



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

(21) International Application Number:
PCT/CN2011/082215

(22) International Filing Date:
15 November 2011 (15.11.2011)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant (for all designated States except US): **INTEL CORPORATION** [US/US]; 2200 Mission College Blvd., M/S: RNB4-150, Santa Clara, California 95052 (US).

(72) Inventors; and

(75) Inventors/Applicants (for US only): **CHEN, Feng** [CN/CN]; Room 403, Building 21 1811 Avenue, Zhang Yang Road, Pudong, Shanghai 200135 (CN). **HAO, Yan** [CN/CN]; No. 880 Zi Xing Road, Shanghai Zizhu Science Park, Shanghai 200241 (CN). **FU, Jin** [CN/CN]; Room 702, No. 18, Zhenyue Road, Yangpu District, Room 502, South-14, No. 800, Dongchuan Road, Shanghai 200438 (CN).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, 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, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

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

(54) Title: SCHEDULING THREAD EXECUTION BASED ON THREAD AFFINITY

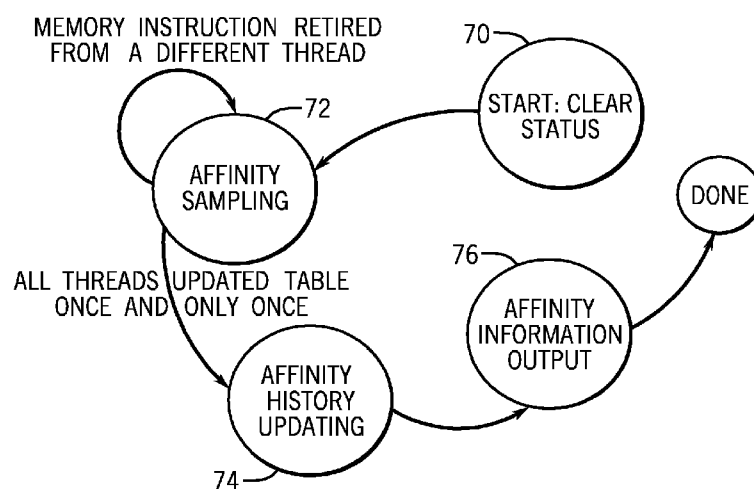


FIG. 5

(57) Abstract: In accordance with some embodiments, spatial and temporal locality between threads executing on graphics processing units may be analyzed and tracked in order to improve performance. In some applications where a large number of threads are executed and those threads use common resources such as common data, affinity tracking may be used to improve performance by reducing the cache miss rate and to more effectively use relatively small-sized caches.

SCHEDULING THREAD EXECUTION BASED ON THREAD AFFINITY

Background

[0001] This relates generally graphics processing units.

[0002] In many computing platforms, separate central processing and graphics processing units may be used. Graphics processing tasks may be offloaded from central processors to graphics processors. Graphics processors may perform a number of specialized tasks including graphics tasks and financial analysis tasks. Typically where heavy mathematical operations are involved, tasks may be offloaded to the graphics processor.

[0003] For example, in financial analysis, a Monte Carlo simulation may be performed. A Monte Carlo simulation predicts option prices based on average prices and variances of stock prices. Graphics processors may be used to do these simulations using thousands of threads. Generally the sequence of thread execution is not optimized and is simply taken in a thread by thread execution order.

Brief Description of the Drawings

Figure 1 is a depiction of a graphics processing unit architecture in accordance with one embodiment;

Figure 2 is a schematic depiction of an affinity history buffer for use in the embodiment of Figure 1;

Figure 3 is a schematic depiction of an affinity sampling table for use in connection with the embodiment of Figure 1;

Figure 4 is a more detailed depiction of an affinity tracking buffer;

Figure 5 is a state diagram for affinity detection in accordance with one embodiment;

Figure 6 is a schematic depiction of affinity sampling in accordance with one embodiment;

Figure 7 is a flow chart for affinity sampling in accordance with one embodiment;

Figure 8 is a depiction of affinity sampling states in accordance with one embodiment;

Figure 9 is a depiction of affinity history buffer updating in accordance with one embodiment;

Figure 10 is a flow chart for affinity history updating in accordance with one embodiment;

Figure 11 is a schematic depiction of affinity information output in accordance with one embodiment;

Figure 12 is a flow chart for affinity information output in accordance with one embodiment.

Detailed Description

[0004] In accordance with some embodiments, spatial and temporal locality between threads executing on graphics processing units may be analyzed and tracked in order to improve performance. In some applications where a large number of threads are executed and those threads use common resources such as common data, affinity tracking may be used to improve performance by reducing the cache miss rate and to more effectively use relatively small-sized caches.

[0005] In some embodiments, a task may be broken into a large number of steps and the task may be repeated for a number of different cases. By calculating the same step across a number of cases, each executed by different groups of threads, affinity between the thread groups may be used to improve performance.

[0006] One possible application of affinity analysis is graphics processing for a Monte Carlo simulation using thousands of threads. A large number of threads such as 128 or 256 threads may be used to calculate each stock option price. A large number of options may be analyzed, for example 10,000 option prices may be calculated at a time. However, different options may use the same state inputs that provide a random number. Thus an opportunity for affinity based performance improvements may be realized because different groups of threads may share the

use of a common resource, in this case, a state input. Since graphics processing unit cache size may be relatively small compared to that of central processing units, affinity analysis may be particularly amenable to improving performance in graphics units.

[0007] Thus instead of calculating the stock option price, one option after the other, the option price may be calculated by breaking the calculation into a plurality of steps and performing one step across a number of options. Calculating the option price one after the other may be inefficient because two groups of threads may access the same data set. Thus through the use of locality identification and exploitation across thread groups, performance may be improved, when breaking up the calculation into a plurality of steps and calculating the same step for a large number of options.

[0008] More specifically, the sequence of thread execution may be reordered or rescheduled to take advantage of locality between threads. Again looking at the example of a Monte Carlo simulation, the simulation calculation may be broken into about one hundred steps. For one example, the first step may be done for a first option and then the calculation may be suspended, followed by doing the first step for a second option and then suspending it and so on. This may reduce the amount of information in the caches so that spatial locality may be used advantageously across thread groups.

[0009] As a result cache miss rates may be reduced in some embodiments. Thus by identifying localities across groups of threads, performance benefits may be achieved by scheduling the order of execution of threads to take advantage of locality between thread groups. Basically, thread groups that have locality may be executed sequentially.

[0010] The same locality benefits may arise in a number of graphics processing tasks, including those generally associated with pixel shaders, because they require different texture resources at different parts of the calculations.

[0011] Referring to Figure 1, in accordance with a graphics processing unit throughput computing architecture 10, an active thread pool 12 may include a number of threads for one single vector execution unit 13. In other words, each vector execution unit 13 may include an active thread pool 12. Thus each vector execution unit may, in one embodiment, be used to determine an option price for one particular stock.

[0012] In accordance with some embodiments of the present invention, each vector execution unit may include a scheduler that includes an affinity history buffer 19. The affinity history buffer may be used for tracking affinity between threads. The scheduler can use this information to execute threads in an order that takes advantage of any detected affinity.

[0013] Each vector execution unit includes a data level one cache 14, which in turn is connected to any number of higher level caches including a last level cache (LLC) 16. A memory controller 18 may be coupled to the data caches.

[0014] As used herein, “resource access affinity” or “affinity” is the result of two different threads using substantially the same resources so that performance advantages may be obtained by scheduling the threads that have affinity to be sequential or substantially sequential. As a result of breaking computing tasks up and executing them on different threads, temporal and spatial locality may be lost. By analyzing spatial and temporal locality, threads may be rescheduled to execute sequentially when they have affinity. This may result in improving the cache hit ratio and memory request coalescing efficiency. An affinity history buffer detects and tracks dominant resource access affinity.

