can be used directly in an instruction. The possible values of (LIDXH, LIDYH) or (LIDX, LIDY) range across the warp size, *i.e.*, (0, 0) to (WarpSize.x - 1, WarpSize.y - 1).

[65] Workgroup (WG) local IDs range across (0, 0) to (WorkGroupSize.x - 1, WorkGroupSize.y - 1). The compiler can compute the full local ID of a thread (*i.e.* in the workgroup address space) by adding the warp local IDs to the local, or workgroup warp origin, which is expressed as a WG local ID at the origin of the warp including the thread under consideration. The compiler can obtain the workgroup warp origin from per-warp registers (wGPRs) assigned by hardware.

[66] The compiler can compute the full global ID of a thread (*i.e.* in the global address space) by adding the warp local IDs to the global warp origin, which is expressed as a global ID at the origin of the warp including the thread under consideration. The compiler can obtain the global warp origin from per-warp registers (wGPRs) assigned by hardware.

[67] In the following example, assume that the first and second wGPRs used to store the warp origin are represented by w(n) and w(n+1). The formula for generating the pair of values representing the full, or workgroup, local ID (LocalID.x and LocalID.y) is as follows:

$$\text{LocalID.x of thread A} = \text{LIDX (A)} + w(n)$$
$$\text{LocalID.y of thread A} = \text{LIDY (A)} + w(n+1)$$

[68] That is to say, the full local (workgroup) ID of thread A includes both an x and a y component. The x component of the full local (workgroup) ID of thread A is determined by adding the x value of the local (workgroup) warp origin stored in the first wGPR w(*n*) to the x value of thread A's warp local ID. The local (workgroup) y value is similarly determined.

[69] In a similar manner, the full global ID of thread A includes an x, y, and z component. The x component of the full global ID of thread A is determined by adding the x value of the global warp origin stored in the first wGPR w(*n*) to the x value of thread A's warp local ID. This is the same local ID used to calculate the local (workgroup) warp origin, but the result is a global result because a global warp origin is used. The global y value is similarly determined. The z value can be read directly from the wGPR used to store the z value of the global warp origin.

[70] Consider the following numerical examples for the four threads included in Quad3 of Warp 3 of Workgroup WG520:

[71] The warp origin of WRP3 509 within Workgroup WG520 is (8*x*, 4*y*) which is stored in two wGPRs, w(n) and w(n+1). The full local IDs of threads 0, 1, 2, and 3 of QUAD3 in WRP3 509 are:

LocalID.x of Thread 12 of WARP3 = w(n) + LIDX = 8 + 2 = 10
LocalID.y of Thread 12 of WARP3 = w(n+1) + LIDY = 4 + 2 = 6

LocalID.x of Thread 13 of WARP3 = w(n) + LIDX = 8 + 3 = 11
LocalID.y of Thread 13 of WARP3 = w(n+1) + LIDY = 4 + 2 = 6

LocalID.x of Thread 14 of WARP3 = w(n) + LIDX = 8 + 2 = 10
LocalID.y of Thread 14 of WARP3 = w(n+1) + LIDY = 4 + 3 = 7

LocalID.x of Thread 15 of WARP3 = w(n) + LIDX = 8 + 3 = 11
LocalID.y of Thread 15 of WARP3 = w(n+1) + LIDY = 4 + 3 = 7

[72] The global warp origin of WRP3 509 is (24*x*,4*y*,2*z*) which is stored in three wGPRs, w(n), w(n=1), and w(n=2). The full local IDs of threads 0, 1, 2, and 3 of QUAD3 in WRP3 509 are:

GlobalID.x of Thread 12 of WARP3 = w(n) + LIDX = 24 + 2 = 26
GlobalID.y of Thread 12 of WARP3 = w(n+1) + LIDY = 4 + 2 = 6
GlobalID.z of Thread 12 of WARP3 = w(n+1) + LIDY = 2

GlobalID.x of Thread 13 of WARP3 = w(n) + LIDX = 24 + 3 = 27
GlobalID.y of Thread 13 of WARP3 = w(n+1) + LIDY = 4 + 2 = 6
GlobalID.z of Thread 12 of WARP3 = w(n+1) + LIDY = 2

GlobalID.x of Thread 14 of WARP3 = w(n) + LIDX = 24 + 2 = 26
GlobalID.y of Thread 14 of WARP3 = w(n+1) + LIDY = 4 + 3 = 7
GlobalID.z of Thread 12 of WARP3 = w(n+1) + LIDY = 2

GlobalID.x of Thread 15 of WARP3 = w(n) + LIDX = 24 + 3 = 27
GlobalID.y of Thread 15 of WARP3 = w(n+1) + LIDY = 4 + 3 = 7

GlobalID.z of Thread 12 of WARP3 = w(n+1) + LIDY = 2

[73] The above discussion illustrates that per-thread GPRs are not needed to calculate full, local (workgroup) or global IDs. Only wGPRs and special registers LIDX and LIDY are used in the above calculations. Thus, there is no need to ramp up a warp by pushing local IDs into per-thread GPRs as we do not need per-thread GPRs to calculate full local (workgroup) or global IDs in a CS shader.

[74] Referring next to Figure 6 a warp local ID lookup table for linear mode in a SIMD32 is illustrated, and will be discussed according to various embodiments of the present disclosure. Linear mode lookup table 600 can be used to map thread identifiers to warp-specific local identifiers, and to provide warp local IDs for calculation of full local IDs when thread data is arranged in a linear mode.

[75] As discussed above with reference to Figure 5, the full local, or global) ID of a thread is determined by adding the warp origin (stored in a wGPR) to the warp specific local identifier obtained from a CS warp local ID register LIDX/LIDXH or LIDY/LIDYH, sometimes referred to herein as a "special" register. The value(s) stored in the CS warp local ID register can be read from a lookup table associated with a warp mode.

[76] Because linear mode lookup table 600 arranges thread data in a linear fashion, both X and Y components are not needed to express a final, or full local ID. Thus, using thread 16 as an example, the wGPR used to store the X value for ThreadID 16 will store a value of 16, while the wGPR used to store the Y value for ThreadID 16 will either not be used, or will store a 0 value. Similarly, using ThreadID 14 as an example, the wGPR used to store the X value for ThreadID 14 will store a value of 14, while the wGPR used to store the Y value for thread 14 will either not be used, or will store a 0 value.

[77] Referring next to Figure 7, a warp local ID lookup table in tiled mode will be discussed according to various embodiments of the present disclosure. Tiled mode lookup table 700 can be used to provide warp local IDs for calculation of full local, or global, IDs when thread data is arranged in a tiled mode.