[0015] Thus referring to Figure 2, a more detailed depiction of the affinity history buffer 19 shown in Figure 1 is provided. Thread lane identifier 20 feeds thread lane identifiers to global thread lane bitmaps 28 and local thread lane bitmaps 32. Memory load address buffer 22 provides a load address to a memory shift register 26. The memory shift register 26 shifts the memory address in some regular way in order to find memory addresses that have affinity. The output of the thread lane

bitmap 28 is provided to a memory slice index 30 that also receives the address shift. The output is a local thread lane bitmap 32 that forms an affinity sampling table 24.

[0016] The output from the local thread lane bitmap 32 is provided as an affinity entry 36. The affinity entry may be provided to a thread assigned status register 38 and ultimately to the thread scheduler or resource allocation logic 40 of the vector execution unit 13 shown in Figure 1.

[0017] The memory address shift logic 26 shown in Figures 2 and 3 may be a configurable bit-wise shift register. The number of bytes to be shifted and the shift direction may be configured by a driver 22 (Figure 3). The driver sets the number of bits based on available cache resource size and workload access granularity. Its input is a virtual address of a memory load operation (blocks 44 and 46 of Figure 3) from the memory load address 22 (Figure 3). Its output is called a memory slice index 30.

[0018] Actually, the memory address shift logic may use other bit-wise operations rather than a normal shift. For example, for two-dimensional based texture access, a right shifting operation may be adjusted based on different tiling algorithms. If threads with the same memory slice index, there is space and/or time locality among those threads. The number of bits to be shifted determines the tolerance of locality.

[0019] The global thread lane bitmap 28 is a register whose bit number is the same as the active thread pool size. For a given kernel, if the graphic processing unit hardware's resource could maintain forty threads, the register may be five bytes in length. If the maximum thread number is thirty two, its length is one word. Its bits are used as a bitmap to map each hardware thread in a pool. For example, the first thread in a pool is mapped to a less significant bit. If one bit of the global thread lane bitmap is set, it stands for a corresponding thread being passed through affinity sampling procedure and vice versa.

[0020] The affinity sampling table 24 (Figure 3) is a two-dimensional table. For each entry in the table, its index field is a memory slice index. Its value field is a local thread lane bitmap. The memory slice index comes from the output of the

memory address shift logic 26. The local thread lane bitmap is a bitmap treated with the same semantics as the global thread lane bitmap 28. If one bit of the local thread lane bitmap is set to one, this means that the corresponding hardware thread's memory slice index equals to the entry's index field. So threads embodying space and/or time locality may be clustered to the same entry. The value of the local thread lane bitmap may be interpreted as an affinity identifier as well. For N elements in the active thread pool, a valid value scope of affinity identifier is from zero to 2^N .

[0021] The affinity tracking buffer 34 includes two components. A first component including the affinity entries 36. This is a buffer designed to track hot affinity identifiers through the access history. The second component is a thread assigned status register 38. The input to this subsystem is the affinity identifier and the output is the hot affinity identifier.

[0022] The affinity tracking buffer 34 is the core unit of this subsystem. An enlarged view of the affinity tracking buffer is shown in Figure 4. It is a fixed size two-dimensional table. Each entry in the tracking buffer contains one index and two value fields. The content of the index field is the affinity identifier 66. The first value field is a hit counter that records the appearance of a given affinity identifier 66 across a series of sampling. Its initial value is zero if the affinity identifier is inserted into the table for the first time. The second value is the previous slot ("PREV SLOT").

[0023] The content of the previous slot represents a slot in the affinity tracking buffer. For an N slot affinity tracking buffer design, the previous slot's value scope is from one to N.

[0024] The previous slot is used to set up an ordered linked list in the affinity tracking buffer. That list is then used to maintain a two-level priority queue for entry replacement and output. The first level of priority in this queue is a hit rate. The entry with the highest value in the hit counter will be least likely to be replaced and most likely to be outputted to the outside system. The second level of priority is the lifetime of the entry. For entries with the same hit count value, the youngest entry will be the least likely to be replaced and most likely to be outputted to the outside

system. The previous slot value points to the next hot entry in this linked list. To speed up the affinity information output, the tail of this linked list is maintained. It points to the hottest affinity group index.

[0025] Referring to Figure 5, a state diagram for affinity detection with the affinity buffer is illustrated. Each iteration includes a start state 70, an affinity sampling state 72 where a memory instruction may be retired from a different thread, an affinity history updating state 74 that receives all threads updated table once and only once and affinity state output state as well as a stop state. The start state is a start point of the state machine. The content in the affinity sampling table is local to the iteration and will be set to zero in the start state. That action includes activities of making bits, flushing to zero for the affinity sample table and the global thread lane bitmap. In the affinity sampling state 72, the affinity sampling subsystem clusters affinity identifiers for the active thread pool. For each thread in the pool, the thread is sampled once and only once. The affinity history updating state 74 identifies and tracks a dominant affinity identifier of the current application. Its content is preserved across iterations. The state may be switched to an affinity information output stage 76 during which the hot affinity identifier is outputted.

[0026] The affinity sampling state 72 is shown in more detail in Figure 6. Using the example where the thread pool has M elements, thread zero is the first thread being sampled, the number "1" is shifted to the left zero times and the result would be to do a bit-wise AND operation with the value of the global thread lane bitmap. The result is zero which stands for thread zero is valid for sampling. The thread identifier and virtual address of the load operation is fed into the subsystem. Assuming a granularity of 4K bytes for identity affinity access, address shifting logic is configured to shift 12 bits. The access address for thread zero is 0x0010f00c. The shifting result is 0x0010f. The subsystem search is through the affinity sampling table and finds a zero entry. A new entry is allocated for 0x0010f and is set to the corresponding bit of thread zero in the local thread lane bitmap field. Also, the corresponding bit in the global thread lane bitmap is set. Similar steps are carried out for the next thread, thread 1 which is illustrated in Figure 6, Step 2.

[0027] Referring to the sequence 90 shown in Figure 7, the sequence may be implemented in hardware, software and/or firmware. In software and firmware embodiments it may be implemented by computer readable instructions stored in a non-transitory computer readable medium.

[0028] The sequence begins in block 92 when a memory instruction is retired. Then at block 94 a tuple of the thread identifier and the memory address is created. A check at diamond 96 determines whether the logical AND result of the tuple is equal to zero. If so, the flow ends. Otherwise at block 98, the address is mapped to a memory slice index by a mapping function.

[0029] Then at block 100, the current memory slice index is searched in the affinity sampling table. If the entry exists as determined in diamond 102, then the flow goes to step 106. Otherwise a new entry is allocated for the current memory slice index at block 104. In block 106, a local thread lane bitmap field is updated with the result of a logic or operation between its old value and the tuple. Then in block 108 the global thread bitmap is updated with the result of a logic or operation between its old value and the tuple.

[0030] When affinity sampling is done, there are three possible statuses in the affinity sampling table, as shown in Figure 8. For category 1 there is only one entry in the sampling table and it is called strong affinity. For category 2, there is at least one entry whose local thread lane bitmap is set more than once and is called significant affinity. In category 3, the entry number equals to the active thread pool size and this is called no affinity. For strong and significant affinity categories, a subsystem outputs a sampling result to the affinity tracking subsystem.

[0031] The affinity tracking buffer update corresponding to state 74 in Figure 5 works like a cache. The affinity tracking buffer tracks hot affinity identifiers. Assuming there are four slots in the affinity tracking buffer before the first buffer updating takes place, there is no entry in the buffer, and all four slots are available for insert. After the first affinity sampling is done, the affinity identifier 0x3800 and 0xf is identified and filled into slots one and two. In the second affinity sampling stage the affinity identifier 0xAADA and 0x5525 is found and filled. From then on,

there is no empty slot. The tail point to slot 4, is the most recently visited. The status of the tracking buffer is shown as status 1 in Figure 9.