[78] As previously discussed, the full local or global ID of a thread can be determined by adding the warp origin (stored in a wGPR) to the warp specific local identifier obtained from a CS warp local ID register LIDX/LIDXH or LIDY/LIDYH. The value(s) stored in the CS warp local ID register can be read from a lookup table associated with a warp mode.

[79] Similar to linear mode lookup table 600, tiled mode lookup table 700 is used to map thread identifiers to warp-specific local identifiers, but tile mode lookup table 700 is used for mapping data in a tiled format like the tiled warp mode illustrated in Figure 5. To illustrate the difference between the tiled and non-tiled format modes, the following examples discuss the same threads discussed above with reference to Figure 6.

[80] In tiled mode lookup table 700, the wGPR used to store the X value for ThreadID 16 will store a value of 4, while the wGPR used to store the Y value for ThreadID 16 will store a value of 0. In contrast to linear mode lookup table 600, however, both X and Y components are required to fully define the warp specific local ID of any particular thread. The wGPR used to store the X value of the warp specific local ID for ThreadID 14 will store a value representing 2, while the wGPR used to store the Y value of the warp specific local ID for ThreadID 14 will store a value of 3.

[81] Referring next to Figure 8, a method 800 of dynamically generating full, or final, local IDs will be discussed in accordance with various embodiments of the present disclosure. As illustrated by block 805, a warp origin of a warp to be processed is stored in a per-warp register. The warp origin can be determined by a CS warp creator, such as CS warp creator 327 (Figure 3. The determination made by the CS warp creator can be based on a warp mode that establishes whether threads within a warp are arranged linearly or in a tiled fashion. Once determined, the warp origin can be stored in the appropriate per-warp register by a compute shader warp creator. In at least one embodiment, the appropriate per-warp register for a particular thread can be determined based a hardware assignment, but other embodiments per-warp registers can assign by software, firmware, or some combination thereof. In various embodiments, the warp origin can be based on a workgroup address space, or a global address space.

[82] As illustrated by block 807, a warp pattern mode is determined. The warp pattern mode can be determined by a warp creator, such as CS warp creator 327 (Figure 3), based on a type of computation being performed, based on a data size, based on a setting associated with a software shader being executed, or the like. In some embodiments, an instruction operand or parameter can be sent to a processing unit to provide information notifying the processing unit which of multiple different warp pattern modes have been selected, or identifying a particular register storing data associated with a particular warp pattern mode.

[83] As illustrated by block 809, the warp pattern mode can be used to select a lookup table corresponding to the appropriate warp pattern mode. For example, a linear mapping table can be selected for a linear warp mode, or any of multiple different tiled mapping tables can be used for different tiled warp modes, depending on the particular type of tiling used by any particular warp mode. The selection of a lookup table can be explicit or implicit. For example a particular table, or other data structure such as a register file or the like, can be explicitly identified by mapping a warp type to a warp table, and then loading data from that table into an appropriate "special" or warp LID register. In other embodiments, implicit identification of a lookup table can be performed by identifying a register storing data from the appropriate table.

[84] As illustrated by block 811, warp-specific local identifiers associated with particular threads, sometimes referred to herein as warp local IDs, are selected from the lookup table by cross referencing a thread ID with a warp local ID associated with the thread ID. The warp local ID can be a set of one or more values specifying offsets relative to a warp origin, allowing each thread within a particular warp to be located if the warp origin is known.

[85] As illustrated by block 813, for each thread of a warp, a warp-specific local ID, obtained from special warp local ID registers, is combined with either a global or workgroup warp origin, obtained from a wGPR associated with the warp being processed, to generate a full local or global identifier. The full local or global identifier specifies a location of the thread relative to a workgroup, or in a global address space, allowing for example, the location of thread 4 in warp 0 to be distinguished from the location of thread 4 in warp 3.

[86] In various aspects of the described embodiments, an instruction executed by the processor can be used to generate the full, or final, local ID as follows:

GENLID Instruction

Function

Generates local ID

Syntax

genlid dst, src0

Valid encoding

dst

src0

Description

dst: GPRs to hold the local ID of the work item in the workgroup

src0: wGPRs holding the starting local ID of the warp in X and Y directions

Execution

Generate the local ID for the warp.

dst[0] = src0[0] + local ID within the warp in X direction; // both linear and tiled mode

dst[1] = src0[1] + local ID within the warp in Y direction; // tiled mode only

[87] In the above example GENLID instruction the warp is aligned to 32 work items in linear mode or 8 x 4 work item boundaries in tiled mode, and hardware uses pre-defined tables to look up the local ID.

[88] As illustrated by block 815, thread data can be retrieved from storage based on the full local identifier. In some embodiments, the data can be retrieved via per-thread GPRs by using the full local identifier to access the thread data from a memory used to store the warp, and cause the thread data to be moved into appropriate per-thread GPRs.

[89] As illustrated by block 817, the processor operates on the thread data retrieved at block 815.

[90] As may be used herein, the terms "substantially" and "approximately" provides an industry-accepted tolerance for its corresponding term and/or relativity between items. Such an industry-accepted tolerance ranges from less than one percent to fifty percent and corresponds to, but is not limited to, component values, integrated circuit process variations, temperature variations, rise and fall times, and/or thermal noise. Such relativity between items ranges from a difference of a few percent to magnitude differences. As may also be used herein, the term(s) "configured to", "operably coupled to", "coupled to", and/or "coupling" includes direct coupling between items and/or indirect coupling between items via an intervening item (e.g., an item includes, but is not limited to, a component, an element, a circuit, and/or a module) where, for an example of indirect coupling, the intervening item does not modify the information of a signal but may adjust its current level, voltage level, and/or power level. As may further be used herein, inferred coupling (i.e., where one element is coupled to another element by inference) includes direct and indirect coupling between two items in the same manner as "coupled to". As may even further be used herein, the term "configured to", "operable to", "coupled to", or "operably coupled to" indicates that an item

includes one or more of power connections, input(s), output(s), etc., to perform, when activated, one or more its corresponding functions and may further include inferred coupling to one or more other items. As may still further be used herein, the term "associated with", includes direct and/or indirect coupling of separate items and/or one item being embedded within another item.

[91] As may also be used herein, the terms "processing module", "processing circuit", "processor", and/or "processing unit" may be a single processing device or a plurality of processing devices. Such a processing device may be a microprocessor, micro-controller, digital signal processor, microcomputer, central processing unit, field programmable gate array, programmable logic device, state machine, logic circuitry, analog circuitry, digital circuitry, and/or any device that manipulates signals (analog and/or digital) based on hard coding of the circuitry and/or operational instructions. The processing module, module, processing circuit, and/or processing unit may be, or further include, memory and/or an integrated memory element, which may be a single memory device, a plurality of memory devices, and/or embedded circuitry of another processing module, module, processing circuit, and/or processing unit. Such a memory device may be a read-only memory, random access memory, volatile memory, non-volatile memory, static memory, dynamic memory, flash memory, cache memory, and/or any device that stores digital information. Note that if the processing module, module, processing circuit, and/or processing unit includes more than one processing device, the processing devices may be centrally located (e.g., directly coupled together via a wired and/or wireless bus structure) or may be distributedly located (e.g., cloud computing via indirect coupling via a local area network and/or a wide area network). Further note that if the processing module, module, processing circuit, and/or processing unit implements one or more of its functions via a state machine, analog circuitry, digital circuitry, and/or logic circuitry, the memory and/or memory element storing the corresponding operational instructions may be embedded within, or external to, the circuitry comprising the state machine, analog circuitry, digital circuitry, and/or logic circuitry. Still further note that, the memory element may store, and the processing module, module, processing circuit, and/or processing unit executes, hard coded and/or operational instructions corresponding to at least some of the steps and/or functions illustrated in one or more of the Figures. Such a memory device or memory element can be included in an article of manufacture.

[92] One or more embodiments of an invention have been described above with the aid of method steps illustrating the performance of specified functions and relationships thereof.

The boundaries and sequence of these functional building blocks and method steps have been arbitrarily defined herein for convenience of description. Alternate boundaries and sequences can be defined so long as the specified functions and relationships are appropriately performed. Any such alternate boundaries or sequences are thus within the scope and spirit of the claims. Further, the boundaries of these functional building blocks have been arbitrarily defined for convenience of description. Alternate boundaries could be defined as long as the certain significant functions are appropriately performed. Similarly, flow diagram blocks may also have been arbitrarily defined herein to illustrate certain significant functionality. To the extent used, the flow diagram block boundaries and sequence could have been defined otherwise and still perform the certain significant functionality. Such alternate definitions of both functional building blocks and flow diagram blocks and sequences are thus within the scope and spirit of the claimed invention. One of average skill in the art will also recognize that the functional building blocks, and other illustrative blocks, modules and components herein, can be implemented as illustrated or by discrete components, application specific integrated circuits, processors executing appropriate software and the like or any combination thereof.

[93] The one or more embodiments are used herein to illustrate one or more aspects, one or more features, one or more concepts, and/or one or more examples of the invention. A physical embodiment of an apparatus, an article of manufacture, a machine, and/or of a process may include one or more of the aspects, features, concepts, examples, etc. described with reference to one or more of the embodiments discussed herein. Further, from figure to figure, the embodiments may incorporate the same or similarly named functions, steps, modules, etc. that may use the same or different reference numbers and, as such, the functions, steps, modules, etc. may be the same or similar functions, steps, modules, etc. or different ones.

[94] Unless specifically stated to the contra, signals to, from, and/or between elements in a figure of any of the figures presented herein may be analog or digital, continuous time or discrete time, and single-ended or differential. For instance, if a signal path is shown as a single-ended path, it also represents a differential signal path. Similarly, if a signal path is shown as a differential path, it also represents a single-ended signal path. While one or more particular architectures are described herein, other architectures can likewise be implemented that use one or more data buses not expressly shown, direct connectivity between elements,

and/or indirect coupling between other elements as recognized by one of average skill in the art.

[95] The term "module" is used in the description of one or more of the embodiments. A module includes a processing module, a processor, a functional block, hardware, and/or memory that stores operational instructions for performing one or more functions as may be described herein. Note that, if the module is implemented via hardware, the hardware may operate independently and/or in conjunction with software and/or firmware. As also used herein, a module may contain one or more sub-modules, each of which may be one or more modules.

[96] While particular combinations of various functions and features of the one or more embodiments have been expressly described herein, other combinations of these features and functions are likewise possible. The present disclosure of an invention is not limited by the particular examples disclosed herein and expressly incorporates these other combinations.

CLAIMS

What is claimed is:

1. A processing system comprising:

means for concurrently performing a same processing operation on a plurality of data elements, the plurality of data elements arranged as threads within one or more warps;

means for mapping thread identifiers to warp-specific local identifiers, the warp-specific local identifiers indicating positions of the threads within the one or more warps, relative to warp origins;

means for storing the warp origins in one or more per-warp registers wherein the warp origins designate starting positions of the one or more warps within an address space;

means for combining the warp-specific local identifiers with the warp origins to generate final identifiers indicating locations of the threads within the address space; and

means for using the final identifiers to access the plurality of data elements.

2. The processing system of claim 1, further comprising:

at least one lookup table; and

means for accessing the warp-specific local identifiers from the at least one lookup table.

3. The processing system of claims 1 or 2, further comprising:

means for determining a warp pattern mode; and

means for mapping thread identifiers to the warp-specific local identifiers based on the warp pattern mode.

4. The processing system of claim 1, wherein the means for using the final identifiers to access the plurality of data elements:

using the final identifiers as an index to retrieve thread data from storage via a per-thread register.

5. The processing system of claims 1 or 4, wherein the means for combining the warp-specific local identifiers with the warp origins includes:

at least one computer instruction designating a per-thread general purpose register as a destination and a per warp general purpose register as a source.

6. A method of reducing warp ramp up associated with initializing registers with full identifiers, the method comprising:

reading, into a warp local ID register, a warp-specific local identifier obtained from a lookup table mapping thread identifiers to warp-specific local identifiers, the warp-specific local identifier indicating positions of threads within a warp relative to the warp origin;

storing the warp origin in one or more per-warp registers, wherein the warp origin designates a starting position of a warp within an address space;

generating full identifiers by combining the warp-specific local identifier and the warp origin using a single-instruction-multiple-data (SIMD) arithmetic-logical-unit (ALU), the full identifiers indicating locations of the threads within the address space; and

using the SIMD ALU to access thread data using the full identifiers.

7. The method of claim 6, wherein:

storing the warp origin includes storing a warp origin value indicating a starting position of the warp within a workgroup; and wherein

generating full identifiers includes generating full local identifiers indicating locations of the threads within the workgroup.

8. The method of claim 6, wherein:

storing the warp origin includes storing a warp origin value indicating a starting position of the warp within a global address space; and wherein

generating full identifiers includes generating full global identifiers indicating locations of the threads within the global address space.