[0032] When the sampling table is ready for output for a third time, 0xAADA and 0x5525 is fed and the corresponding hit counters are increased which is shown in status 2 of Figure 9. When the fourth sampling result is ready (identifiers 0x360 and 0x7) there is no empty slot available. The buffer eliminates the oldest entry with the lowest hit count, which in this case 0x3800 and 0xF in slots 1 and 2. The status is shown as status 3 in Figure 9. The detailed control flow is shown in Figure 10.

[0033] Figure 10 illustrates a sequence which may be implemented in hardware, software, and/or firmware. In software and firmware embodiments, it may be implemented by computer readable instructions stored in a non-transitory computer readable medium such as an optical, magnetic or semiconductor memory.

[0034] In order to update the affinity history buffer, the flow starts by determining whether there were more entries in affinity sampling table at diamond 110. If so, the corresponding local thread lane bitmap of that entry is obtained as indicated in block 112. The number of ones in the bitmap is counted in block 114. If that number is greater than one as determined in diamond 116, the value of the bitmap in the affinity history buffer is searched in block 118.

[0035] A check at diamond 120 finds the corresponding entry. If so, the hit count is increased in block 122. If not, a check at diamond 124 determines there is an empty slot available. If so, an affinity entry record is created in the target slot as indicated in block 128. Otherwise a slot is found for replacing based on the least recently used entry (block 124).

[0036] The affinity tracking buffer help puts affinity information if needed. The index field of each entry describes an affinity thread group and a bitmap. An example of the affinity information output is shown in Figure 11.

[0037] Before the output starts, the thread assigns status register is flush to zero. The affinity tracking buffer outputs the entry 0xAADA as a first candidate pointed to by the tail. The 0xAADA does a bit-wise and operation against a thread assigned

status register. The result is zero meaning that all threads in the bitmap of 0xAADA are valid. The thread assigned status register is updated with the result of the bit-wise OR operation between the old value and 0xAADA. Then 0x5525 is outputted in the same way. The output procedure stops on the third candidate due to one of its threads being already outputted. A detailed chart is shown in Figure 12.

[0038] Referring to Figure 12, an affinity information output sequence may be implemented in software, firmware and/or hardware. In software and firmware embodiments it may be implemented by computer executed instruction stored in a non-transitory computer readable medium such as a semiconductor, optical or magnetic memory.

[0039] The sequence starts at block 130 by doing a bit-wise AND operation between the current affinity identifier and the thread assigned status register. If the result is equal to zero as determined in diamond 132, the affinity identifier is output in block 134. In block 136 the thread assigned status register value is updated with the result of the bit-wise OR operation between its old value and the current affinity identifier. Then in block 138 the current entry is replaced with the one pointed to by previous slot.

[0040] The graphics processing techniques described herein may be implemented in various hardware architectures. For example, graphics functionality may be integrated within a chipset. Alternatively, a discrete graphics processor may be used. As still another embodiment, the graphics functions may be implemented by a general purpose processor, including a multicore processor.

[0041] References throughout this specification to “one embodiment” or “an embodiment” mean that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one implementation encompassed within the present invention. Thus, appearances of the phrase “one embodiment” or “in an embodiment” are not necessarily referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be instituted in other suitable forms other than the particular embodiment illustrated and all such forms may be encompassed within the claims of the present application.

[0042] While the present invention has been described with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.

What is claimed is:

- 1 1. A method comprising:
2 breaking a repetitive graphics processing task into steps;
3 performing the same step across a number of said repetitive tasks;
4 identifying affinities between tasks; and
5 ordering threads performing said tasks to take advantage of affinity between
6 tasks.
- 1 2. The method of claim 1 including using address shift logic to identify tasks that
2 may have affinity.
- 1 3. The method of claim 2 including using a driver to configure the address shift
2 logic, said driver to set the number of bits to be shifted based on available cache size,
3 said shift logic to output a memory slice index.
- 1 4. The method of claim 3 including denominating threads with the same memory
2 slice index as having affinity.
- 1 5. The method of claim 4 including setting up a two-dimensional affinity sampling
2 table whose index field is the memory slice index and whose value field is a local
3 thread lane bitmap that indicates whether the memory slice index equals an entry's
4 index field.
- 1 6. The method of claim 5 including clustering threads with affinity in one entry in
2 the table.
- 1 7. The method of claim 1 including tracking affinity in a buffer that establishes an
2 ordered linked list to maintain a two-level priority queue for entry replacement.

- 1 8. A non-transitory computer readable medium storing instructions to enable a
2 computer to:
3 break a repetitive graphics processing task into steps;
4 perform the same step across a number of said repetitive tasks;
5 identify affinities between tasks; and
6 order threads performing said tasks to take advantage of affinity between
7 tasks.
- 1 9. The medium of claim 8 further storing instructions to use address shift logic to
2 identify tasks that may have affinity.
- 1 10. The medium of claim 9 further storing instructions to configure the address
2 logic, said driver to set the number of bits to be shifted based on available cache size,
3 said shift logic to output a memory slice index.
- 1 11. The medium of claim 10 further storing instructions to denominate threads
2 with the same memory slice index as having affinity.
- 1 12. The medium of claim 11 further storing instructions to set up a two-
2 dimensional affinity sampling table whose index field is the memory slice index and
3 whose value field is a local thread lane bitmap that indicates whether the memory
4 slice index equals an entry's index field.
- 1 13. The medium of claim 12 further storing instructions to address cluster threads
2 with affinity in one entry in the table.
- 1 14. The medium of claim 8 further storing instructions to track affinity in a buffer
2 that establishes an ordered linked list to maintain a two-level priority queue for entry
3 replacement.

- 1 15. An apparatus comprising:
2 a controller to break a repetitive graphics processing task into steps, perform
3 the same step across a number of said repetitive tasks, identify affinities between
4 tasks, and order threads performing said tasks to take advantage of affinity between
5 tasks; and
6 a memory coupled to said controller.
- 1 16. The apparatus of claim 15 to said controller to use address shift logic to
2 identify tasks that may have affinity.
- 1 17. The apparatus of claim 16 including a driver to configure the address shift
2 logic, said driver to set the number of bits to be shifted based on available cache size,
3 and shift logic to output a memory slice index.
- 1 18. The apparatus of claim 17 said controller to denominate threads with the
2 same memory slice index as having affinity.
- 1 19. The apparatus of claim 18 said controller to set up a two-dimensional affinity
2 sampling table whose index field is the memory slice index and whose value field is
3 a local thread lane bitmap that indicates whether the memory slice index equals an
4 entry's index field.
- 1 20. The apparatus of claim 19 said controller to cluster threads with affinity in one
2 entry in the table.
- 1 21. The apparatus of claim 15 said controller to track affinity in a buffer that
2 establishes an ordered linked list to maintain a two-level priority queue for entry
3 replacement.

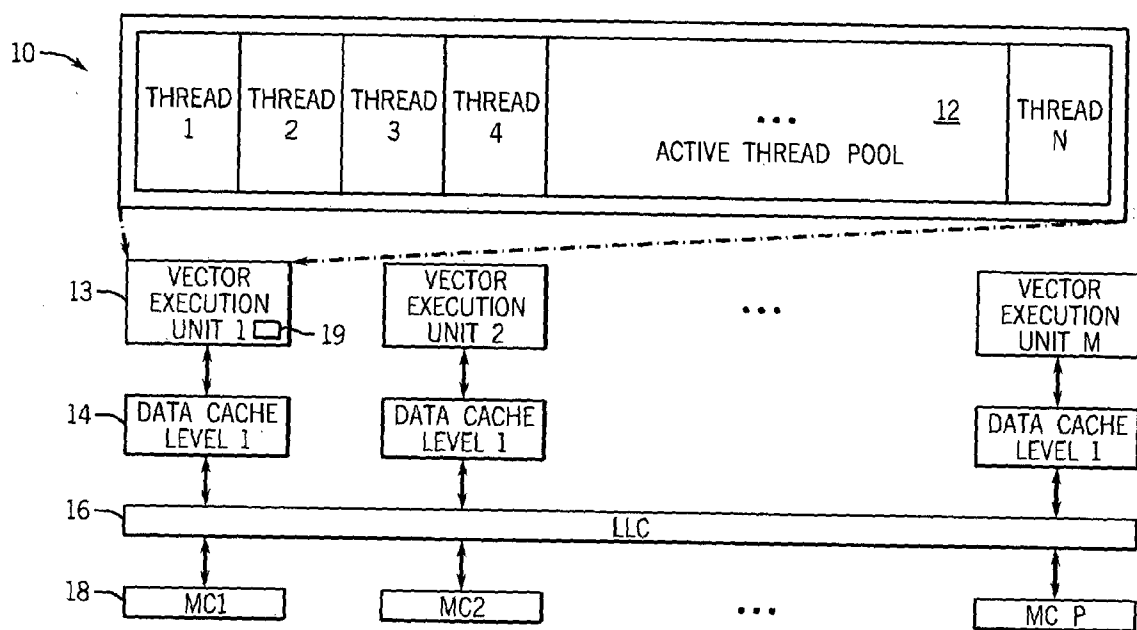


FIG. 1

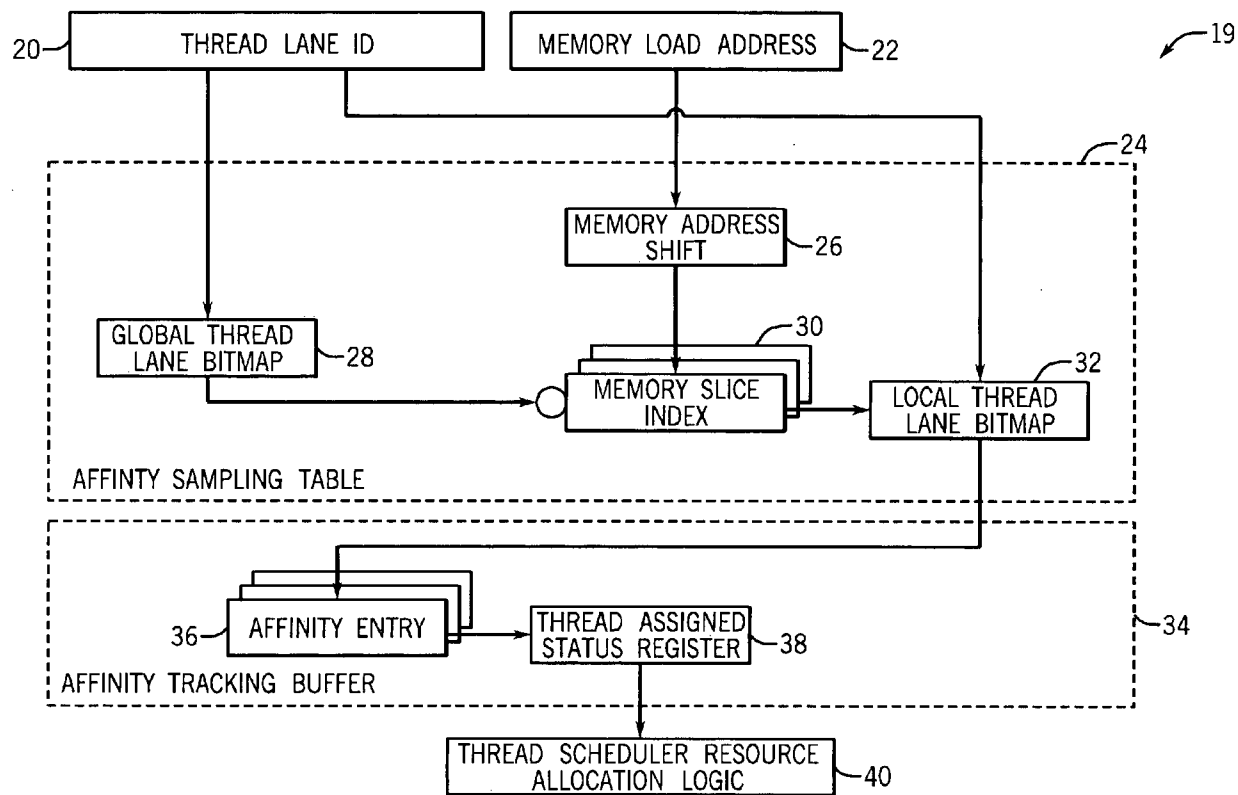


FIG. 2