9. The method of claim 6 or 8 wherein storing the warp origin in the one or more per-warp registers includes:

storing, in a plurality of per-warp registers, multidimensional global coordinates indicating a starting position of a particular warp within a global address space.

10. The method of claim 6, 7, 8, or 9 further comprising:

in response to generation of the full identifiers by the SIMD ALU, using the full identifiers to access the thread data from storage.

11. The method of claim 6, 7, 8, 9 or 10, further comprising:

determining a warp pattern mode; and

selecting the lookup table, from among a plurality of lookup tables, different lookup tables storing identifiers based on different warp pattern modes.

12. The method of claims 6, 7, 8, 9, 10, or 11 wherein generating the full identifiers includes:

executing an instruction provided to the SIMD ALU via an instruction decoder, the instruction specifying:

at least one per-thread general purpose register as a destination; and

the one or more per-warp registers as a source.

13. A method for use in a parallel processing system, the method comprising:

storing a warp origin in one or more per-warp registers, wherein the warp origin designates a starting position of a warp within an address space;

obtaining warp-specific local identifiers from a data structure mapping thread identifiers to warp-specific local identifiers, the warp-specific local identifiers indicating positions of threads within the warp relative to the warp origin;

combining the warp-specific local identifiers with the warp origin to generate full identifiers; and

storing thread data into a per-thread register using the full identifiers.

14. The method of claim 13, wherein:

storing the warp origin includes storing a warp origin value indicating a starting position of the warp within a workgroup; and wherein

combining the warp-specific local identifiers with the warp origin to generate full identifiers includes generating full local identifiers indicating locations of the threads within the workgroup.

15. The method of claim 13, wherein:

storing the warp origin includes storing, in a plurality of per-warp registers, multidimensional global coordinates indicating a starting position of a particular warp within a global address space; and wherein

combining the warp-specific local identifiers with the warp origin to generate full identifiers includes generating full global identifiers indicating locations of the threads within the global address space.

16. The method of claim 13, 14, or 15 wherein:

the data structure includes a pre-defined lookup table.

17. The method of claim 13, 14, 15, or 16, further comprising:

determining a warp pattern mode; and

selecting the data structure based on the warp pattern mode.

18. The method of claims 13, 14, 15, 16, or 17, wherein obtaining warp-specific local identifiers from a data structure includes:

reading the warp-specific local identifiers into full or half precision warp local ID registers.

19. The method of claim 13, wherein:

combining the warp-specific local identifiers with the warp origin is performed by executing an instruction on multiple data in parallel, the instruction configured to cause warp-specific local identifiers associated with each thread in the warp to be added to the warp origin.

20. A method for reducing warp ramp up associated with initializing registers to include full identifiers, the method comprising:

step for storing warp origins separately from warp-specific local identifiers, the warp-specific local identifiers indicating positions of threads within a warp relative to a warp origin;

step for combining the warp-specific local identifiers with the warp origins in real-time to generate full identifiers; and

step for accessing thread data using the full identifiers.

21. The method of claim 20, wherein the step for storing warp origins separately from warp-specific local identifiers includes:

storing the warp origins in one or more per-warp general purpose registers; and

storing the warp-specific local identifiers in a lookup table.

22. The method of claim 21, wherein the step for storing warp origins separately from warp-specific local identifiers further includes:

determining a warp pattern mode; and

selecting the lookup table, from among a plurality of lookup tables, different lookup tables storing identifiers based on different warp pattern modes.

23. The method of claim 20, 21. or 22, wherein the step for combining the warp-specific local identifiers with the warp origins includes:

generating full identifiers by executing an instruction in a single-instruction-multiple-data (SIMD) arithmetic-logical-unit (ALU), the instruction specifying:

at least one per-thread general purpose register as a destination; and

the one or more per-warp registers as a source.

24. A processing system comprising:

a single-instruction-multiple-data (SIMD) arithmetic-logical-unit (ALU) including a plurality of registers;

an instruction decoder coupled to the SIMD ALU;

at least one per-thread register coupled to the SIMD ALU;

at least one per-warp register coupled to the SIMD ALU, the at least one per-warp register including information indicating a warp origin associated with a warp including at least one thread;

memory storing a lookup table including information mapping thread identifiers to warp-specific local identifiers, each warp-specific local identifier indicating a relative position of a thread within the one or more warps; and

the SIMD ALU configured to combine the warp-specific local identifiers with the warp origins to generate final identifiers indicating locations of threads within the address space.

25. The processing system of claim 24, wherein the SIMD ALU is further configured to:

receive an instruction from the instruction decoder, the instruction designating the at least one per-thread register as a destination, and the at least one per-warp register as a source;

retrieve the warp origin of a particular warp from the at least one per-warp register associated with the particular warp;

retrieve, from the lookup table, warp-specific local identifiers of threads included in the particular warp;

combine the warp origin and the warp-specific local identifiers to generate full identifiers specifying thread locations within the address space; and

use the full identifiers to store data in particular per-thread registers corresponding to particular threads included in the particular warp.

26. The processing system of claims 25, wherein the memory stores a plurality of lookup tables associated with different warp patterns.

27. The processing system of claim 26, further comprising:

one or more full or half precision warp local ID registers configured to obtain the warp-specific local identifiers from a table selected according to a warp pattern mode.

28. The method of claim 24, wherein:

the at least one per-warp register stores a warp origin value indicating a starting position of the warp within a workgroup; and

the SIMD ALU generates final local identifiers indicating locations of the thread within the workgroup.

29. The method of claim 24, wherein:

storing the warp origin includes storing a warp origin value indicating a starting position of the warp within a global address space; and wherein

the SIMD ALU generates final global identifiers indicating locations of the thread within the global address space.

30. A non-transitory computer readable medium tangibly embodying a program of instructions to be executed by a processor, the program of instructions including:

at least one instruction, upon execution, causing the processor to retrieve a warp origin of a particular warp from one or more per-warp registers associated with the particular warp;

at least one instruction, upon execution, causing the processor to retrieve, from a lookup table, a warp-specific local identifier of a thread included in the particular warp;

at least one instruction, upon execution, causing the processor to combine the warp origin and the warp-specific local identifiers to generate a full identifier specifying a thread location within an address space; and

at least one instruction, upon execution, causing the processor to use the full identifier to access data from storage.

31. The non-transitory computer readable medium of claim 30, further comprising:

at least one instruction, upon execution, causing the processor to select a particular lookup table from a plurality of different lookup tables based on a warp pattern mode associated with the particular warp.