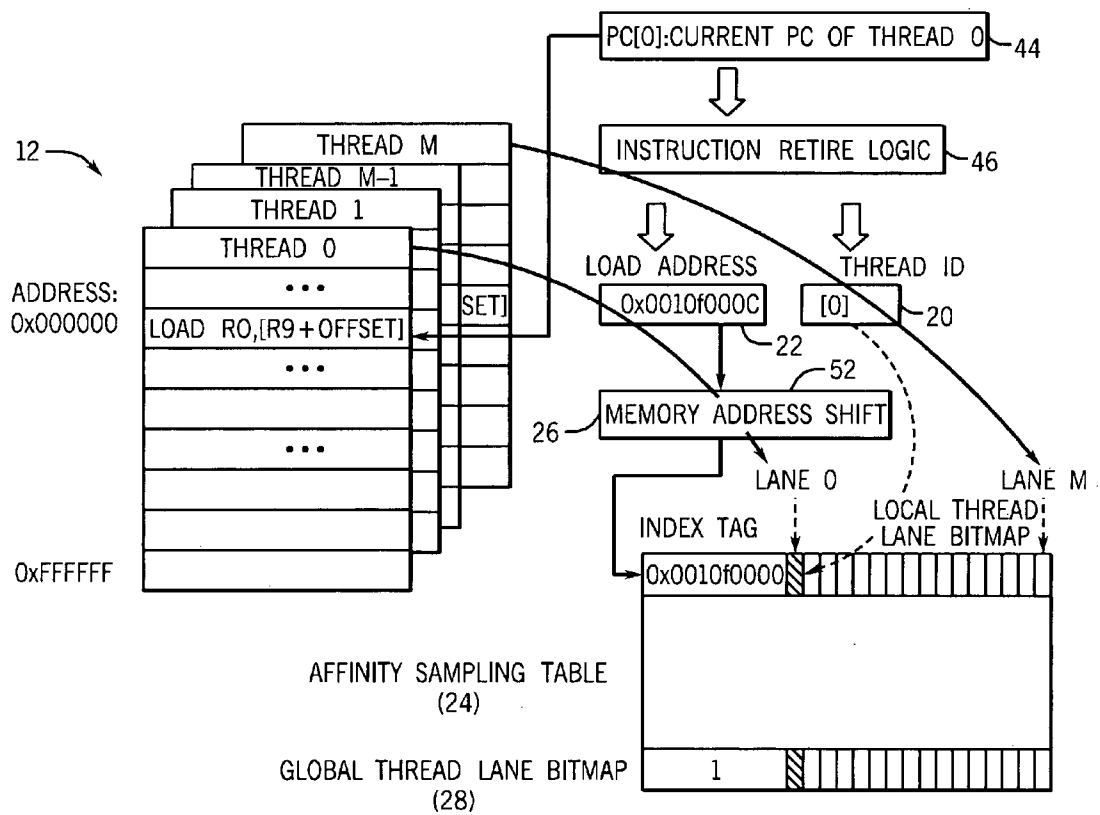


FIG. 3

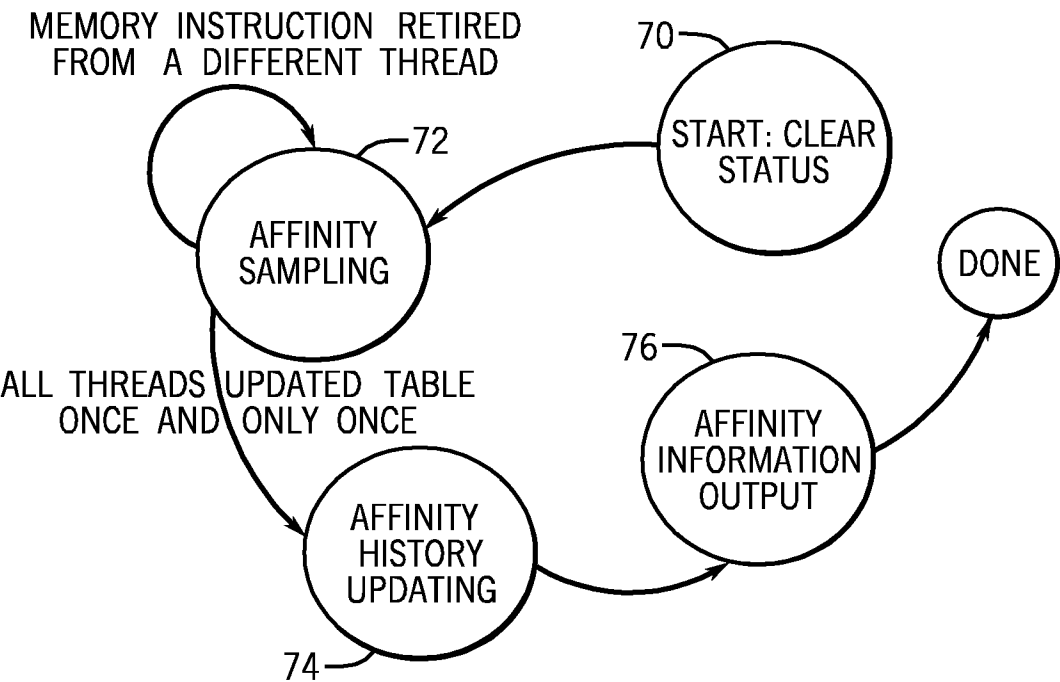
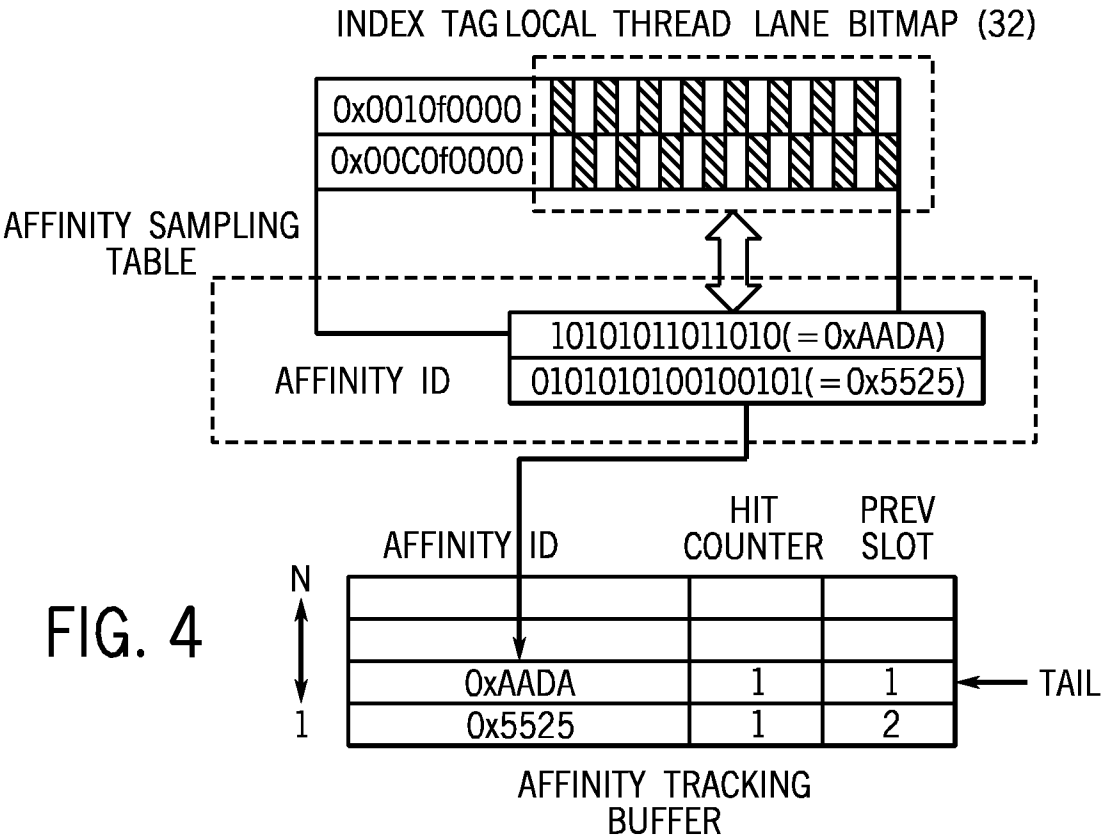


FIG. 5

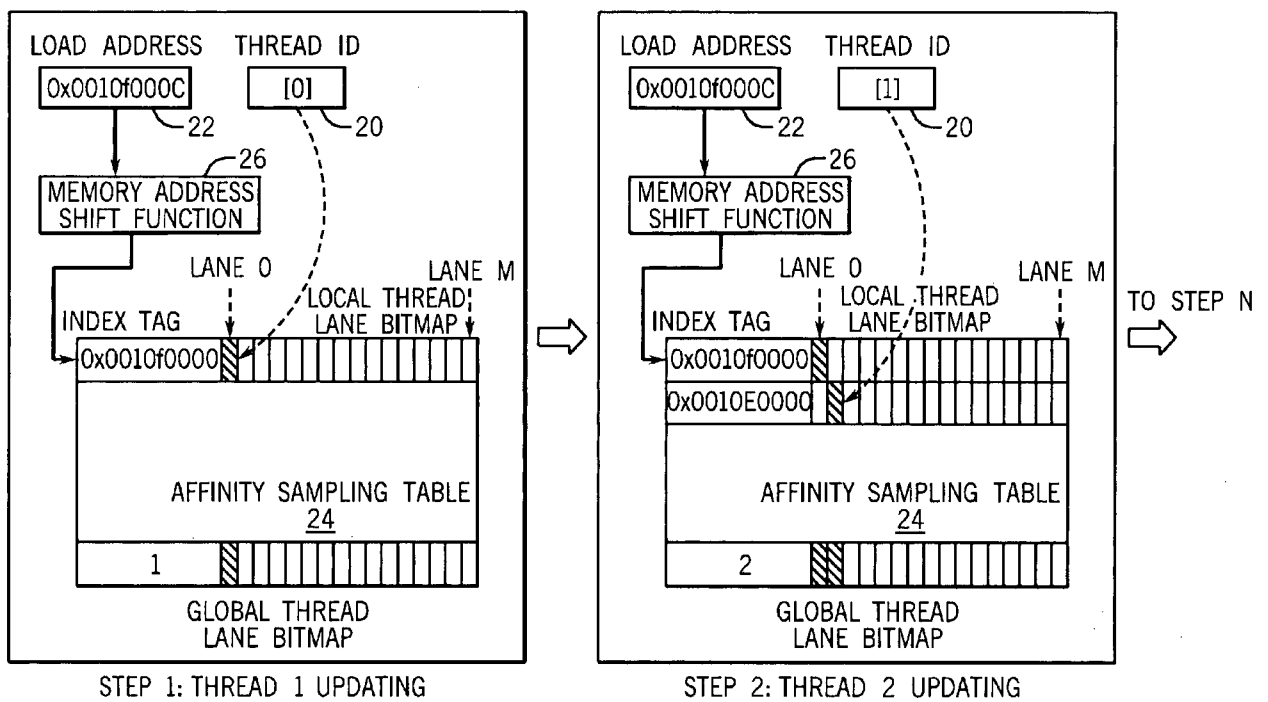


FIG. 6

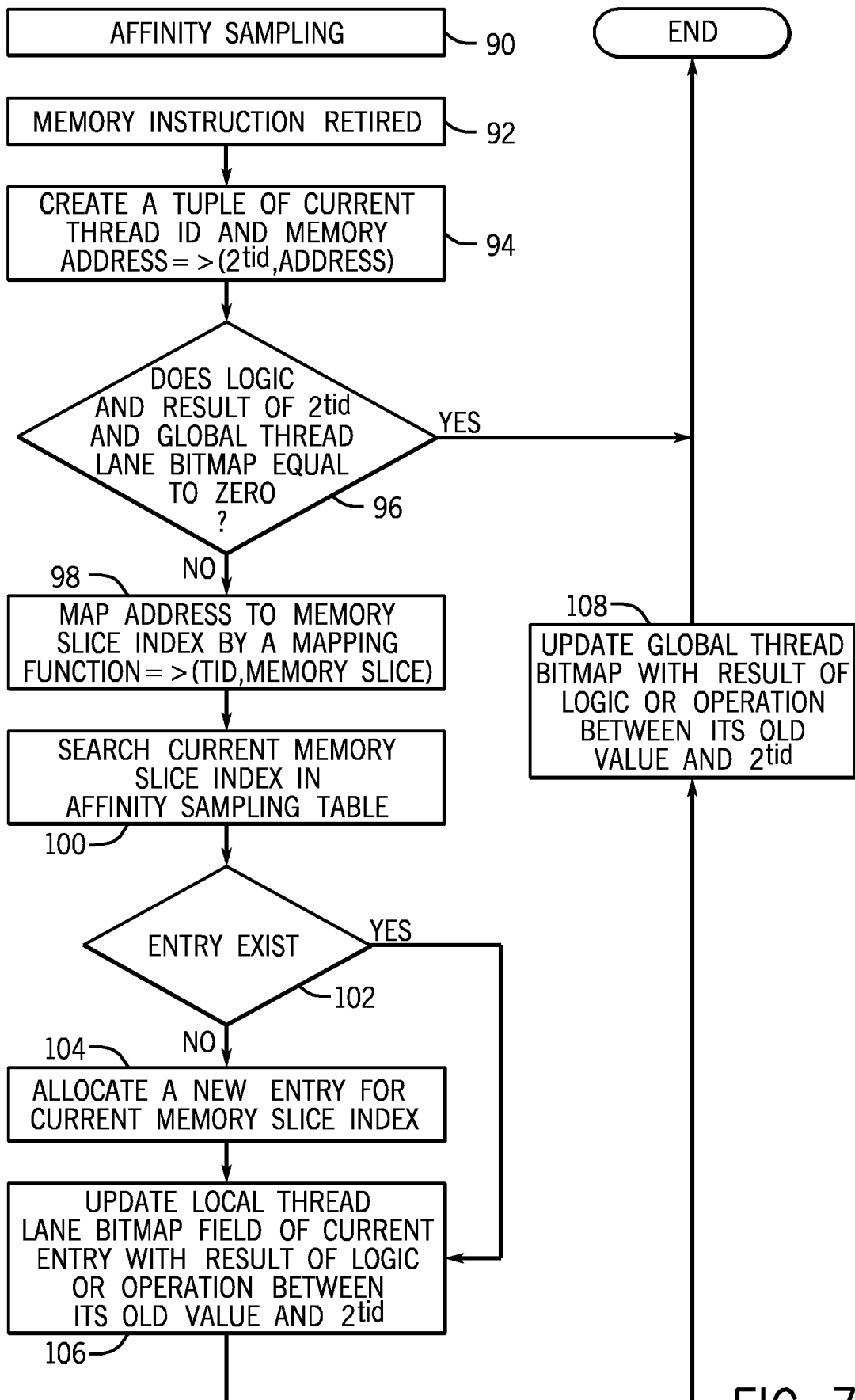


FIG. 7

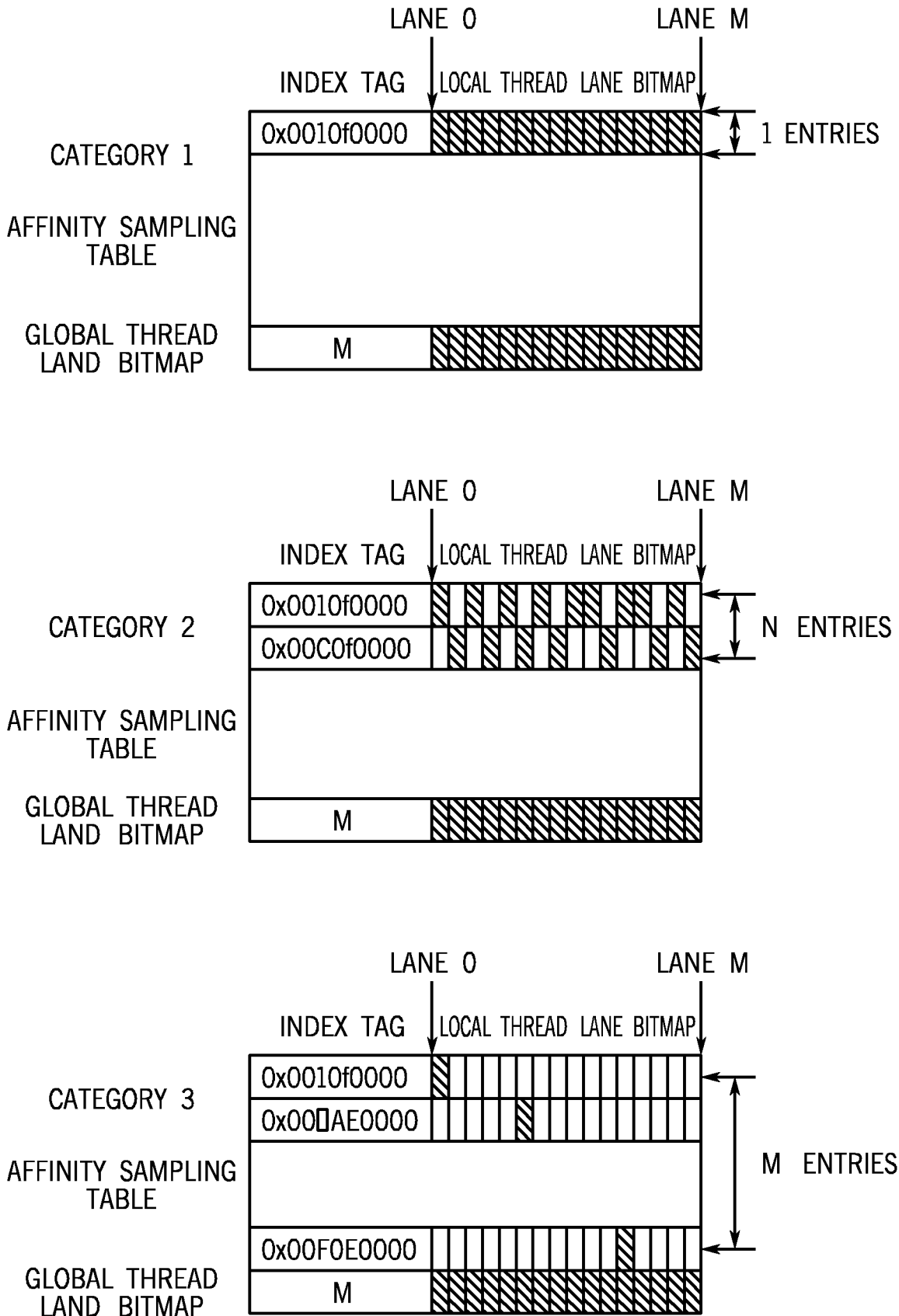


FIG. 8

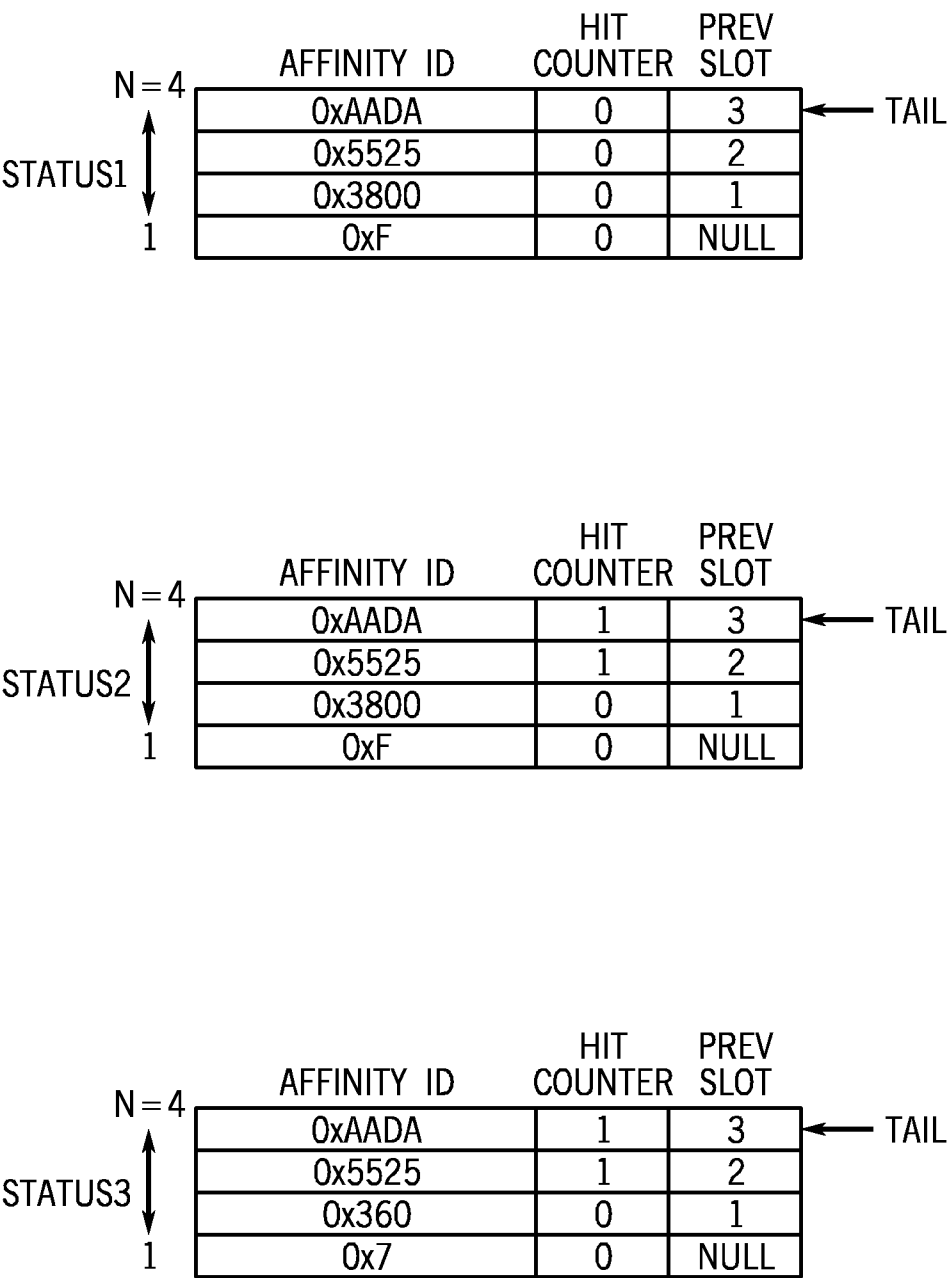


FIG. 9

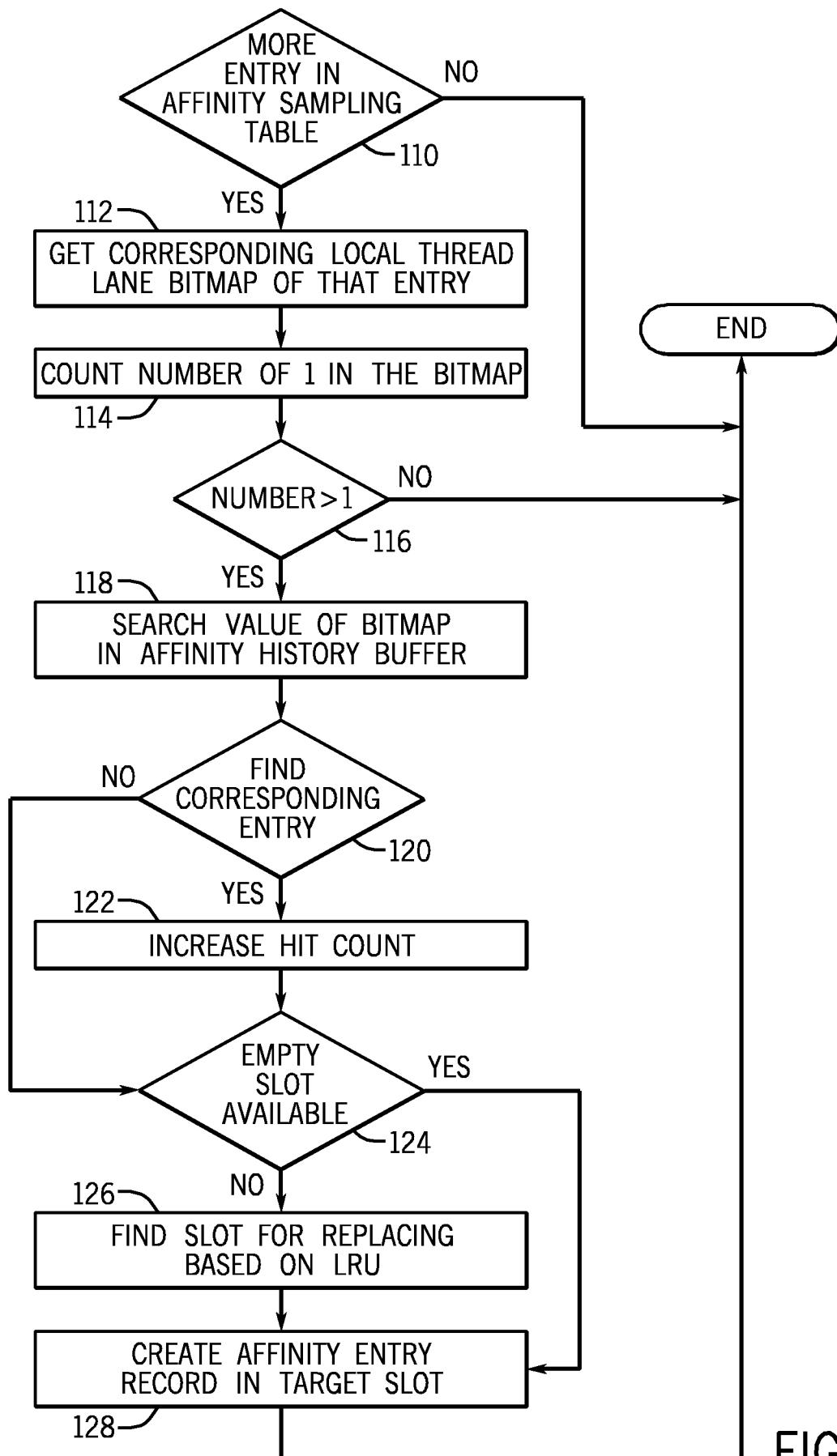
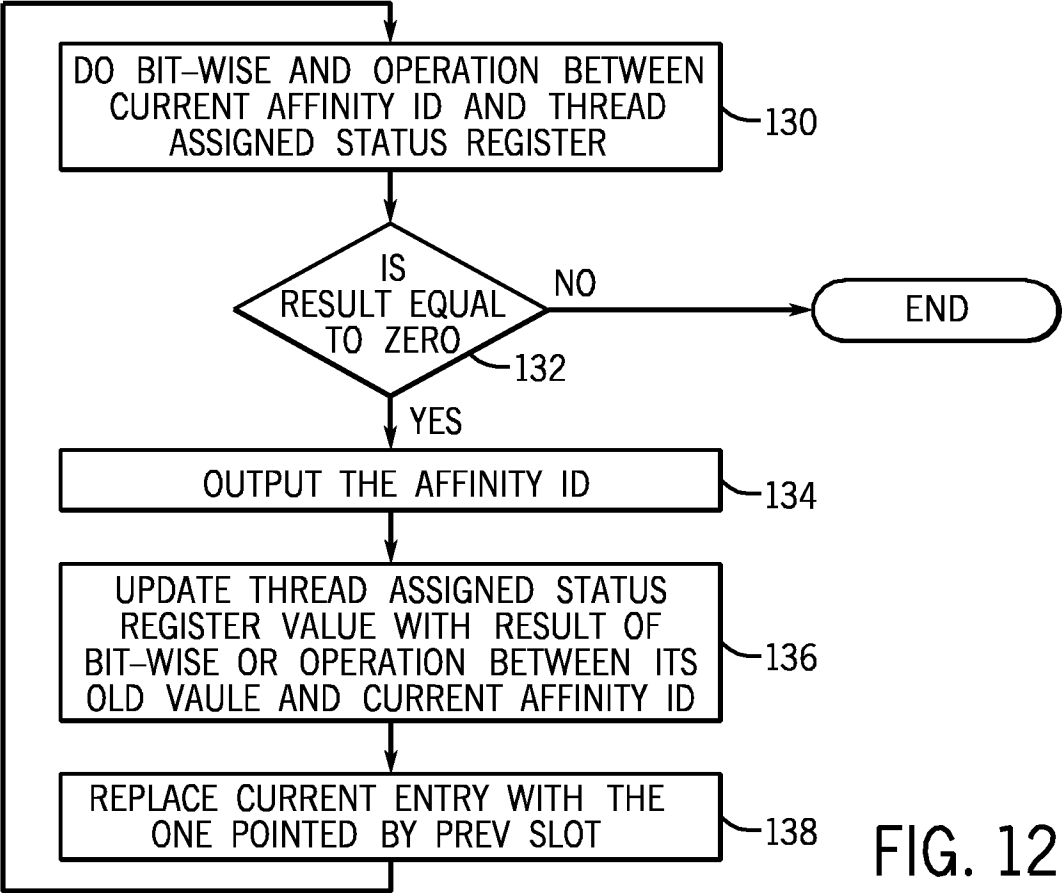
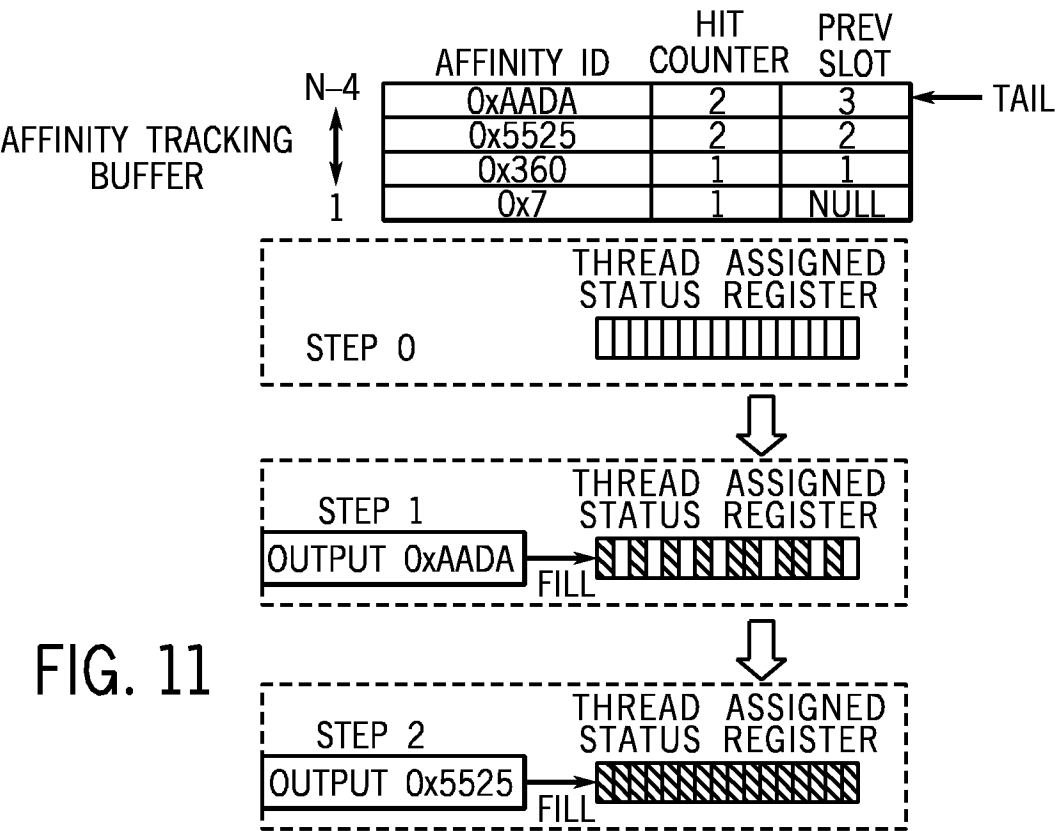


FIG. 10



INTERNATIONAL SEARCH REPORT

International application No.

PCT/CN2011/082215

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)

IPC: G06F 9/-

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)

Databases: DWPI; EPODOC; CNPAT; CPRSABS

Search terms: task, affinity, relation, relationship, graphics, thread

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CN101076783A(AB INITIO SOFTWARE CORP) 21 Nov. 2007 (21.11.2007) The whole document	1-21