32. The non-transitory computer readable medium of claims 30 or 31, further comprising:

at least one instruction, upon execution, causing the processor to retrieve the warp-specific local identifier from the lookup table via full or half precision warp local ID registers.

33. The non-transitory computer readable medium of claim 30, wherein:

a single instruction causes the processor to perform all of the following functions:

retrieve a warp origin of a particular warp from one or more particular per-warp registers associated with the particular warp;

retrieve, from a lookup table, a warp-specific local identifier of a thread included in the particular warp;

combine the warp origin and the warp-specific local identifiers to generate a full identifier specifying a thread location within an address space; and

wherein the single instruction:

specifies an operation that causes the processor to combine the warp origin and the warp-specific local identifiers to generate the full identifier;

specifies the per-thread register as a destination; and

specifies the one or more particular per-warp registers as a source to retrieve the warp origin.

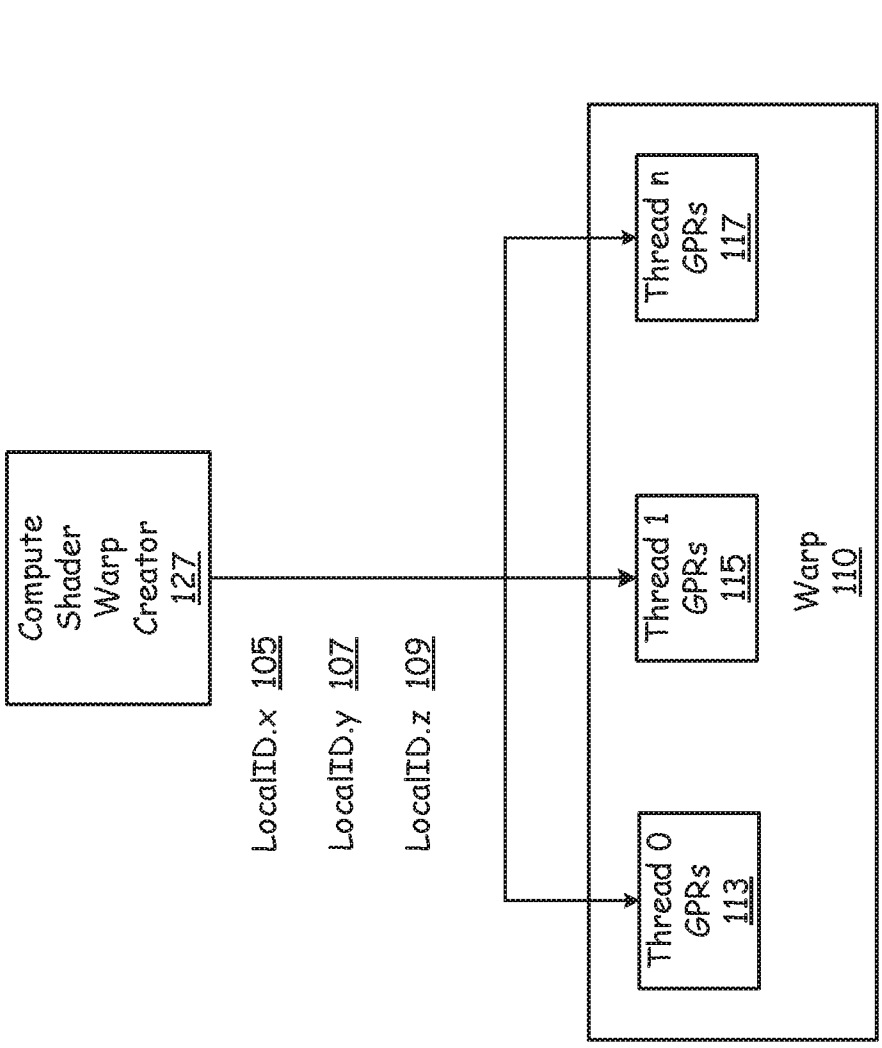


FIG. 1
(Prior Art)

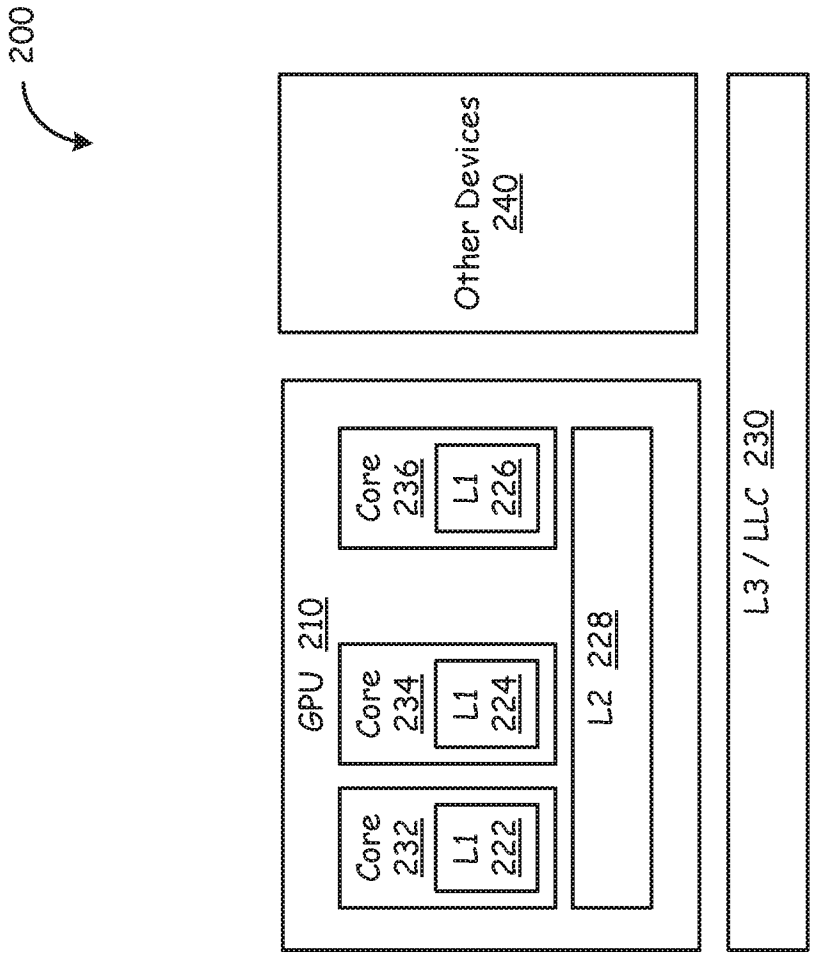

200
Other Devices 240
GPU 210
Core 232
L1 222
Core 234
L1 224
Core 236
L1 226
L2 228
L3 / LLC 230

FIG. 2

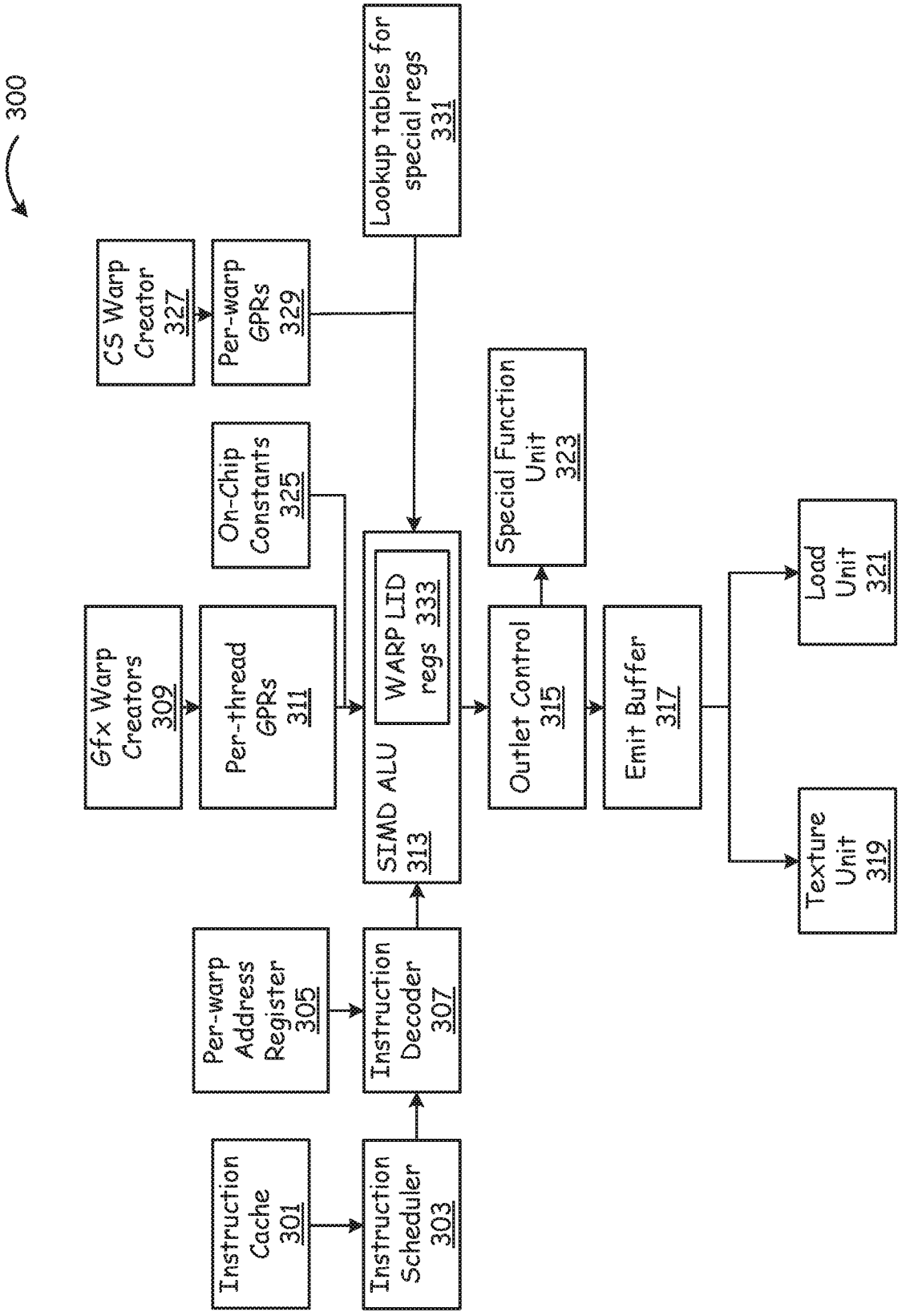

300
Instruction Cache 301
Instruction Scheduler 303
Per-warp Address Register 305
Instruction Decoder 307
Gfx Warp Creators 309
Per-thread GPRs 311
SIMD ALU 313
WARP LID regs 333
On-Chip Constants 325
CS Warp Creator 327
Per-warp GPRs 329
Lookup tables for special regs 331
Outlet Control 315
Special Function Unit 323
Emit Buffer 317
Load Unit 321
Texture Unit 319