A	CN101286138A(UNIV ZHEJIANG) 15 Oct. 2008(15.10.2008) The whole document	1-21
A	CN101615120A(UNIV ELECTRONIC SCI&TECHNOLOGY) 30 Dec. 2009(30.12.2009) The whole document	1-21

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

* Special categories of cited documents:	“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
“A” document defining the general state of the art which is not considered to be of particular relevance	“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
“E” earlier application or patent but published on or after the international filing date	“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
“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)	“&” document member of the same patent family
“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	

Date of the actual completion of the international search 25 Jun. 2012 (25.06.2012)	Date of mailing of the international search report 19 Jul. 2012 (19.07.2012)
Name and mailing address of the ISA/CN The State Intellectual Property Office, the P.R.China 6 Xitucheng Rd., Jimen Bridge, Haidian District, Beijing, China 100088 Facsimile No. 86-10-62019451	Authorized officer WANGYan Telephone No. (86-10)62411625

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2011/082215

Patent Documents referred in the Report	Publication Date	Patent Family	Publication Date
CN101076783A	21.11.2007	CN101076783B	07.07.2010
		JP2007528080A	04.10.2007
		IN200605108P1	24.08.2007
		AU2005221076B2	01.04.2010
		US2005210473A1	22.09.2005
		EP1735703A2	27.12.2006
		WO2005086842A3	15.03.2007
		US7467383B2	16.12.2008
		AU2005221076A1	22.09.2005
		WO2005086842A2	22.09.2005
CN101286138A	15.10.2008	None	
CN101615120A	30.12.2009	None	