FIG. 3

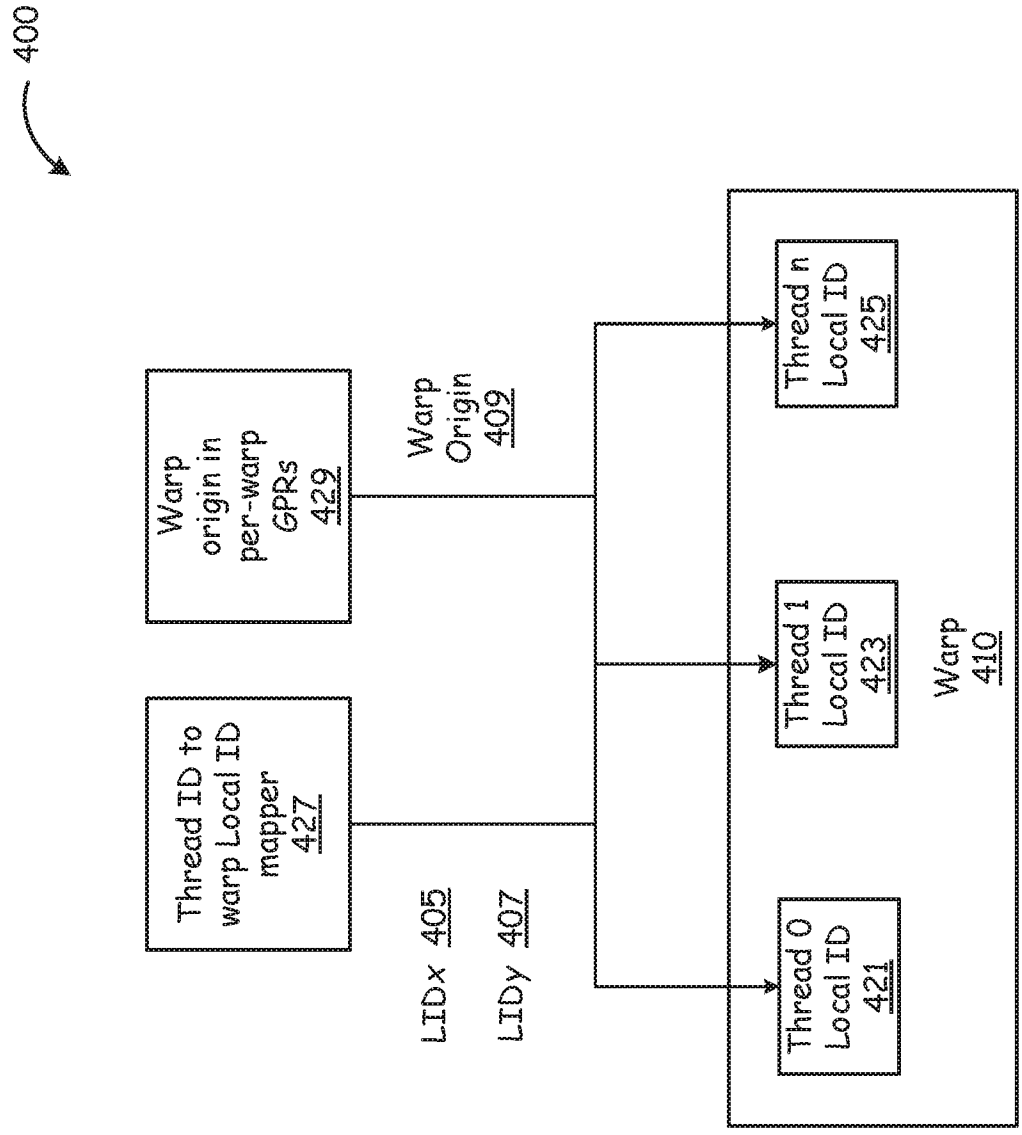
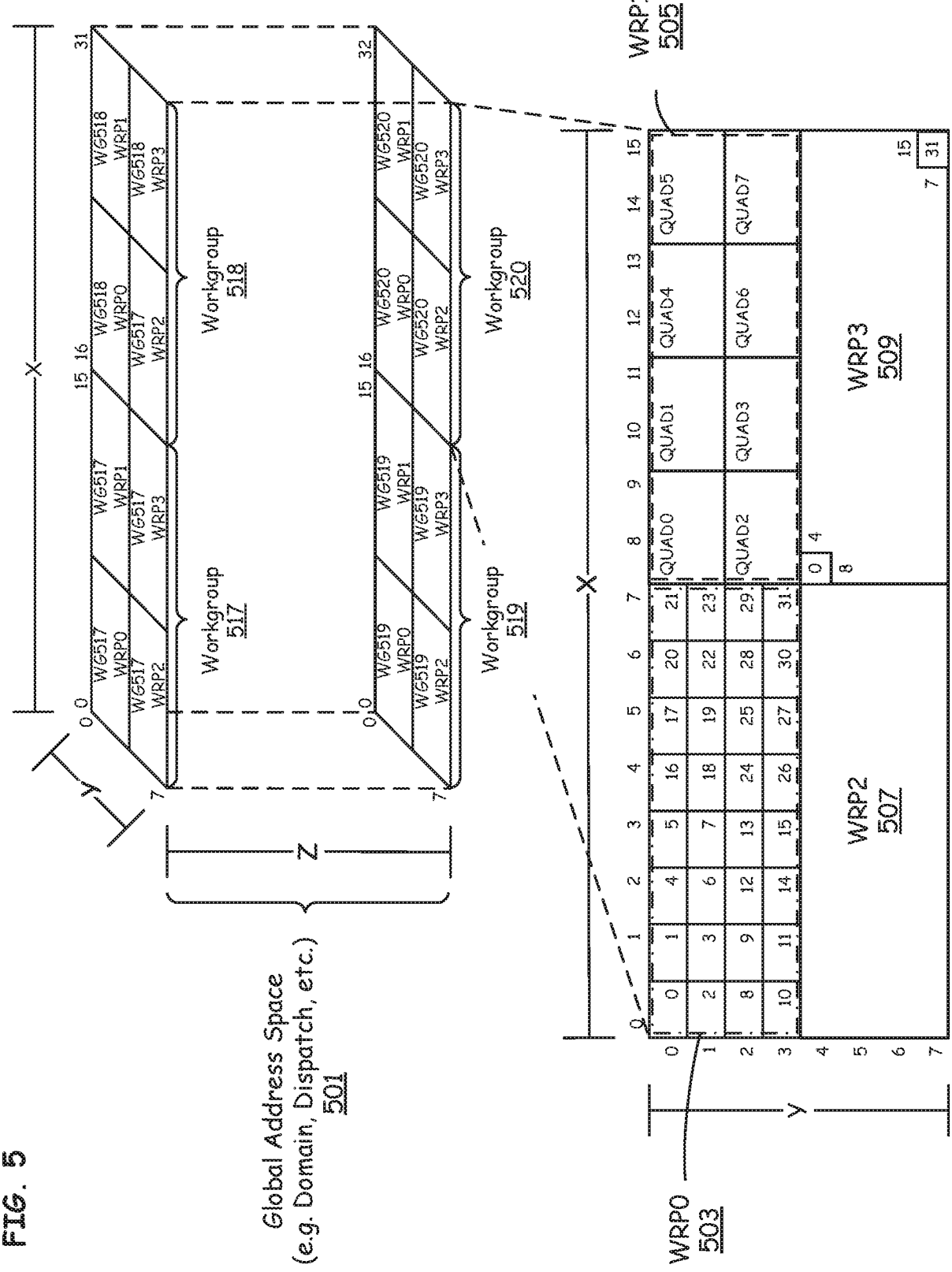

400
Warp origin in per-warp GPRs 429
Warp Origin 409
Thread ID to warp Local ID mapper 427
LIDx 405
LIDy 407
Thread n Local ID 425
Thread 1 Local ID 423
Thread 0 Local ID 421
Warp 410

FIG. 4

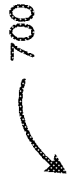
FIG. 5


Global Address Space
(e.g. Domain, Dispatch, etc.)
501
Workgroup
517
Workgroup
518
Workgroup
519
Workgroup
520
WRP0
503
WRP1
505
WRP2
507
WRP3
509
QUAD0
QUAD1
QUAD2
QUAD3
QUAD4
QUAD5
QUAD6
QUAD7
X
Y
Z

600

ThreadID	LIDX/LIDXH	LIDY/LIDYH		ThreadID	LIDX/LIDXH	LIDY/LIDYH
0	0	0		16	16	0
1	1	0		17	17	0
2	2	0		18	18	0
3	3	0		19	19	0
4	4	0		20	20	0
5	5	0		21	21	0
6	6	0		22	22	0
7	7	0		23	23	0
8	8	0		24	24	0
9	9	0		25	25	0
10	10	0		26	26	0
11	11	0		27	27	0
12	12	0		28	28	0
13	13	0		29	29	0
14	14	0		30	30	0
15	15	0		31	31	0

FIG. 6



ThreadID	LIDX/LIDXH	LIDY/LIDYH		ThreadID	LIDX/LIDXH	LIDY/LIDYH
0	0	0		16	4	0
1	1	0		17	5	0
2	0	1		18	4	1
3	1	1		19	5	1
4	2	0		20	6	0
5	3	0		21	7	0
6	2	1		22	6	1
7	3	1		23	7	1
8	0	2		24	4	2
9	1	2		25	5	2
10	0	3		26	4	3
11	1	3		27	5	3
12	2	2		28	6	2
13	3	2		29	7	2
14	2	3		30	6	3
15	3	3		31	7	3

FIG. 7

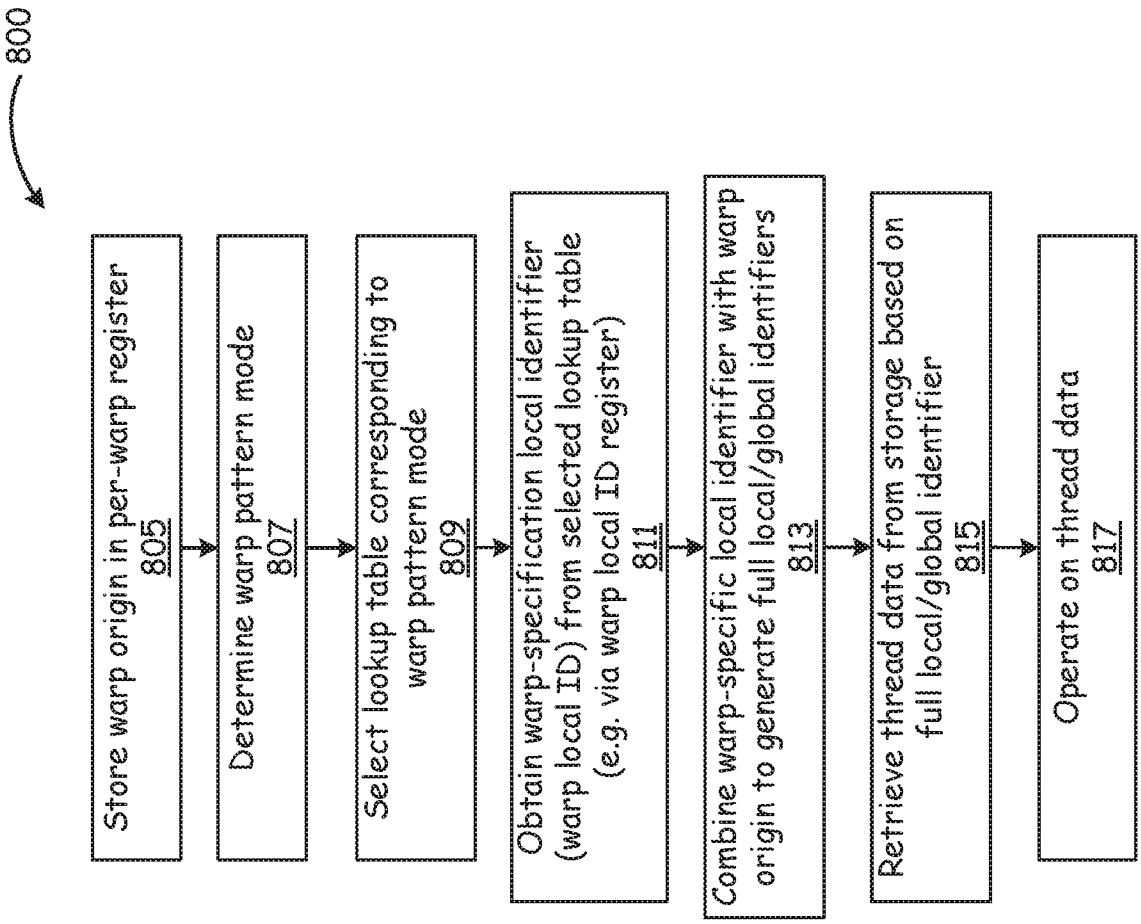

800
Store warp origin in per-warp register
805
Determine warp pattern mode
807
Select lookup table corresponding to warp pattern mode
809
Obtain warp-specification local identifier (warp local ID) from selected lookup table (e.g. via warp local ID register)
811
Combine warp-specific local identifier with warp origin to generate full local/global identifiers
813
Retrieve thread data from storage based on full local/global identifier
815
Operate on thread data
817

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2019/090181

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/46(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

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

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

CNPAT, WPI, EPODOC, ISI, CNKI: thread, group, warp, ID, identifier, address, position, local, origin, start, begin

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2011078689 A1 (SHEBANOW, Michael C. et al.) 31 March 2011 (2011-03-31) abstract, description, paragraphs [0067]-[0089]	1-33
A	CN 106648545 A (TIANJIN UNIVERSITY) 10 May 2017 (2017-05-10) the whole document	1-33
A	CN 105579967 A (QUALCOMM INCORPORATED) 11 May 2016 (2016-05-11) the whole document	1-33
A	US 2014164737 A1 (KALRAY) 12 June 2014 (2014-06-12) the whole document	1-33
A	US 2011078358 A1 (SHEBANOW, Michael C.) 31 March 2011 (2011-03-31) the whole document	1-33

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
02 December 2019	**19 December 2019**

Name and mailing address of the ISA/CN	Authorized officer
National Intellectual Property Administration, PRC **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing** **100088** **China**	**LI,Na**
Facsimile No. **(86-10)62019451**	Telephone No. **86-(10)-53961403**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2019/090181

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2011078689 A1	31 March 2011	None	
CN 106648545 A	10 May 2017	None	
CN 105579967 A	11 May 2016	EP 3053038 A1	10 August 2016
		KR 20160065121 A	08 June 2016
		WO 2015050681 A1	09 April 2015
		US 2015095914 A1	02 April 2015
		JP 2016532180 A	13 October 2016
US 2014164737 A1	12 June 2014	None	
US 2011078358 A1	31 March 2011	None	